Invited Paper: Heterogeneous Acceleration for Design Rule Checking

Zhuolun He Bei Yu
Department of Computer Science & Engineering
The Chinese University of Hong Kong

***Abstract*—The advances of heterogeneous CPU-GPU computing platforms have marked their great potential for algorithm acceleration. Yet, how to orchestrate such hybrid devices remains a concern for programmers and researchers. To obtain a desired performance gain, different strategies, such as parallel computing, heterogeneous scheduling, and data movement minimization, should be carefully considered and effectively combined. In this talk, we first review efforts for efficient design rule checking in the literature. Then, we introduce the parallel sweepline paradigm for design rule checking, and demonstrate how to accelerate design rule checking with such paradigm on heterogeneous CPU-GPU platforms.**

I. Introduction

Graphics Processing Units (GPUs) are ubiquitous nowadays as they offer massive parallel computing power to support modern applications like high performance computing and deep neural network execution. Conventionally, GPUs are considered as *accelerator*s, while CPUs are termed *host*s for managing I/O and scheduling in the context. On the other hand, CPUs and GPUs have their unique features, and hence the collaboration between them is inevitable to maximize system performance [1]. In fact, many top500 supercomputers [2] are quipped with GPU cores, such as Frontier [3], LUMI [4], and Leonardo [5]. In particular, Frontier uses 37,888 GPUs (8,335,360 cores) and 9,472 CPUs (606,208 cores) to achieve over one ExaFLOPS (10^{18}) peak performance. It is now clear that the CPU-GPU paradigm is promising and new heterogeneous computing techniques are required.

Recent years have seen massively parallel computing employed in electronic design automation (EDA) tools. Guo et al. have developed GPU accelerated static timing analysis (STA) [6] by parallelizing RC computation and timing propagation, achieving more than $3\times$ speedup compared with multithreaded OpenTimer [7]. Later on, Guo et al. developed GPU accelerated path-based STA and achieved significant speedup. Lin et al., Liu et al., and others have paid great efforts to GPU accelerated global placement, resulting in state-of-the-art academic placers DREAMPlace [8] and X-place [9]. They also try to address detailed placement with similar framework [10]. As for the routing stage, GPU accelerated pattern routing [11], maze routing [12], and Steiner tree construction [13] have been investigated. Other attempts include gate-level logic simulation [14], circuit simulation [15], logic optimization [16], capacitance extraction [17], electromagnetic analysis [18], and so on. It is believed that GPU acceleration has "opened up new directions for revisiting classical EDA problems with advancement in AI hardware and software" [8].

In this paper, we discuss heterogeneous acceleration for design rule checking. Design rule checking ensures geometric validity of layout, which is essential in physical design and physical verification. Design rule checking could be time-consuming due to the complexity and large number of rules, the large scale of the design, and the large times of repeated execution. Therefore, accelerating design rule checking is critical to reduce design cycle time.

The rest of the paper is organized as follows: Section II introduces related background. Section III reviews parallel computing and relevant techniques for design rule checking in the literature. Section IV recaps parallel sweepline paradigm for design rule checking. Section V discusses heterogeneous acceleration for design rule checking. Section VI concludes the paper.

II. Background

A. GPU and CUDA

GPU has gained tremendous development over past decades. In general, GPU is a device containing device memory, cache, and streaming multiprocessors. Streaming multiprocessors consist of arithmetic cores for arithmetic instructions and operations. It is connected with the CPU host and host memory through PCIe buses.

The CUDA toolkit is a computing platform for GPU programming. It gives direct access to GPU's abstract instruction set and parallel computing elements [19]. CUDA provides hierarchy of threads, where parallel kernels are composed of many threads executing the same sequential program, and threads are grouped into *thread blocks*. Threads inside the same block can communicate through shared memory. These threads and blocks have their unique IDs.

In a CUDA program, we refer to the CPU as *host* and the GPU as *device*, and both of them maintain separate memory spaces in their own DRAM. In the compilation flow, a host compiler (e.g., GCC) compiles the *host* code into an executable on the host, while Nvidia C Compiler (NVCC) cross-compiles the device code (i.e., those qualified

This work is supported in part by National Key R&D Program of China 2020YFA0711900, 2020YFA0711903, and Shanghai Artificial Intelligence Laboratory.

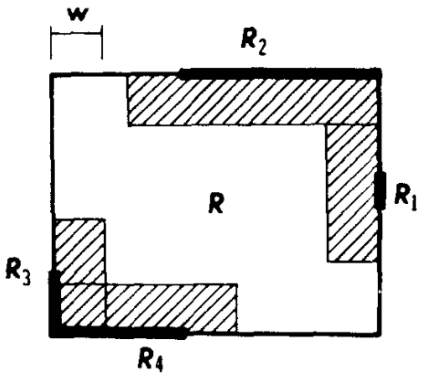


Fig. 1 Illustration of formal design rule definition by set operations. Adopted from [20].

by `__device__`) into CUDA binary, which will be handled by the CUDA runtime system whenever it is invoked from the host program.

B. Design Rule Checking

Design rules are constraints to layout objects imposed by manufacturing technology and issues. These rules are usually geometric shape and position requirements to improve chip yield. Minimal distance constraints are the most common constraints, which include, depending on the positional relation between objects, width rules, spacing rules, extension rules, enclosure rules, and so on. During physical design, detailed routers and even other tools in previous stages have to observe and conform to the design rules as much as possible. During physical verification, design rule checker must examine the whole layout and report existing design rule violations. Usually the violations have to be fixed before taping out.

III. Efficient Design Rule Checking: A Survey

In this section, we review efforts toward efficient design rule checking in the literature.

A. Algebraic Design Rule Checking

In the early stage, people wish to develop layout model and theoretical basis for VLSI design rule checkers in an algebraic way. Historically, there are majorly two approaches that try to define design rules in a formal way.

The first one by Modarres and Lomax [20] defines a layout model using set theory notations, also known as *mask-based*, where primitives are rectangles. Two set functions (`AND`, `DIFF`) are defined on layers. The core idea to present a design rule is to partition an edge (imagine that when another polygon intersects with the edge, it partitions the edge into several parts), and then use `EXPAND` or `SHRINK` operations to mark a *no-touch* area. Fig. 1 illustrates the first approach, where the no-touch areas are constructed by partially shrinking the rectangle with respect to partition (constructed by other set operations on edges) using the `SHRINK` operation.

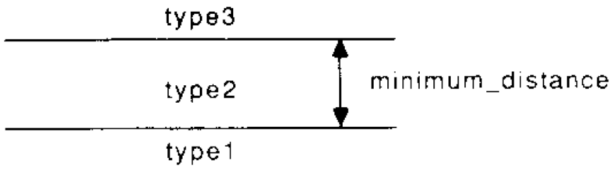


Fig. 2 Illustration of edge-based design rule description in the JCH model. Adopted from [21].

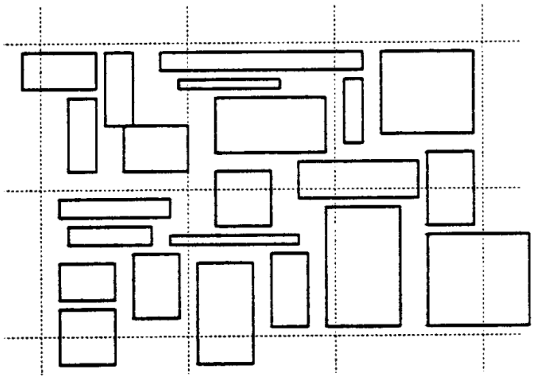


Fig. 3 Illustration of binning. Adopted from [24].

The second one (the JCH model [21]) is *edged-based*, which is later extended by Riepe and Sakallah [22]. In this setting, the *no-touch* areas are defined w.r.t. polygon edges. Fig. 2 illustrates the JCH model, where we use type1/type2, type3/type2 to denote a rule check on an edge pair, where type 2 is in the middle between the two edges, and the other side of the first edge (resp. the second edge) must be type1 (resp. type3). For example, $\bar{A}/A, \bar{A}/A$ represents a width check, since the required minimum distance of type 2, denoted by A, is inside an object. Similarly, $A/\bar{A}, A/\bar{A}$ represents an intra-layer spacing check as type 2 ($\bar{A}$) is outside objects. However, algebraic design rule checking focuses more on the conceptual definition of rules, rather than an efficient execution scheme.

B. Layout Data Structures

The straightforward way to represent layout objects is to store them in a large list, where scanning the lists takes linear time w.r.t. number of objects. Binning [23] is a popular technique by dividing the layout into tiles, as illustrated in Fig. 3. Many general spatial data structures have been utilized, such as quad-tree [25] and kd-tree [26]. R-tree and its variants store rectangles in a B^+-tree like data structure with heuristic optimization, which are widely adopted in layout representation. Clever customization is important to achieve good performance. Hinted quad tree [27] enables a multi-storage scheme, which accelerates *region query* without needing to start a search at the root of the tree. Fig. 4 illustrate

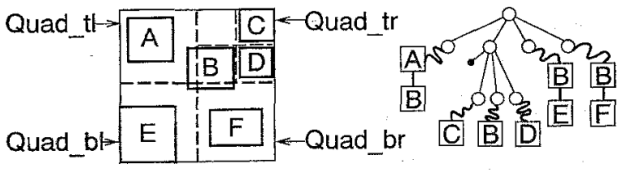


Fig. 4 Illustration of hinted quad tree that enables a multi-storage scheme. Adopted from [27].

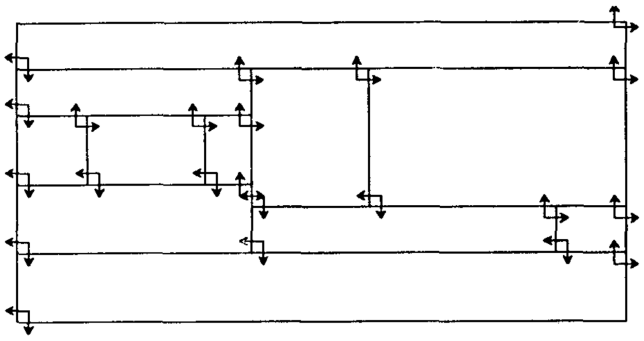


Fig. 5 Illustration of corner stitching that supports efficient incremental design rule checking. Adopted from [28].

such storage scheme. Corner stitching [24] is introduced to support efficient incremental design rule checking. In the corner stitching technique, rectangular objects are stitched together at their corners, and empty space is also explicitly represented, as illustrated in Fig. 5. Corner stitching supports efficient neighbour search, which is the fundamental requirement for Magic [28] and other interactive layout editors.

C. Parallel Design Rule Checking

There have been extensive efforts towards parallel design rule checking. Nandy [29] presented a distributed solution for DRC by exploiting either spatial independence or layer independence in layout data. Similarly, Hsu et al. [30] introduced partition based strategy for layout operations. The above two methods are considered region-based methods for spatial parallelism. Gregoretti and Segall [31] introduced a bottom up approach for DRC; Hedenstierna and Jeppson [32] extended the halo algorithm for parallel processing, both of which utilized design hierarchy for cell-level parallelism. Carlson and Rutenbar [33] developed a parallel mask verification algorithms for the Connect Machine. In their scanline approach, they first assign each edge a direction, which can be used to determine opaque regions and transparent regions. Then they use a set of counters to number regions, which are highly parallelizable and efficient for layout boolean operations (see Fig. 6). The whole scheme is edge-based for fine-grained parallelism with the sweepline algorithm, which is also adopted by FPGA design rule checker [34], and GPU accelerated X-check [35] and OpenDRC [36]. Marantz proposed in his master thesis [37] to decompose the

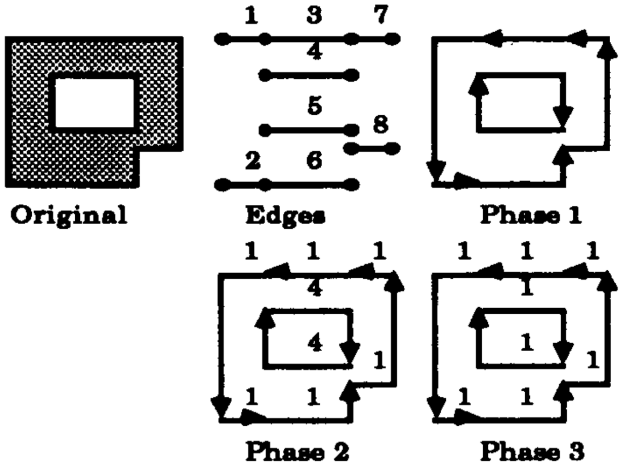


Fig. 6 Illustration of region numbering for (electrically) connected region recognition. Adopted from [33].

design rule checking program and separate the commands on different processors. This approach, understood as task parallelism, is in fact very practical since modern rule deck consists of a large number of rules. Macpherson and Banerjee proposed ProperDRC [38] that combines task parallelism and data parallelism.

In fact, these works gain benefits from different hardware platforms, including SIMD engines [39], multiprocessors [33], [38], GPU [35], [36], specialized hardware [34], [40], and distributed systems [29], [30], [37], [41].

TABLE I summarizes the parallel design rule checking algorithms and engines in the literature. It can be seen that coarse-grained parallelism are almost developed for distributed systems, while edge-based algorithms, enabling massively fine-grained parallelism, are suitable for multiprocessors, GPUs, and so on.

IV. Parallel Sweepline for DRC

For completeness, we recap the parallel sweepline scheme and its application for DRC introduced by *X-Check* [35].

A. Sweepline Algorithms

Sweepline algorithms can be conceptually regarded as moving a sweepline on the plane to process a set of points (a.k.a. *event point*s) one by one. Event points $p_i \in P$ are processed in a total order $\prec$: $P \times P \mapsto \{0, 1\}$. At each point, the algorithm builds an intermediate data structure $t_i \in T$ with the previous data structure t_{i-1} and the current point p_i using an *update function* $h : T \times P \mapsto T$ (i.e., $t_i = h(t_{i-1}, p_i)$). The initial prefix structure is t_0. In this way, we define a sweepline algorithm as a five tuple:

$$SW = \{P, \prec, T, t_0, h\}. \tag{1}$$

To describe a parallel sweepline algorithm, two further two operators are defined, a *fold function* $\rho : \langle P \rangle \mapsto T$ that converts a sequence of points to a prefix structure, and a *combine function* $f : T \times T \mapsto T$ that combines/reduces two

		Multiprocessor	GPU	Specialized Hardware	Distributed Systems
Data-Parallelism	Region-based				[29], [30]
	Hierarchy-based	[31]			[32]
	Edge-based	[33]	[35], [36]	[34], [40]	
Task-Parallelism					[37]
Task- and Data-Parallelism		[38]			

TABLE I Summary of parallel design rule checking.

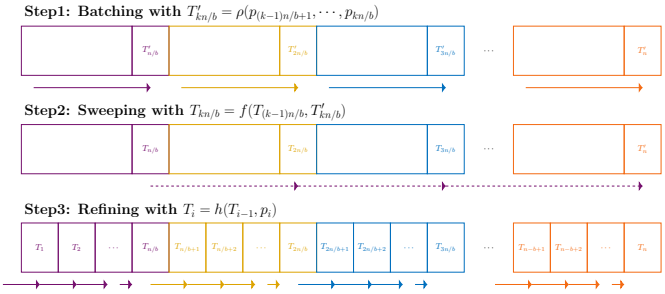


Fig. 7 Parallel prefix build for sweepline algorithms in three steps: batching, sweeping, and refining. Each rectangle block represents a prefix structure, where different colors indicate different blocks. Each colored arrow represents workload of a thread. Adapted from [42].

prefix structures. We require f to be associative. A parallel sweepline paradigm is defined as:

$$PSW = \{P, \prec, T, t_0, h, \rho, f\}. \tag{2}$$

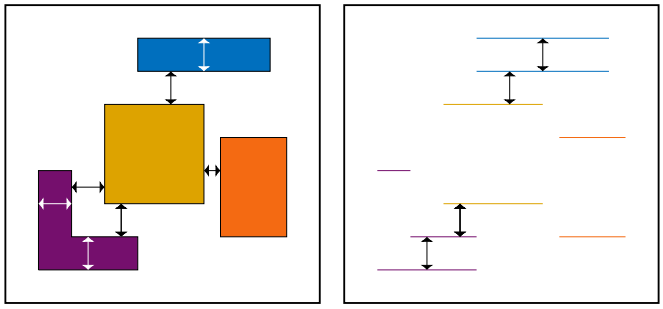
The essence of the parallel sweepline algorithm is to make use of the associativity of the *combine function* f. More precisely, repeatedly updating a sequence of points $\langle P \rangle$ into a sequence of prefix structures $\langle T \rangle$ using the *update function* h, is equivalent to first converting the points into (partial) prefix structures, and then combining the partial prefix structures using the *combine function* f. In [42], they propose to compute such prefix structures in three steps:

1) **Batching**. The inputs are sorted and evenly split into b blocks. Each thread converts the consecutive n/b points in one block into a partial sum (i.e., prefix) $T'_{kn/b}$ for $k = 1, 2, \cdots, b$ using the *fold function* ρ.
2) **Sweeping**. A single thread is invoked to sweep the b partial sums using the *combine function* f to compute the prefix structures $T_{n/b}, T_{2n/b}, \cdots, T_n$.
3) **Refining**. The rest of the prefix structures are built using the b prefix structures built in the second step. In each block, the points are processed sequentially to update the prefix structures using h. The b blocks can be done in parallel.

Fig. 7 illustrates the parallel prefix structure build.

B. Distance Check

X-Check aims to solve a general *distance check* problem.



(a) Distance Check (informal) (b) Distance Check

Fig. 8 Distance check. See Problems 1 and 2.

Problem 1 (Distance Check (informal))**.** *A layout can be seen as a set of axis-parallel polygonal objects. The distance rule says the following: any two edges must not be closer than a predefined minimal distance. A distance violation is a pair of edges in the layout that violate the distance rule. Given a layout, the task is to report all the distance violations.*

Without loss of generality, only horizontal segments are considered. A more formal definition of the above problem can be given:

Problem 2 (Distance Check)**.** *Given a set $\mathcal{H}$ of horizontal segments in $\mathbb{R}^2$, report the segment pairs from $\mathcal{H}^2$ whose horizontal projection is nonempty, and vertical distance is smaller than δ. Formally, we want to report:*

$$\begin{aligned} &\{([l_1, r_1] \times y_1, [l_2, r_2] \times y_2) \in \mathcal{H}^2\} \\ &\textit{s.t. } [l_1, r_1] \cap [l_2, r_2] \neq \emptyset, |y_1 - y_2| < \delta \end{aligned} \tag{3}$$

Fig. 8 illustrates the problem formulation.

C. Parallel Sweepline for Distance Check

Firstly, sort segments in ascending y-coordinates. We explain the algorithm by introducing the components in Equation (2).

- The event point set P includes all the y-coordinates of the segments.
- The total order $\prec$ is the total order $<$ on the y-coordinates.
- The prefix structure contains a set $\mathcal{S}$ of segments that are **below current segment within δ in y-direction**.
- The identity t_0 contains an empty set $\emptyset$.
- The *update function* h processes the segments by adding the segment to $\mathcal{S}$, and delete the segments that are below current segment by more than δ.

- For the *fold function* ρ, it suffices to first binary search for the lowest segment that is within δ to the highest segment, and then add the segments in between to the set $\mathcal{S}$.
- The *combine function* f is defined by first taking the union of the sets, and then delete the elements that are below the target segment by more than δ. Note that f is associative because the set operations are associative.

By the construction, the prefix structures contain all the candidate segments below each segment, in the sense that their distances in the y-direction are within δ. It remains to check whether each pair of segments overlap in the x-direction. Each violation will be reported by the algorithm exactly once. Algorithm 1 summarizes the vertical sweeping algorithm. Analysis shows that vertical sweeping runs in

Algorithm 1 Vertical Sweeping

Require: A set $\mathcal{H}$ of horizontal segments.
Ensure: Segment pairs that violate the distance rule.
1: Sort segments by ascending y-coordinates;
2: Partition the sorted segments into b blocks;
3: **For** each block **do in parallel** ▷ Batching
4: Find the lowest segment that is within δ to the highest segment in the block;
5: **Endfor**
6: Sweep the partial results among the b blocks; ▷ Sweep
7: **For** each block **do in parallel** ▷ Refine
8: Refine the prefix structures;
9: **Endfor**
10: **For** each prefix structure **do in parallel** ▷ Report
11: Report the violations in the prefix structure;
12: **Endfor**

$O(n \cdot \text{polylog}(n))$ work and $O(\sqrt{n} \cdot \text{polylog}(n))$ depth.

V. Heterogeneous Acceleration for DRC

We now discuss heterogeneous acceleration for design rule checking. Since recent works on DRC concentrate on simple rules like width and spacing, we would like to examine if the scheme works for complex rules. Specifically, we focus on two common and important spacing rules, namely the *Parallel Run Length* (PRL) spacing rule and the *End-of-Line* (EOL) spacing rule. These rules appear in ISPD'18 [43] and ISPD'19 [44] initial detailed routing contests.

A. PRL and EOL Spacing

In PRL spacing rule, a *spacing table* specifies the required spacing between objects, which depends on the parallel run length of the edges as well as the maximum width of both objects. Since big spacing requirement may be triggered when two wires run in parallel for a long distance, it is better to be avoided by the router. Fig. 9 illustrates the PRL rule.

In EOL spacing rule, the required spacing from a short edge is slightly larger than the common required spacing. Fig. 10 illustrates the EOL rule, where any overlap on the marked grey region is an EOL spacing violation.

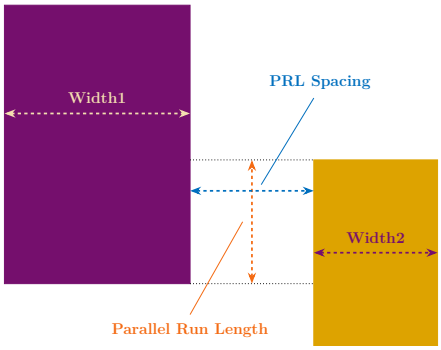


Fig. 9 Illustration of the Parallel Run Length spacing rule. The required spacing depends on parallel run length as well as the maximum width of both objects. Redrawn from LEF/DEF 5.8 Language Reference.

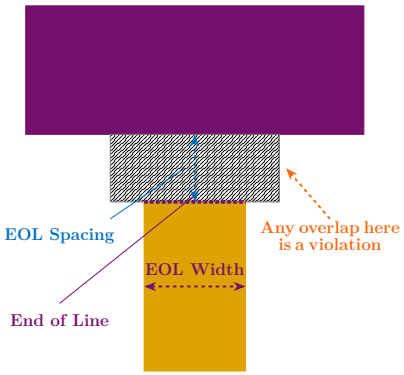


Fig. 10 Illustration of the EOL spacing rule. The required spacing is for short edges, and any overlap on the marked grey area is a violation. Redrawnz from LEF/DEF 5.8 Language Reference.

B. Parallel Sweepline for PRL and EOL Spacing

To accelerate the above rules, the first step is to investigate if the parallel sweepline scheme fits the rules. Intuitively, since the two rules are still 'spacing' rules, they should belong to the 'distance check' problems. As introduced in previous sections, distance check problems can be handled by the parallel sweepline paradigm. Having a closer look, it can be easily derived that the many sweepline components, including event points P, total order $\prec$, initial prefix structure t_0, combine function f, are directly reusable. The remaining operators involve the *'threshold'* δ, which is *ambiguous* in the context, since there exists different spacing requirements for different objects and cases. However, recall that the final decision of violation is not handled by the sweepline procedure itself; as long as enough information is provided, the sweepline framework still functions as a highly efficient filter to locate DRC region. In particular, suppose we use the largest spacing requirement among 1) the spacing rule, 2) the largest spacing value in the spacing table, and 3) the EOL spacing rule, then all kinds of spacing violation can still be captured by our sweepline framework.

Design	Size	Rows	CPU	GPU	GPU\S1	GPU\S2	GPU\S3	GPU\S4	GPU\S5	GPU\all
aes	294052	277	2128	189	187	193	213	192	186	202
ethmac	1007152	507	14318	436	441	455	430	520	440	534
ibex	303004	277	2404	187	194	197	194	202	193	212
jpeg	1182541	537	19692	455	465	480	458	526	503	575
sha3	301382	277	2298	180	178	192	177	185	191	217
Average			19.22	**1.00**	1.01	1.05	1.03	1.09	1.04	1.18

TABLE II Ablation study of heterogeneous acceleration. Runtimes are in ms.

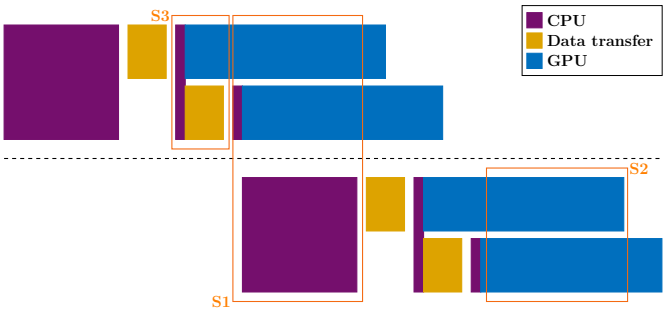


Fig. 11 Illustration of heterogeneous acceleration strategies S1, S2, and S3 in orange boxes.

C. Heterogeneous Acceleration

The motivation for heterogeneous acceleration includes both to improve resource usage and to match computation patterns of algorithms and of hardware devices. Here are some of the general strategies:

- Concurrent GPU computation and CPU computation.
- Concurrent GPU computation between streams.
- Overlap data transfer and computation.
- Minimize data transfer overhead.
- Avoid GPU invocation for small data batch.

OpenDRC [36] introduces an adaptive layout partition scheme that partitions a layout into independent rows. This partition scheme collects unique y coordinates of objects, runs an interval merging algorithm to decide partition positions, and finally performs DRC in a row-by-row manner. The partition scheme enables some design space for heterogeneous acceleration. As guided by the general strategies, we would like to consider the following techniques in the implementation of design rule checking:

- S1: GPU computation of the previous row and CPU preprocessing of the next row can be executed concurrently.
- S2: GPU computation for horizontal edges and vertical edges can be executed concurrently by different streams.
- S3: Data movement for one batch of data and GPU sorting of another can be overlapped.
- S4: Differentiate horizontal and vertical edges.
- S5: No GPU computation will be invoked if a row has only a limited number of objects.

Fig. 11 illustrates heterogeneous acceleration strategies S1, S2, and S3, where the purple boxes denote CPU execution, golden boxes denote data transfer, and blue boxes denote GPU execution. The dashed line separates the computation patterns for different rows. The orange rectangles highlights the heterogeneous acceleration. In fact, these three strategies bear a similar idea to improve concurrency, which leads to vertically overlapping rectangles in the pipeline figure.

D. Experimental Evaluation

Here we demonstrate experimental evaluation. All the designs are synthesized by the OpenROAD [45] flow. We run the spacing checks on the M1 layer of designs, and perform ablation study to examine the effectiveness of the above strategies. To avoid cold starts, we collect program runtime after a few warm up runs. TABLE II lists all the runtime results, where GPU\S[i] indicates removing the i-th strategy from the implementation. It can be seen that removing strategies each contributes to 1% to 9% speed-down compared with the complete implementation. Strategy four is the most significant one, as it directly affects the total amount of data movement between host and device. Nevertheless, even we remove all five strategies, the program is still $16.3\times$ faster than CPU implementation, which indicates the significance of GPU acceleration.

VI. Conclusion

Heterogeneous acceleration is no doubt a promising paradigm to archive high performance computing. Together with the prosperous development of artificial intelligence, the CPU-GPU hybrid platforms are ubiquitous and revolutionized the landscape of high performance computing. This paper discusses heterogeneous acceleration for design rule checking. We first reviews efforts to enable efficient design rule checking in the literature, especially the parallel sweepline paradigm for. Then we discuss how to extend the parallel sweepline paradigm to more complex design rules, including PRL and EOL design rules. Finally, we present strategies to improve the performance of heterogeneous acceleration for DRC, and demonstrate their effectiveness with ablation studies.

Recent years have seen successful applications of parallel and heterogeneous acceleration to EDA point tools. Acceleration does not only reduce runtime, but also indirectly improves PPA as more design iterations are possible within the same timing budget. However, improving tool quality with parallel computing requires more systematic investigation of comprehensive stages, as well as new and extensible methodologies to accommodate very complex algorithms and heursitics in reality. We wish to see more clever attempts along this line.

REFERENCES

[1] S. Mittal and J. S. Vetter, "A survey of cpu-gpu heterogeneous computing techniques," *ACM Computing Surveys*, vol. 47, no. 4, pp. 1–35, 2015.

[2] "TOP500," https://www.top500.org/lists/top500/2023/06/, accessed: 2023-08-24.

[3] "Frontier," https://www.olcf.ornl.gov/frontier/, accessed: 2023-08-24.

[4] "LUMI," https://www.lumi-supercomputer.eu/zw, accessed: 2023-08-24.

[5] M. Turisini, G. Amati, and M. Cestari, "LEONARDO: A Pan-European Pre-Exascale Supercomputer for HPC and AI Applications," *arXiv preprint arXiv:2307.16885*, 2023.

[6] Z. Guo, T.-W. Huang, and Y. Lin, "GPU-accelerated static timing analysis," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2020, pp. 1–9.

[7] T.-W. Huang and M. D. Wong, "OpenTimer: A high-performance timing analysis tool," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 895–902.

[8] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement," in *ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[9] L. Liu, B. Fu, M. D. Wong, and E. F. Young, "XPlace: an extremely fast and extensible global placement framework," in *ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 1309–1314.

[10] Y. Lin, W. Li, J. Gu, H. Ren, B. Khailany, and D. Z. Pan, "ABCDPlace: Accelerated batch-based concurrent detailed placement on multithreaded CPUs and GPUs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 12, pp. 5083–5096, 2020.

[11] S. Liu, Y. Pu, P. Liao, H. Wu, R. Zhang, Z. Chen, W. Lv, Y. Lin, and B. Yu, "FastGR: Global routing on CPU-GPU with heterogeneous task graph scheduler," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2022.

[12] S. Lin, J. Liu, E. F. Young, and M. D. Wong, "GAMER: GPU-Accelerated Maze Routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, no. 2, pp. 583–593, 2022.

[13] Z. Guo, F. Gu, and Y. Lin, "GPU-Accelerated Rectilinear Steiner Tree Generation," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2022, pp. 1–9.

[14] Y. Zhang, H. Ren, A. Sridharan, and B. Khailany, "Gatspi: GPU accelerated gate-level simulation for power improvement," in *ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 1231–1236.

[15] J. Zhao, Y. Wen, Y. Luo, Z. Jin, W. Liu, and Z. Zhou, "SFLU: Synchronization-Free Sparse LU Factorization for Fast Circuit Simulation on GPUs," in *ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 37–42.

[16] S. Lin, J. Liu, T. Liu, M. D. Wong, and E. F. Young, "NovelRewrite: node-level parallel AIG rewriting," in *ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 427–432.

[17] K. Zhai, W. Yu, and H. Zhuang, "GPU-friendly floating random walk algorithm for capacitance extraction of VLSI interconnects," in *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. IEEE, 2013, pp. 1661–1666.

[18] J. Guan, S. Yan, and J.-M. Jin, "An accurate and efficient finite element-boundary integral method with GPU acceleration for 3-D electromagnetic analysis," *IEEE Transactions on Antennas and Propagation*, vol. 62, no. 12, pp. 6325–6336, 2014.

[19] "Nvidia's CUDA: The End of the CPU?" https://www.tomshardware.com/reviews/nvidia-cuda-gpu,1954.html, accessed: 2023-08-26.

[20] H. Modarres and R. J. Lomax, "A formal approach to design-rule checking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 6, no. 4, pp. 561–573, 1987.

[21] K. O. Jeppson, S. Christensson, and N. Hedenstierna, "Formal definitions of edge-based geometric design rules," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 12, no. 1, pp. 59–69, 1993.

[22] M. A. Riepe and K. A. Sakallah, "The edge-based design rule model revisited," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 3, no. 3, pp. 463–486, 1998.

[23] J. L. Bentley and J. H. Friedman, "Data structures for range searching," *ACM Computing Surveys*, vol. 11, no. 4, pp. 397–409, 1979.

[24] J. K. Ousterhout, "Corner stitching: A data-structuring technique for VLSI layout tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 3, no. 1, pp. 87–100, 1984.

[25] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta informatica*, vol. 4, no. 1, pp. 1–9, 1974.

[26] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[27] G. G. Lai, D. S. Fussell, and D. Wong, "Hinted quad trees for VLSI geometry DRC based on efficient searching for neighbors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 15, no. 3, pp. 317–324, 1996.

[28] G. S. Taylor and J. K. Ousterhout, "Magic's incremental design-rule checker," in *ACM/IEEE Design Automation Conference (DAC)*, 1984, pp. 160–165.

[29] S. Nandy, "Geometric Design Rule Check of VLSI Layouts in Distributed Computing Environment," *International Conference on VLSI Design*, vol. 1, no. 2, pp. 155–167, 1994.

[30] K.-T. Hsu, S. Sinha, Y.-C. Pi, C. Chiang, and T.-Y. Ho, "A distributed algorithm for layout geometry operations," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2011, pp. 182–187.

[31] F. Gregoretti and Z. Segall, "Analysis and evaluation of VLSI design rule checking implementation in a multiprocessor," in *International Conference on Parallel Processing (ICPP)*, 1984, pp. 7–14.

[32] N. Hedenstierna and K. Jeppson, "A parallel hierarchical design rule checker," in *European Conference on Design Automation*. IEEE Computer Society, 1992, pp. 142–143.

[33] E. C. Carlson and R. A. Rutenbar, "Mask verification on the connection machine," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1988, pp. 134–140.

[34] Z. Luo, M. Martonosi, and P. Ashar, "An edge-endpoint-based configurable hardware architecture for VLSI layout Design Rule Checking," *International Conference on VLSI Design*, vol. 10, no. 3, pp. 249–263, 2000.

[35] Z. He, Y. Ma, and B. Yu, "X-check: Gpu-accelerated design rule checking via parallel sweepline algorithms," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2022, pp. 1–9.

[36] Z. He, Y. Zuo, J. Jiang, H. Zheng, Y. Ma, and B. Yu, "OpenDRC: An Efficient Open-Source Design Rule Checking Engine with Hierarchical GPU Acceleration," in *ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[37] J. D. Marantz, "Exploiting parallelism in VLSI CAD," 1986.

[38] K. MacPherson and P. Banerjee, "Parallel algorithms for VLSI layout verification," *Journal of Parallel and Distributed Computing*, vol. 36, no. 2, pp. 156–172, 1996.

[39] S. Koranne, "A high performance SIMD framework for design rule checking on Sony's PlayStation 2 Emotion Engine platform [IC layout]," in *International Symposium on Signals, Circuits and Systems*. IEEE, 2004, pp. 371–376.

[40] R. Kane and S. Sahni, "A systolic design rule checker," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1984, pp. 243–250.

[41] A. Pais, M. Anido, and C. Oliveira, "Developing a distributed architecture for design rule checking," in *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, vol. 2, 2001, pp. 678–681.

[42] Y. Sun and G. E. Blelloch, "Parallel range, segment and rectangle queries with augmented maps," in *Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2019, pp. 159–173.

[43] S. Mantik, G. Posser, W.-K. Chow, Y. Ding, and W.-H. Liu, "ISPD 2018 initial detailed routing contest and benchmarks," in *ACM International Symposium on Physical Design (ISPD)*, 2018, pp. 140–143.

[44] W.-H. Liu, S. Mantik, W.-K. Chow, Y. Ding, A. Farshidi, and G. Posser, "ISPD 2019 initial detailed routing contest and benchmark with advanced routing rules," in *ACM International Symposium on Physical Design (ISPD)*, 2019, pp. 147–151.

[45] 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 *ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–4.