Parallel *kd*-tree with Batch Updates

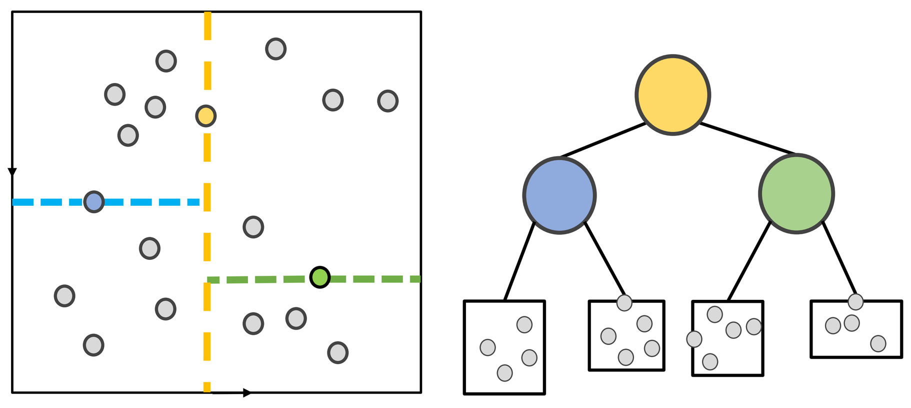
Ziyang Men, Zheqi Shen, Yan Gu, Yihan Sun

University of California, Riverside

Paper

Code

What is a *kd*-tree?



Build a *kd*-tree [Bentely' 79]:

1. Find the median of the objects.
2. Partition into two parts.
3. Recursive.

Fully balanced tree $O(n \log n)$

Why a new parallel *kd*-tree?

Challenges:

- Conventionally, *kd*-trees are maintained fully-balanced and is hard to support efficient updates while maintaining balance. Existing parallel update algorithms either fully rebuild the tree or sacrifice query performance.
- High performance also requires strong guarantee in work (time complexity), span (parallelism) and I/O (cache) complexity.

Tree	Build	Batch Insert (1%)	Batch Delete (1%)	10-NN (1%)	Range Report (10K)
Uniform-2D-1000M					
Ours	3.15	.104	.121	.381	.391
LOG	37.9	2.16	.396	2.96	2.62
BHL	31.7	31.4	30.9	.487	2.06
CGAL	1147	1660	41.2	1.04	311
Varden-2D-1000M					
Ours	3.66	.055	.049	.172	.382
LOG	34.2	2.01	1.06	2.05	2.63
BHL	30.2	29.4	29.0	.242	1.95
CGAL	429	849	13.0	.511	296

I/O inefficiency. Rebuild the whole tree to keep full balanced. Query on logarithmic trees.

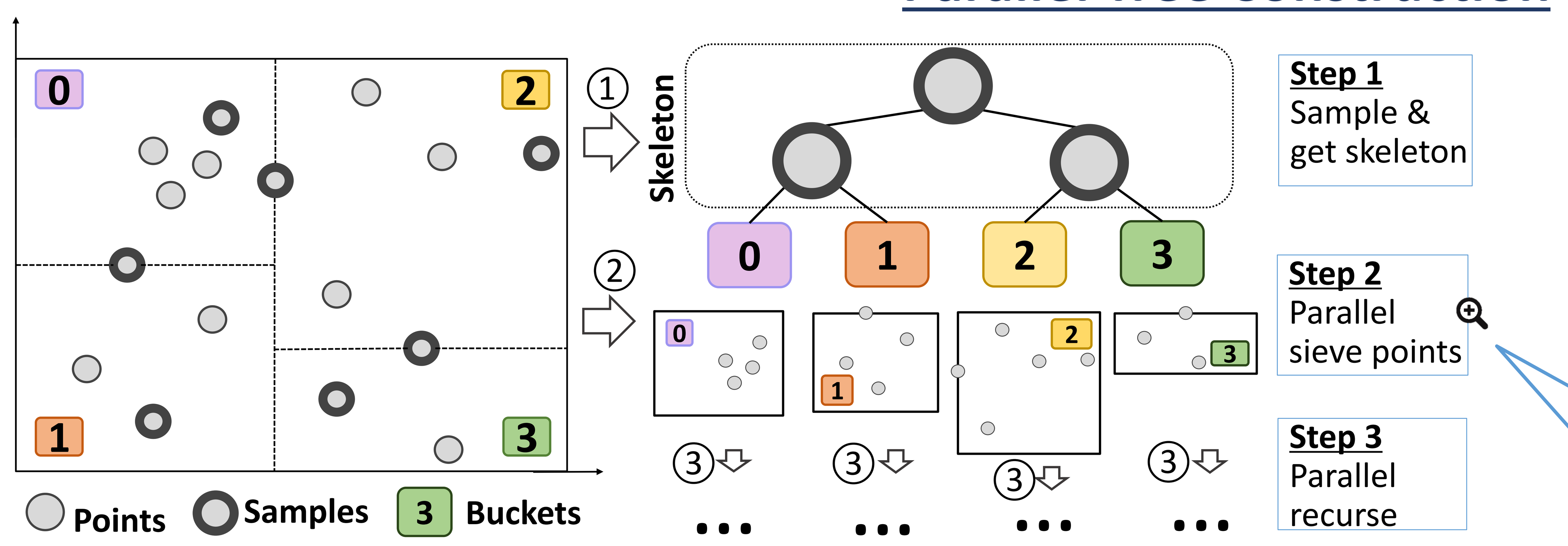
Our Contribution:

Propose the Pkd-tree that is highly parallel, I/O-efficient, and can support efficient updates.

Techniques:

1. An efficient tree construction algorithm *optimizes* work, span and I/O complexity.
2. A *reconstruction-based* update algorithm guarantees the tree to be weight-balanced.
3. A highly *efficient* implementation.

Parallel Tree Construction



General Idea

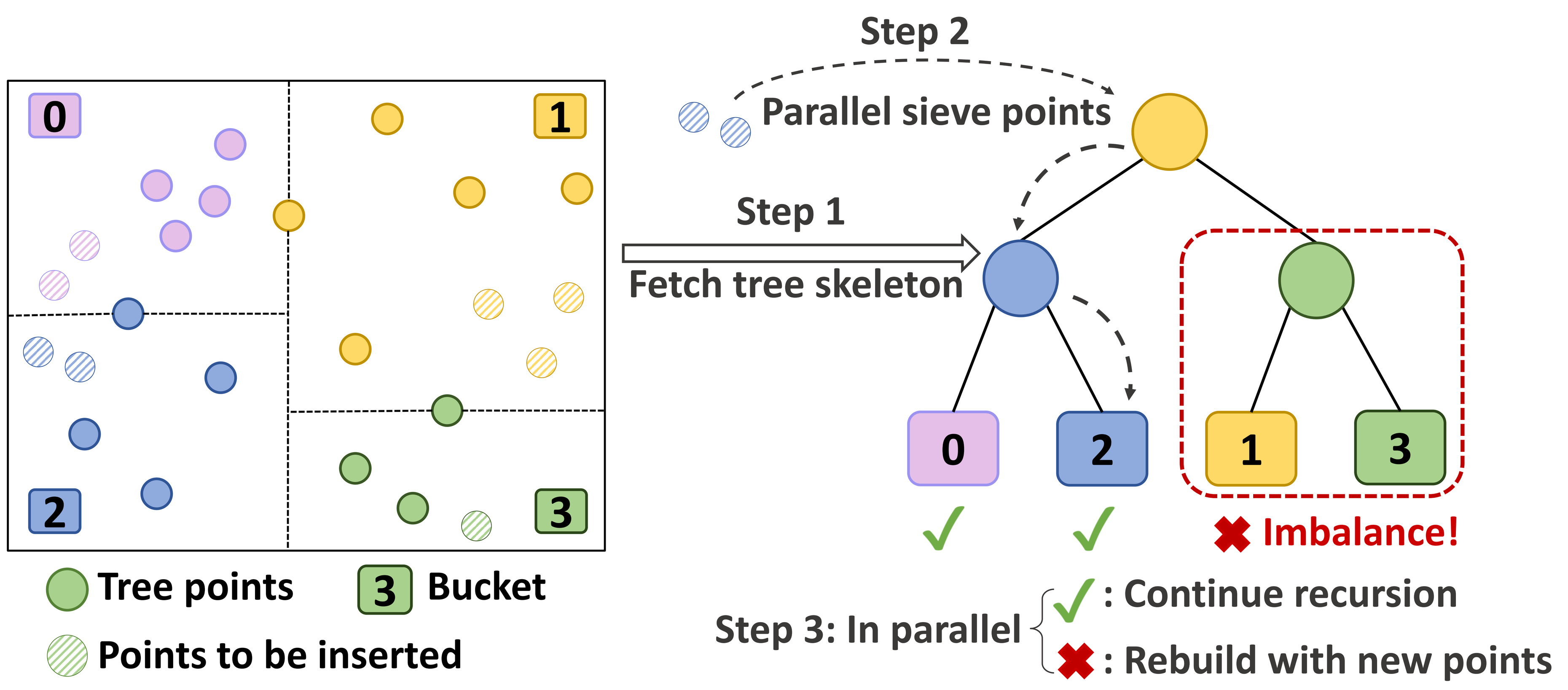
1. A slightly *unbalanced* tree.
   - $\log n + O(1)$ tree height or $O(\log n)$ height.
   - Use *samples* to estimate the median.
2. Build *multiple* levels at once
   - One round of data movement is sufficient.
   - I/O efficient *points sieving algorithm* to partition points.

Input	a	b	c	d	e	f	g	h	i
Bucket	0	2	3	1	2	1	0	3	1

Output	a	g	d	f	i	b	e	c	h
Bucket	0	0	1	1	1	2	2	3	3

- Parallel is hard, e.g., avoid data race, keep I/O efficiency …
- Borrow ideas from the cache-efficient parallel sorting [SPAA' 20]

Parallel Batch Updates



Handling of imbalance

1. The *kd*-tree *does not* support rotation.
   - Instead, we choose to rebuild the imbalanced sub-tree.
   - Known as *partial rebuild* [SSBM' 83].
2. Use *weighted-balanced* scheme.
   - Allow the tree to be off from perfectly balanced by a factor of α, where $0 < \alpha < 1$.
   - Otherwise, rebuilds the sub-tree.

The cost of partial rebuild can be *amortized* to tree nodes visit.

- $O(\log^2 n)$ work per element.

Experiments

Machine: 96 cores, 192 hyper-threads, 1.5 TB main memory, code in C++.

Real-world datasets

	Points	Dims	Op.	Ours	Log-tree	BHL-tree	CGAL
HT	928K	10	Build	.008	.678	.061	.472
			1-NN	.008	2.53	.015	.015
			10-NN	.043	2.81	.059	.020
			Range	.095	.651	.478	1.38
HH	2.04M	7	Build	.054	.716	.102	t.o.
			1-NN	.058	1.26	1.60	-
			10-NN	.229	2.40	3.19	-
			Range	.080	.819	.564	-
CHEM	4.21M	16	Build	.059	7.07	.786	2.52
			1-NN	.042	16.1	.123	.035
			10-NN	3.53	17.3	3.32	3.95
			Range	.412	4.28	2.64	3.14
GL	24M	3	Build	.256	1.34	.792	s.f.
			1-NN	.274	3.74	1.31	-
			10-NN	.775	14.4	9.37	-
			Range	.192	1.40	1.30	-
CM	321M	3	Build	1.54	16.7	13.3	184
			1-NN	2.79	25.9	5.24	5.94
			10-NN	9.09	s.f.	s.f.	33.0
			Range	.136	1.88	1.63	26.0
OSM	1298M	2	Build	5.08	51.3	56.6	497
			1-NN	8.73	134	13.0	10.5
			10-NN	16.5	214	30.6	22.6
			Range	.107	4.87	3.80	62.9

Table 1: Tree construction and query time on read-world datasets for Pkd-tree and baselines. *Lower is better.* k-NN queries are performed in parallel on all points in the dataset. "Range" is the time for 1000 range report queries with output size between 10^4-10^6. The fastest runtime for each benchmark is underlined. "s.f.": segmentation fault. "t.o.": time out (more than 3 hours).

Synthetic datasets

- **Uniform**: points are uniformly distributed
- **Varden**: points are skewed distributed

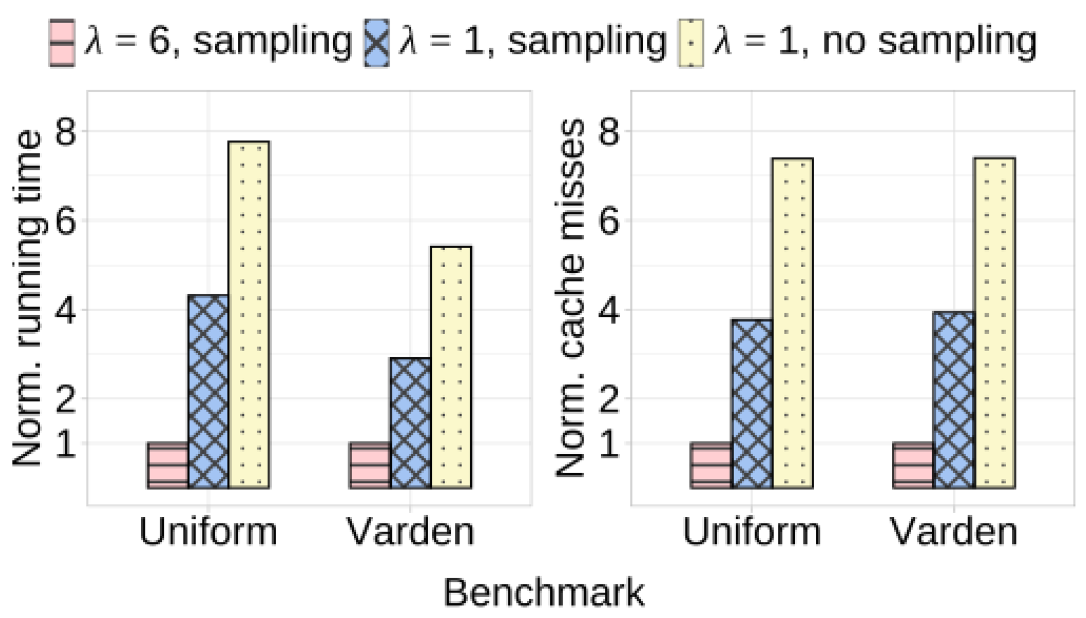


Figure 1: The evaluation of the performance gain for techniques in tree construction. *Lower is better*. The datasets have 1000M size in 3 dimensions. The y-axis normalized to the final version that builds 6 levels at once and using sampling.

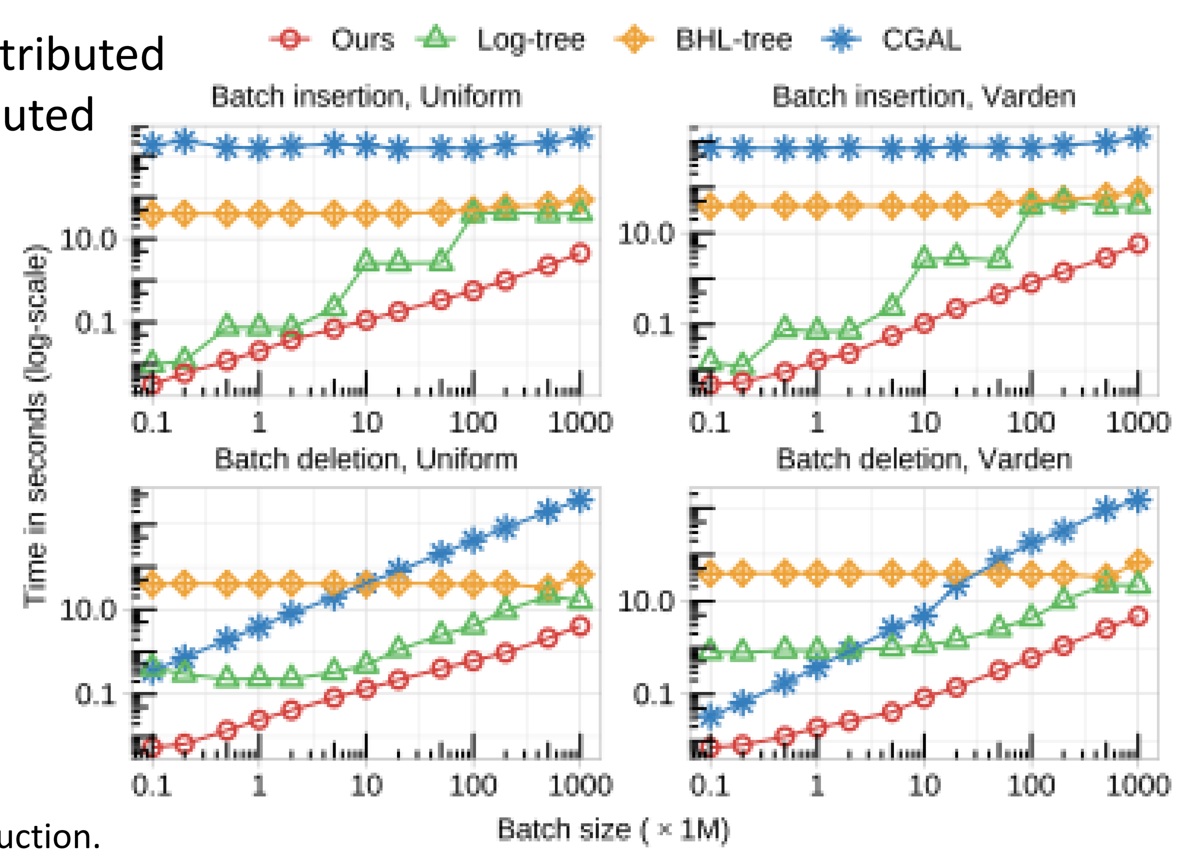


Figure 2: Time required for batch update on points from Varden and Uniform on a tree with 1000M points in 3 dimensions. *Lower is better.*

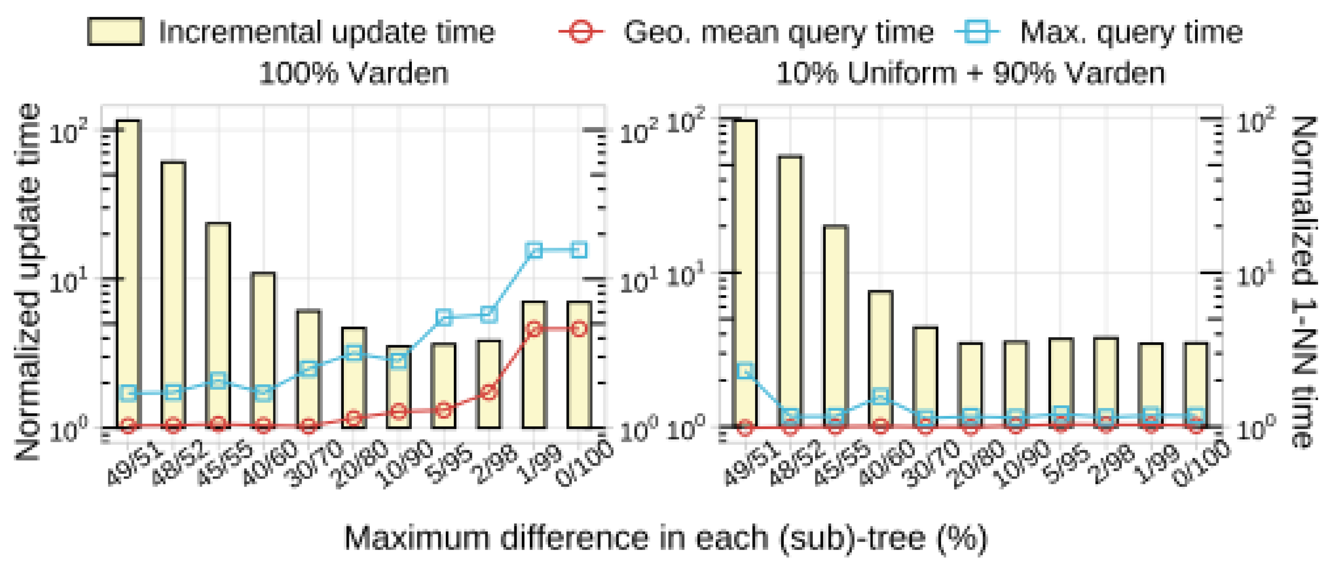


Figure 3: Incremental update time and query time with different imbalance ratio α. *Lower is better*. The dataset contains 1000M points in 3D, divided into 1000 batches, and incrementally inserted into an initially empty tree. We perform 1-NN queries in Uniform distribution after each insertion.

References:

[Bentely' 79]: Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. Commun. ACM 18, 9 (1975), 509–517.
[SSBM' 83]: Mark H Overmars. 1983. The design of dynamic data structures. Vol. 156. Springer Science & Business Media.
[CGAL' 20]: The CGAL Project. 2020. CGAL User and Reference Manual (5.1 ed.). CGAL Editorial Board. https://doc.cgal.org/5.1/Manual/packages.html.
[SPAA' 20]: Guy E. Blelloch, Phillip B. Gibbons, and Harsha Vardhan Simhadri. 2010. Low depth cache-oblivious algorithms. In ACM Symposium on Parallelism in Algorithms and Architectures (SPAA).
[SIGMOD' 22]: Yiqiu Wang, Shangdi Yu, Laxman Dhulipala, Yan Gu, and Julian Shun. 2022. ParGeo: a library for parallel computational geometry. In European Symposium on Algorithms (ESA).