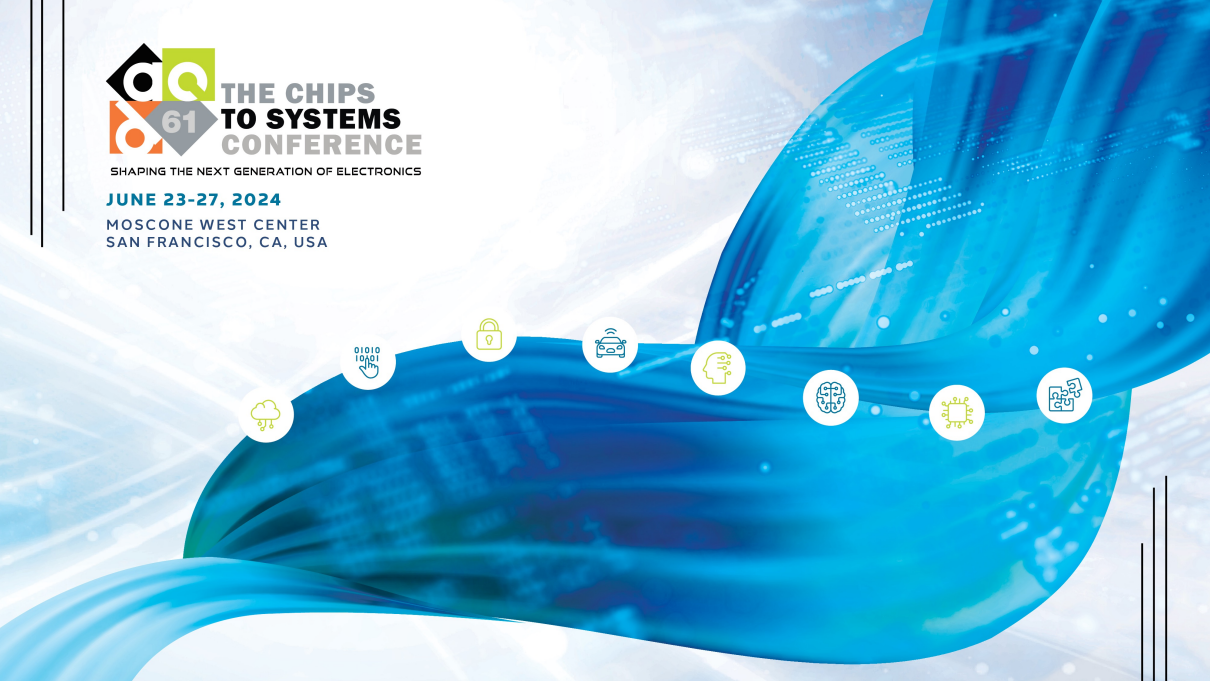

61
THE CHIPS
TO SYSTEMS
CONFERENCE
SHAPING THE NEXT GENERATION OF ELECTRONICS
JUNE 23-27, 2024
MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

61 THE CHIPS TO SYSTEMS CONFERENCE
SHAPING THE NEXT GENERATION OF ELECTRONICS
JUNE 23-27, 2024
MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

PDRC: Package Design Rule Checking via GPU-Accelerated Geometric Intersection Algorithms for Non-Manhattan Geometry

Jiaxi Jiang[1], Lancheng Zou[1], Wenqian Zhao[1], Zhuolun He[1], Tinghuan Chen[2], Bei Yu[1]

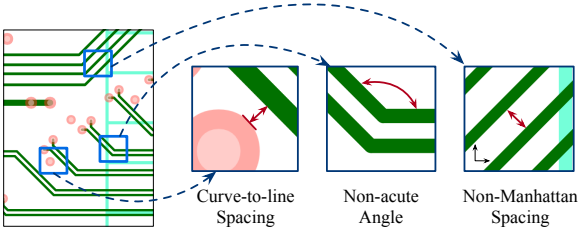
[1]Chinese University of Hong Kong
[2]Chinese University of Hong Kong, Shenzhen

Outline

Introduction

Non-Manhattan geometry: X or any-angle shapes used in routing[1][2].
Design rule checking (DRC): Verify a design's layout against geometric rules.

The increasing size of package designs may impact the efficiency of DRC.



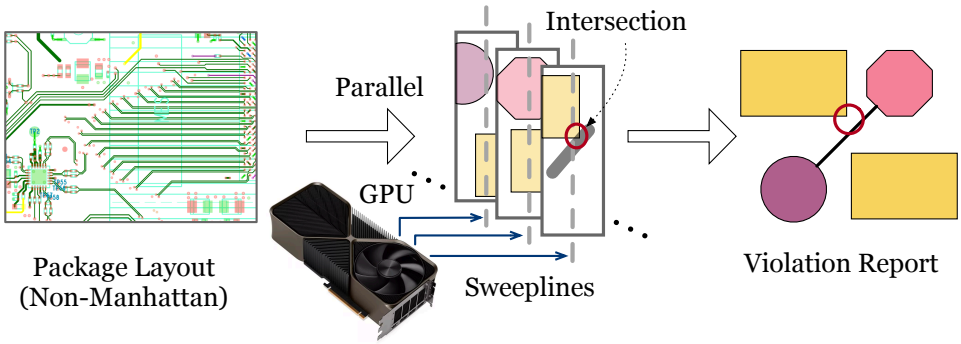
Non-Manhattan geometry and design rules

[1]M.-H. Chung *et al.*, "Any-Angle Routing for Redistribution Layers in 2.5D IC Packages", in *Proc. DAC*, 2023, pp. 1–6.

[2]T. Chen *et al.*, "TRouter: Thermal-driven PCB Routing via Non-Local Crisscross Attention Networks", *IEEE TCAD*, 2023.

Overview:PDRC

PDRC deals with package/PCB designs featuring **non-Manhattan geometry**, expanding layout shapes and employing **GPU-accelerated geometric intersection algorithms** to finish design rule checking.



From a computational geometry perspective

Previous researches use computational geometry tools to tackle design rule checking[3][4][5].

partition the input space and data

partition space Quad-tree recursively partitions the layout space.

partition data R-tree partitions layout objects, minimizing coverage and overlap among subnodes.

spatial order The spatial order of line segments is determined by the coordinates of their intersection points with the sweepline.

---

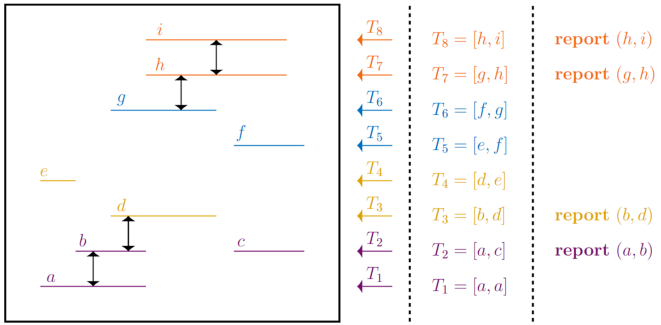
[3] G. G. Lai *et al.*, "Hinted quad trees for VLSI geometry DRC based on efficient searching for neighbors", *IEEE TCAD*, vol. 15, no. 3, pp. 317–324, 1996.

[4] A. Guttman, "R-trees: A dynamic index structure for spatial searching", in *Proc. SIGMOD*, 1984, pp. 47–57.

[5] Bentley and Ottmann, "Algorithms for reporting and counting geometric intersections", *IEEE TC*, vol. 100, no. 9, pp. 643–647, 1979.

Special case:Manhattan geometry

X-Check[6] utilizes the y-coordinates of horizontal segments to establish order, akin to partitioning the layout along the y-axis.

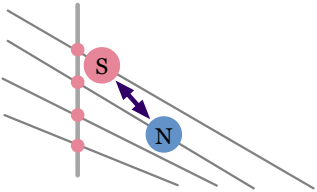


(a) Vertical Sweeping

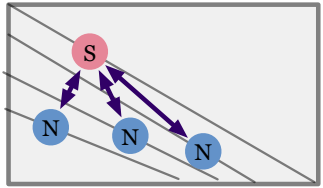
[6]Z. He *et al.*, "X-Check: GPU-Accelerated Design Rule Checking via Parallel Sweepline Algorithms", in *Proc. ICCAD*, 2022, pp. 1–9.

A rough comparison

sweepline: The efficiency of the sweepline algorithm is attained by limiting the search space to immediate neighbors.
bounding box: The partitioning strategy employs bounding boxes to approximate diagonal lines, creating empty spaces that reduce pruning efficiency.



Sweepline



Bounding Box

Geometric intersection with sweepline

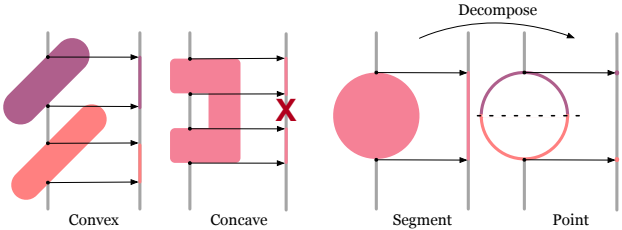
Uniqueness Any vertical (horizontal) line through the objects intersects the object exactly once.

Orderliness For objects intersecting the line, they can have a total order.

Computability Given two objects, it is possible to compute the intersection point.

All convex polygons exhibit *Uniqueness* property.

To have an *order*, points are preferred over segments.



Issues come from decomposition

Decomposition prevents the detection of **inclusion**.

Reassemble the decomposed shapes by connecting the points.

Evaluate if any overlap exists between line segments on the same sweepline to perform an **inclusion check**.

By **sorting and scanning** the endpoints of line segments, we can efficiently detect overlapping segments with a single scan.

Iterative parallel sweepline algorithm

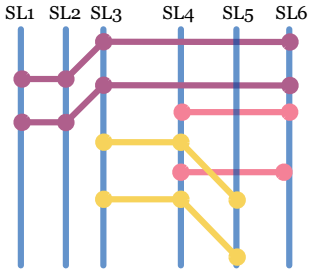
event Starting or ending points of segments, or intersection points.

Sweepline processes all events in order, **maintaining the order** of intersecting objects with the sweepline using a **binary tree**, and utilizing a **priority queue** to **maintain the order of all events**.

Observation[7] that concurrently processing events yields the same results as sequential execution.

[7] A. Paudel and S. Puri, "Openacc based gpu parallelization of plane sweep algorithm for geometric intersection", in *Proc. WACCPS*, 2019, pp. 114–135.

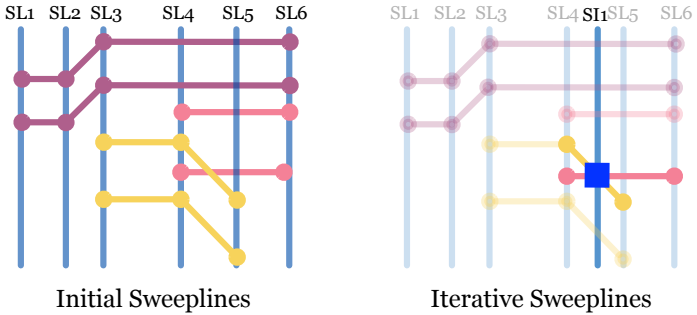
Iterative parallel sweepline algorithm



Initial Sweeplines

- The **initial sweeplines** are generated based on the x-coordinates of line segments

Iterative parallel sweepline algorithm

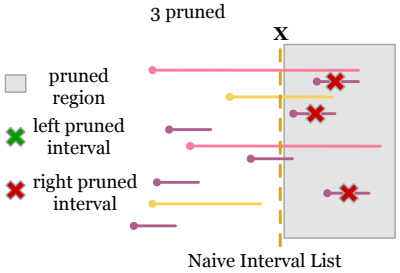


Initial Sweeplines

Iterative Sweeplines

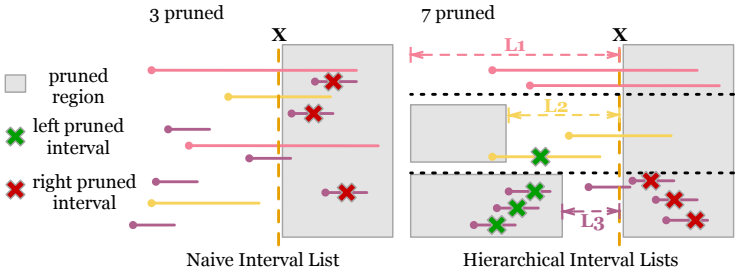
- The **initial sweeplines** are generated based on the x-coordinates of line segments
- While **iterative sweeplines** are produced from intersection points.

Hierarchical Interval Lists for GPU



- Efficiently determining sets of segments intersected by the sweepline at each position requires **stab queries** for line segments projected along the *x*-axis.

Hierarchical Interval Lists for GPU



- Efficiently determining sets of segments intersected by the sweepline at each position requires **stab queries** for line segments projected along the *x*-axis.
- We assign distinct **hierarchical labels** to line segments based on their lengths, allowing for the organization of segments from each hierarchy into separate interval lists.

Iterative parallel sweepline algorithm

For each event's position,

Label Identify the left and right boundaries in interval lists $\mathcal{L}$.

Merge Merge the segment sequence as $\mathcal{S}$ from $\mathcal{L}$.

Sort Sort segments in $\mathcal{S}$ based on the y-coordinates where they intersect with the event sweepline.

Check Scan segments and update intersection events.

Iterative Check Choose merged segments, sort and scan.

Complexity analysis

Label Binary searches across k interval lists with a total of n intervals yield a depth of $O(\log \frac{n}{\omega_k})$. The total work required is at most $O(k \log \frac{n}{k})$ for each position.

Merge In the worst-case scenario, the depth reaches $O(\sqrt{n})$, leading to a total work of $O(p\sqrt{n})$ across p positions.

Sort Parallel radix sorting, when applied to each list, achieves a depth of $O(d \log n)$, where d denotes the number of digits, with the total work $O(dp\sqrt{n})$.

Check Each position requires a single scan through the interval list for inclusion and intersection checks, leading to a depth of $O(\sqrt{n})$ and a work of $O(p\sqrt{n})$.

Iterative Check When few intersections occur, the "Iterative Check" stage requires $O(1)$ iterations.

Benchmarks

Our benchmarks include some industrial PCB designs.

Table: The statistics of our benchmarks.

Benchmark	#C	#P	#N
xc7z020_t	443	1737	428
xc7z020_b	572	1390	383
xc7z030_t	447	1936	442
xc7z030_b	653	1539	416
hs3690_t	910	3529	998
hs3690_b	656	1878	496

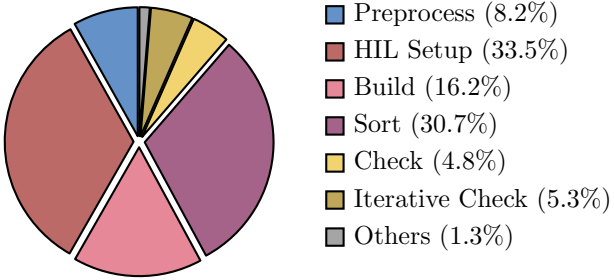
The statistics of our benchmarks are listed on Table 1, where #C, #P, #N denote the numbers of components, pads and nets respectively.
Additionally, we replicate our largest benchmark, by factors of 4, 8, and 16.

Runtime comparisons

Benchmark	#Segments	Klayout flat		Klayout deep		Klayout tile		R-tree boost		PDRC
		RT	Ratio	RT	Ratio	RT	Ratio	RT	Ratio	
xc7z020_t	39368	301	43.0×	272	38.9×	146	20.9×	82	11.7×	**7**
xc7z020_b	18014	232	77.3×	173	57.7×	65	21.7×	31	10.3×	**3**
xc7z030_t	45972	273	91.0×	275	91.7×	108	36.0×	90	30.0×	**3**
xc7z030_b	19500	235	78.3×	189	63.0×	69	23.0×	217	72.3×	**3**
hs3690_t	68604	825	165.0×	705	141.0×	331	66.2×	129	25.8×	**5**
hs3690_b	35082	452	150.7×	571	190.3×	192	64.0×	64	21.3×	**3**
4hs3690_t	274416	3334	222.3×	2772	184.8×	569	37.9×	1113	74.2×	**15**
4hs3690_b	140328	1849	205.4×	2240	248.9×	358	39.8×	532	59.1×	**9**
8hs3690_t	548832	6721	268.8×	5636	225.4×	1064	42.6×	3693	147.7×	**25**
8hs3690_b	280656	3731	186.6×	4445	222.3×	612	30.6×	1715	85.8×	**20**
16hs3690_t	1097664	13470	244.9×	11274	205.0×	1996	36.3×	12821	233.1×	**55**
16hs3690_b	561312	7505	220.7×	8929	262.6×	1136	33.4×	5944	174.8×	**34**
Average			143.0×		136.8×		35.3×		51.2×	

Table: Runtime (ms) comparisons of design rule checking(spacing).

Average runtime breakdown



The construction of **hierarchical interval lists** (“HIL”) and **sweepline statuses** (“Build” and “Sort”) account for the majority of the time.
Owing to the limited number of intersection checks needed by sweepline algorithms, **Check** accounts for a small portion of the total runtime.

Conclusion and Roadmap

We've implemented an **iterative parallel sweepline algorithm** optimized for GPUs using **position-level parallelism**, and have efficiently conducted **interval stabbing queries** using **hierarchical interval lists**.

In the future, we plan to develop more **work-depth-efficient** algorithms and **GPU-based data structures**.

THE CHIPS TO SYSTEMS CONFERENCE

SHAPING THE NEXT GENERATION OF ELECTRONICS

JUNE 23-27, 2024

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

THANK YOU!