10th International Conference on Model and Data Engineering
21-23 June 2021
Tallinn, Estonia

GPU-BASED ALGORITHMS FOR PROCESSING THE K NEAREST-NEIGHBOR QUERY ON DISK-RESIDENT DATA

POLYCHRONIS VELENTZAS[1]*, MICHAEL VASSILAKOPOULOS[1] AND ANTONIO CORRAL[2]

[1] DEPT. OF ELECTRICAL & COMPUTER ENGINEERING, UNIVERSITY OF THESSALY, VOLOS, GREECE

[2] DEPT. OF INFORMATICS, UNIVERSITY OF ALMERIA, SPAIN

(*) SPEAKER

GOBIERNO DE ESPANA
MINISTERIO DE ECONOMIA Y COMPETITIVIDAD

TIN2017-83964-R
MINECO research project

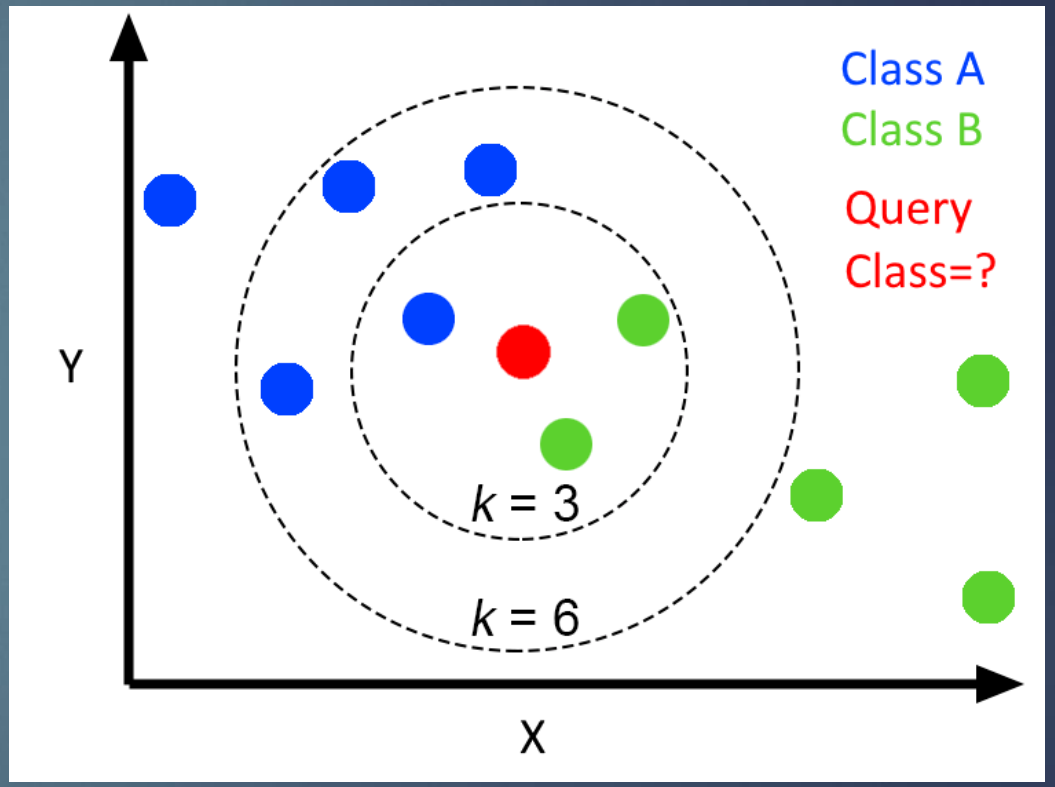
Outline

GPU-based Algorithms for Processing the k Nearest-Neighbor Query on Disk-resident Data

- The k-NN Query
- Problem and Motivation
- Paper Contributions
- Algorithmic Characteristics
- Algorithm of the k-NN Disk Brute-Force Query
- Algorithm of the k-NN Disk Plane-sweep Query
- Experimental Evaluation
- Conclusions and Future Work

The k Nearest Neighbor Query

- Between two spatial datasets (**Query** and **Reference**) it retrieves the **K points of Reference** with the **smallest** distance to **every point** of the Query dataset
- Used in AI, Regression, Classification
- Applications: medicine, economy, entertainment



Problem and Motivation

- The **k Nearest Neighbor** (kNN) query belongs to **Nearest Neighbor searches**, which have numerous modern applications, like in GIS systems, mobile computing, clustering, outlier detection, etc.
- There have been several attempts to effectively solve this query so far **(using the parallelism of GPU devices)**, but **these attempts** do not
  - focus in large reference datasets
  - maximize the utilization of the disk resident data (using SSD/HDD-resident reference datasets)

Paper Contributions

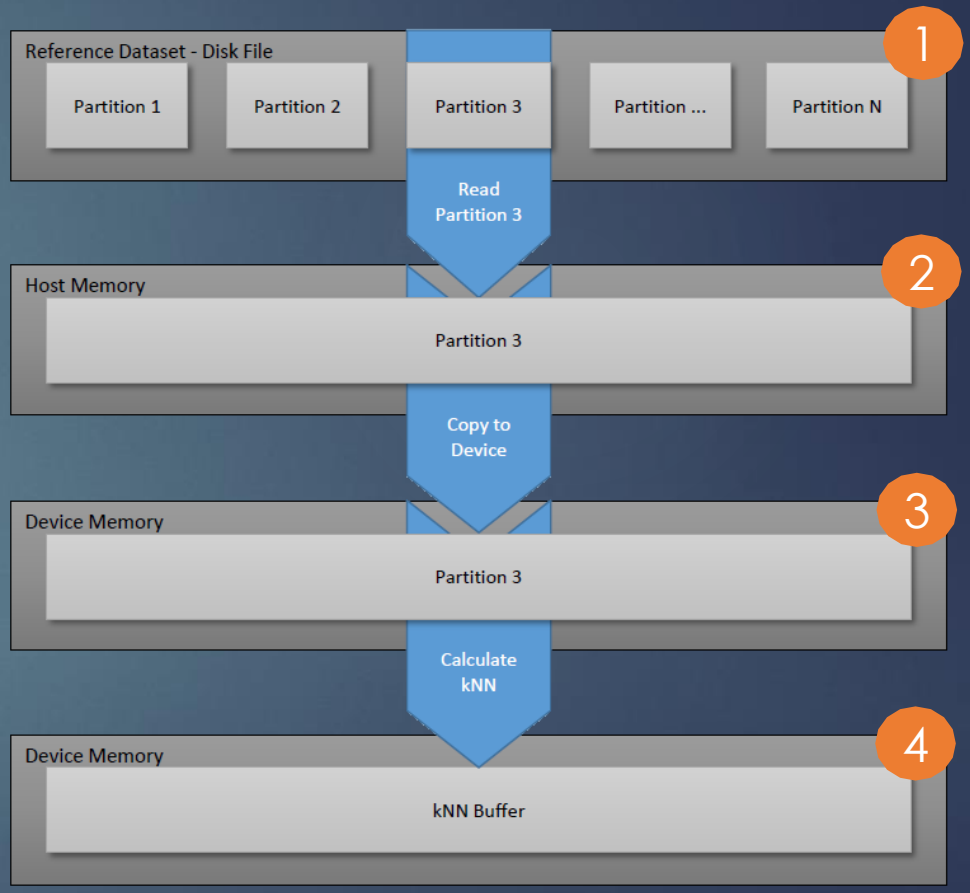
- We propose and implement four **new GPU-based algorithms for the k-NN query on Disk-resident Data**, suitable for **Big Data** processing, using the CUDA runtime API

  This algorithm

  - Loads disk resident data and efficiently computes the k-NNs of all query points, using partitioning
  - Uses a k-NN list buffer to avoid (expensive) distance sorting of big datasets and to store only k candidates for each query point (saving device memory)
- We present an extensive experimental comparison using synthetic datasets produced by the SpiderWeb generator

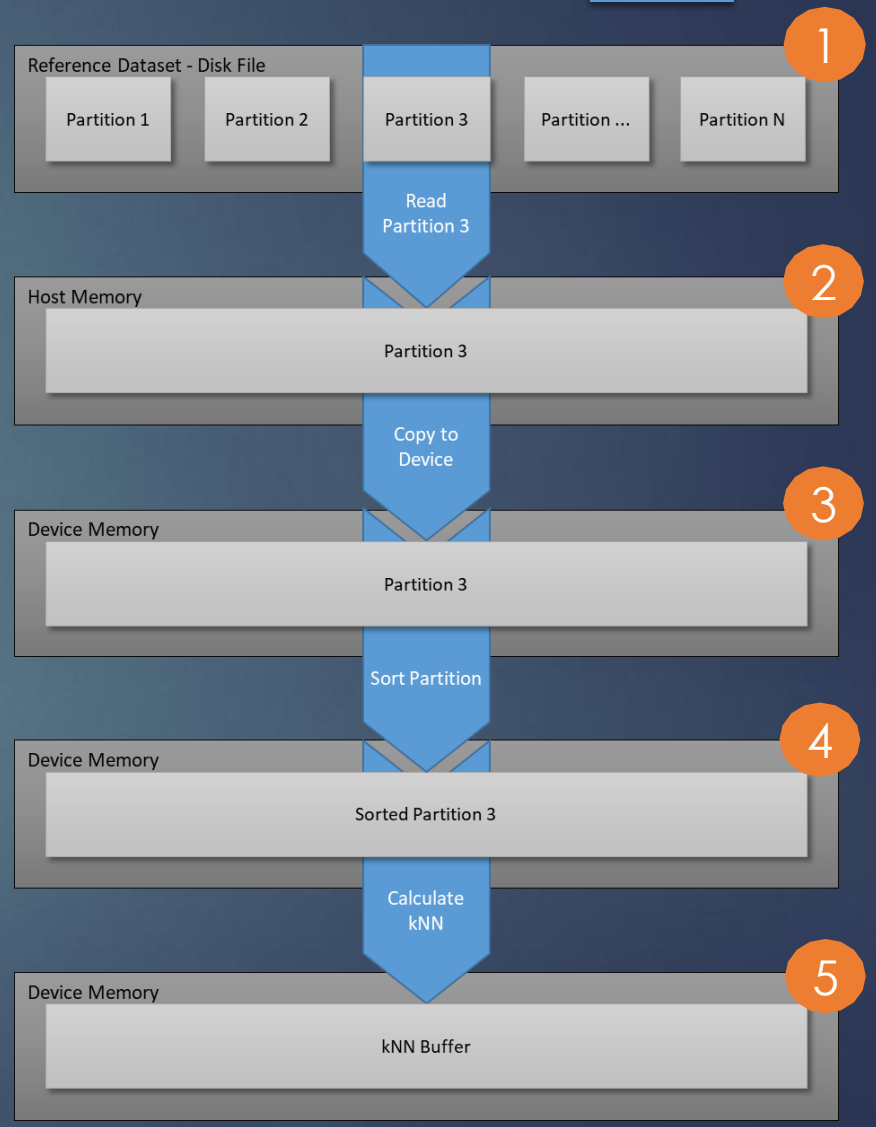
Disk Brute-force Algorithm (DBF)

- Host loads a partition from the reference datafile and transfers it to the device
- Every query point is assigned to a GPU thread
- The GPU starts the k-NN calculation simultaneously for all threads
- If the number of query points is bigger than the total available GPU threads, then the execution progresses whenever a block of threads finishes the previously assigned query points calculation
- The k-NNs are stored to the k-NN buffer
- Processing continues with the next partition



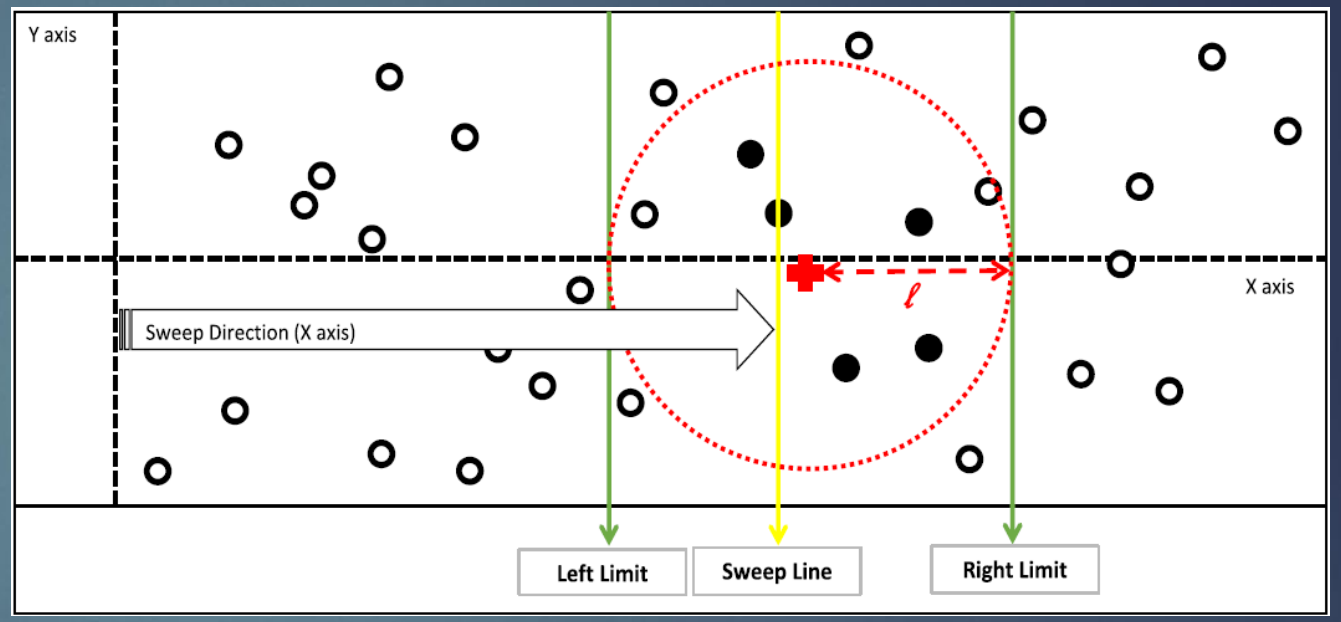
Disk Plane-sweep Algorithm (DPS) (1)

- Host loads a partition from the reference datafile and transfers it to the device
- The reference points partition is sorted on the x-axis
- Every query point is assigned to a GPU thread
- The GPU starts the k-NN plane sweep calculation simultaneously for all threads
- If the number of query points is bigger than the total available GPU threads, then the execution progresses whenever a block of threads finishes the previously assigned query points calculation
- The k-NNs are stored to the k-NN buffer
- Processing continues with the next partition



Disk Plane-sweep Algorithm (DPS) (2)

- Every thread creates a sweep line and sweeps all reference points
- The sweep-line hops every time to the next reference point until it approaches the x-value of the query point
- For every reference point within the rectangle, we calculate the Euclidean distance



Buffering methods (1)
k-NN distance list buffer (KNN-DLB)

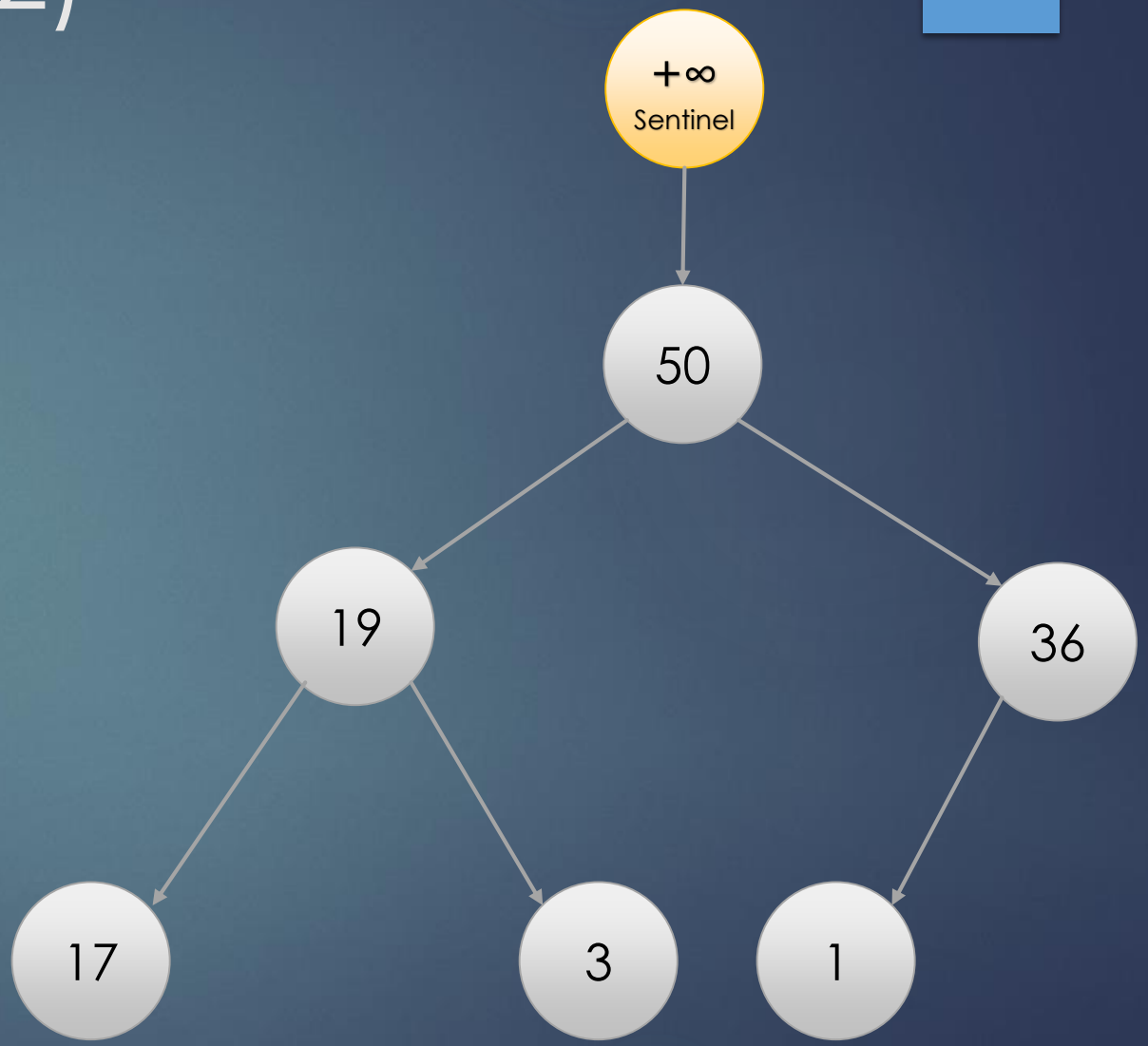
- The k-NN Distance List Buffer (KNN-DLB) is an array storing all calculated distances with size K, per thread (minimizing device memory utilization)
- When the buffer is not full, we append the calculated distances
- When the buffer is full, we check every newly calculated distance with the largest one in KNN-DLB. If it is smaller, we simply replace the largest distance with the current one (and avoid using a sorting algorithm)
- The resulting buffer contains the right k-NNs, but not in an ascending order

		KNN Distance List Buffer, k=10									
	New Distance	1	2	3	4	5	6	7	8	9	10
First 10 distances are appended to the list	5,1	5,1									
	2,7	5,1	2,7								
	4,0	5,1	2,7	4,0							
	2,8	5,1	2,7	4,0	2,8						
	11,2	5,1	2,7	4,0	2,8	11,2					
	1,7	5,1	2,7	4,0	2,8	11,2	1,7				
	3,5	5,1	2,7	4,0	2,8	11,2	1,7	3,5			
	0,6	5,1	2,7	4,0	2,8	11,2	1,7	3,5	0,6		
	0,1	5,1	2,7	4,0	2,8	11,2	1,7	3,5	0,6	0,1	
	7,1	5,1	2,7	4,0	2,8	11,2	1,7	3,5	0,6	0,1	7,1
Distances smaller that the maximum distance, replace it	8,5	5,1	2,7	4,0	2,8	8,5	1,7	3,5	0,6	0,1	7,1
	6,9	5,1	2,7	4,0	2,8	6,9	1,7	3,5	0,6	0,1	7,1
	1,6	5,1	2,7	4,0	2,8	6,9	1,7	3,5	0,6	0,1	1,6
	5,8	5,1	2,7	4,0	2,8	5,8	1,7	3,5	0,6	0,1	1,6

Buffering methods (2)

max Heap

- We are using max-Heal with sentinel
- Max Heap is a priority queue represented by a complete binary tree which is implemented using an array
- The first element (after the sentinel) stores the maximum value of the Heap



Experimental Evaluation – Algorithms

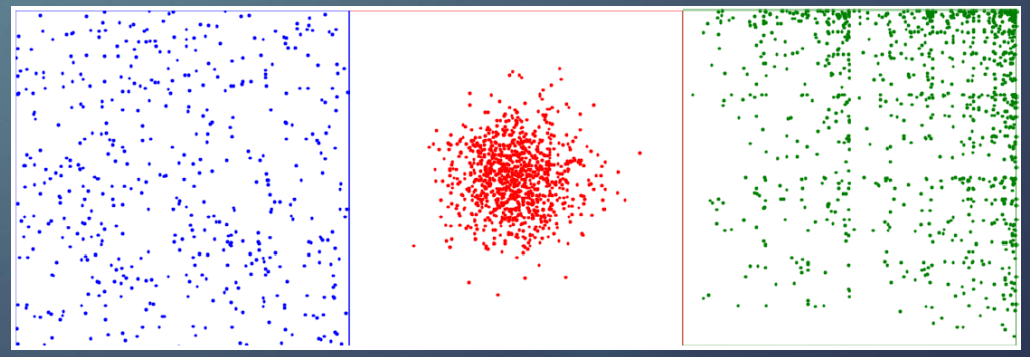
Experimental evaluation of 4 kNN algorithms:

1. DBF, Disk Brute-force using KNN-DLB buffer
2. DBF Heap, Disk Brute-force using max Heap buffer
3. DPS, Plane-sweep using KNN-DLB buffer
4. DPS Heap, Plane-sweep using max Heap buffer

Experimental Evaluation Data

- SpiderWeb Dataset generator data
- Experiment distributions, Left=Uniform, Middle=Gaussian, Right=Bit

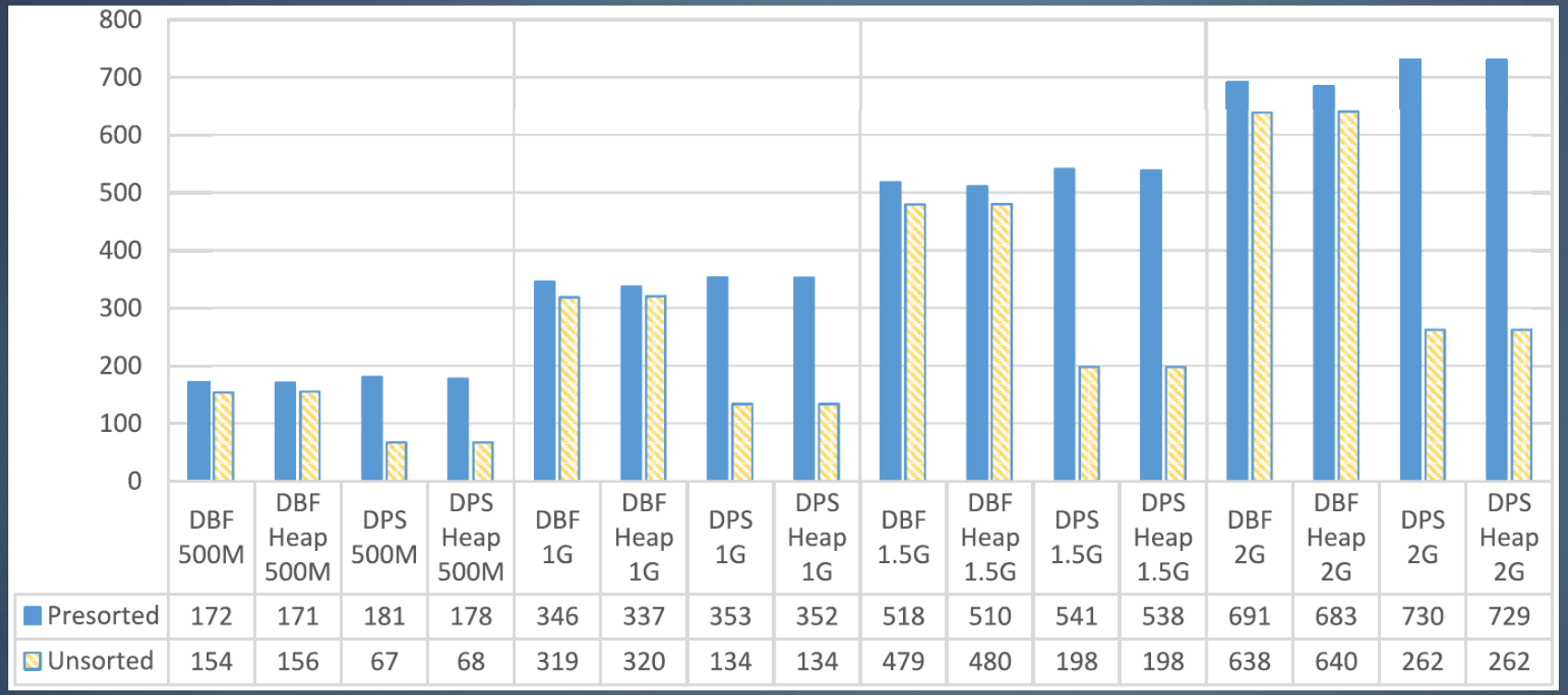
Distribution	Size	Seed	File Size	Dataset usage
Bit	500M	1	16GB	Reference
Bit	1G	2	32GB	Reference
Bit	1.5G	3	48GB	Reference
Bit	2G	4	64GB	Reference
Uniform	10	5	32B	Query
Gaussian	10K	6	320KB	Query
Gaussian	20K	7	640KB	Query
Gaussian	30K	8	960KB	Query
Gaussian	40K	9	1,3MB	Query
Gaussian	50K	10	1,6MB	Query



Experimental Evaluation: setup

- Dell G5 15 laptop
- Ubuntu 20.04
- Six-core (12-thread) Intel I7 CPU
- 16GB of main memory
- 1TB SSD disk
- NVIDIA Geforce 2070 (Mobile Max-Q) GPU with 8GB of memory

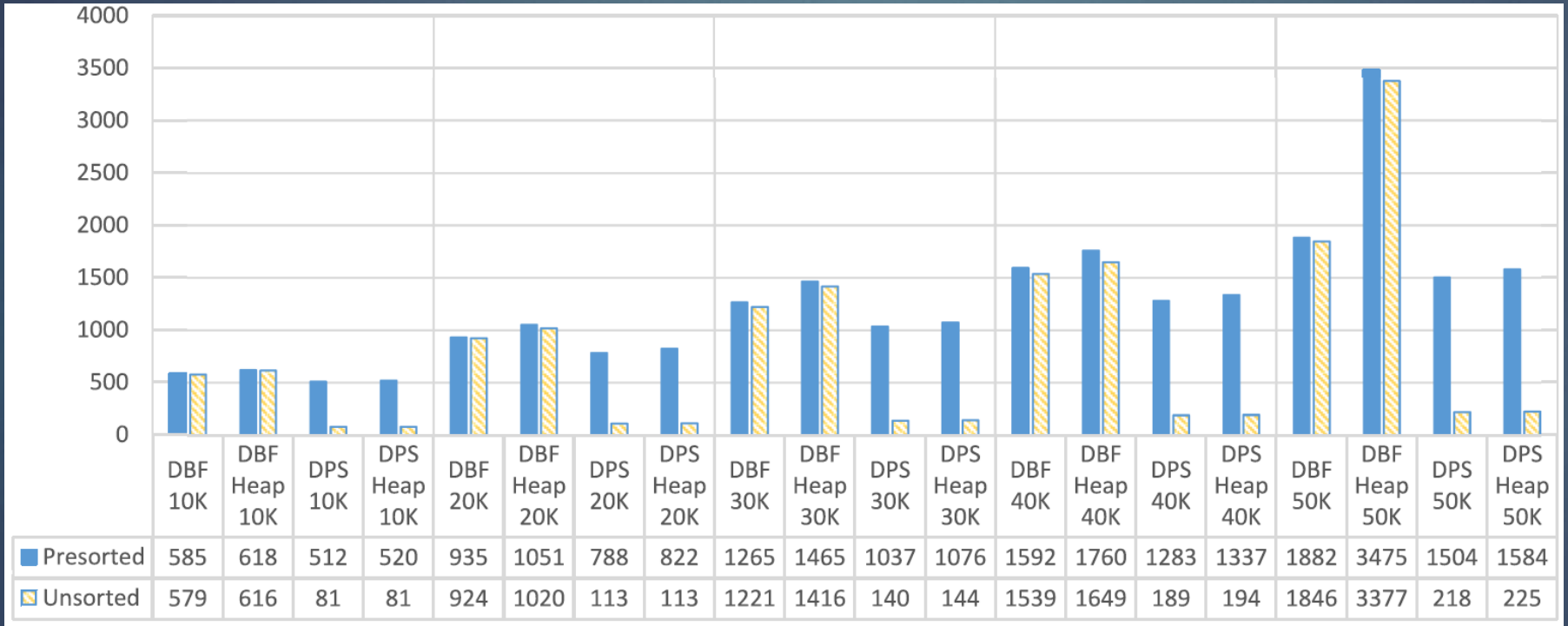
Experiments:
Reference dataset scaling



	DBF 500M	DBF Heap 500M	DPS 500M	DPS Heap 500M	DBF 1G	DBF Heap 1G	DPS 1G	DPS Heap 1G	DBF 1.5G	DBF Heap 1.5G	DPS 1.5G	DPS Heap 1.5G	DBF 2G	DBF Heap 2G	DPS 2G	DPS Heap 2G
Presorted	172	171	181	178	346	337	353	352	518	510	541	538	691	683	730	729
Unsorted	154	156	67	68	319	320	134	134	479	480	198	198	638	640	262	262

- All methods performed better for the unsorted dataset
- Plane-sweep methods performed exceptionally better than for the presorted dataset
- Plane-sweep methods were more than 1.7 times faster than Brute-force ones, in the unsorted dataset experiments

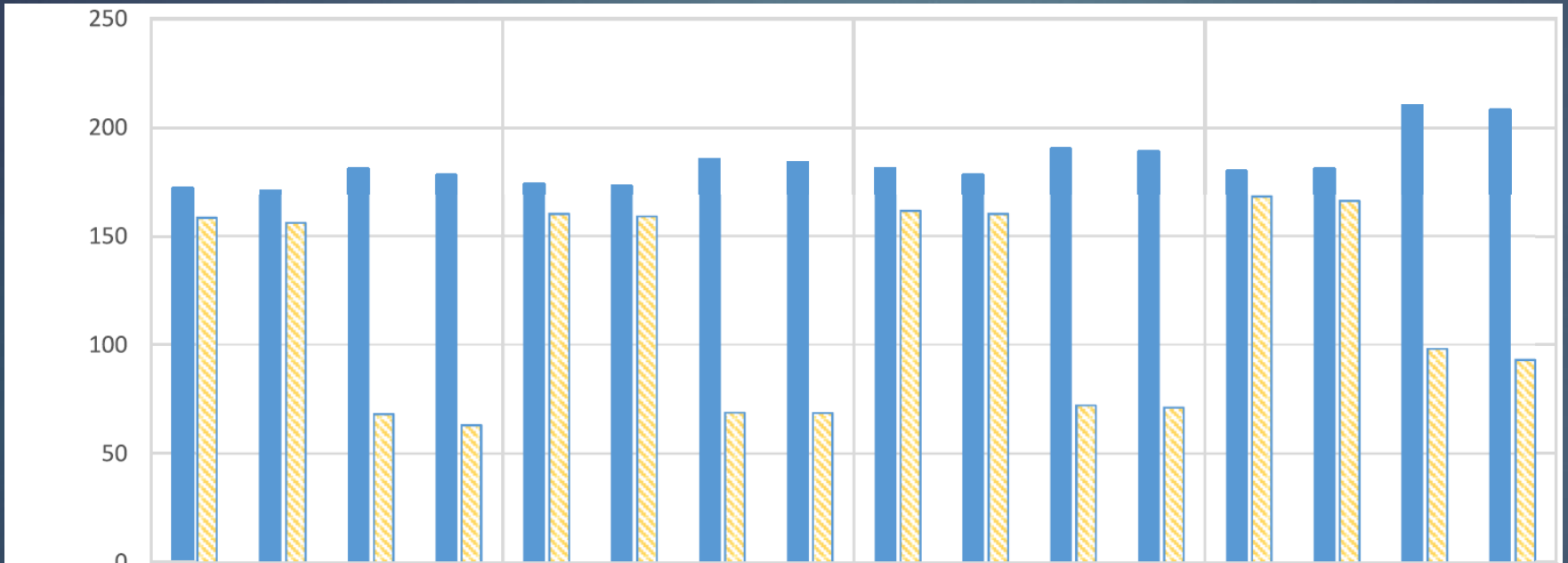
Experiments: Query dataset scaling



	DBF 10K	DBF Heap 10K	DPS 10K	DPS Heap 10K	DBF 20K	DBF Heap 20K	DPS 20K	DPS Heap 20K	DBF 30K	DBF Heap 30K	DPS 30K	DPS Heap 30K	DBF 40K	DBF Heap 40K	DPS 40K	DPS Heap 40K	DBF 50K	DBF Heap 50K	DPS 50K	DPS Heap 50K
Presorted	585	618	512	520	935	1051	788	822	1265	1465	1037	1076	1592	1760	1283	1337	1882	3475	1504	1584
Unsorted	579	616	81	81	924	1020	113	113	1221	1416	140	144	1539	1649	189	194	1846	3377	218	225

- All the methods performed better with the unsorted dataset
- Plane-sweep methods performed once again exceptionally better when using the presorted dataset
- Plane-sweep methods were more than 7 to 15 times faster than Brute-force ones, in the unsorted dataset experiments

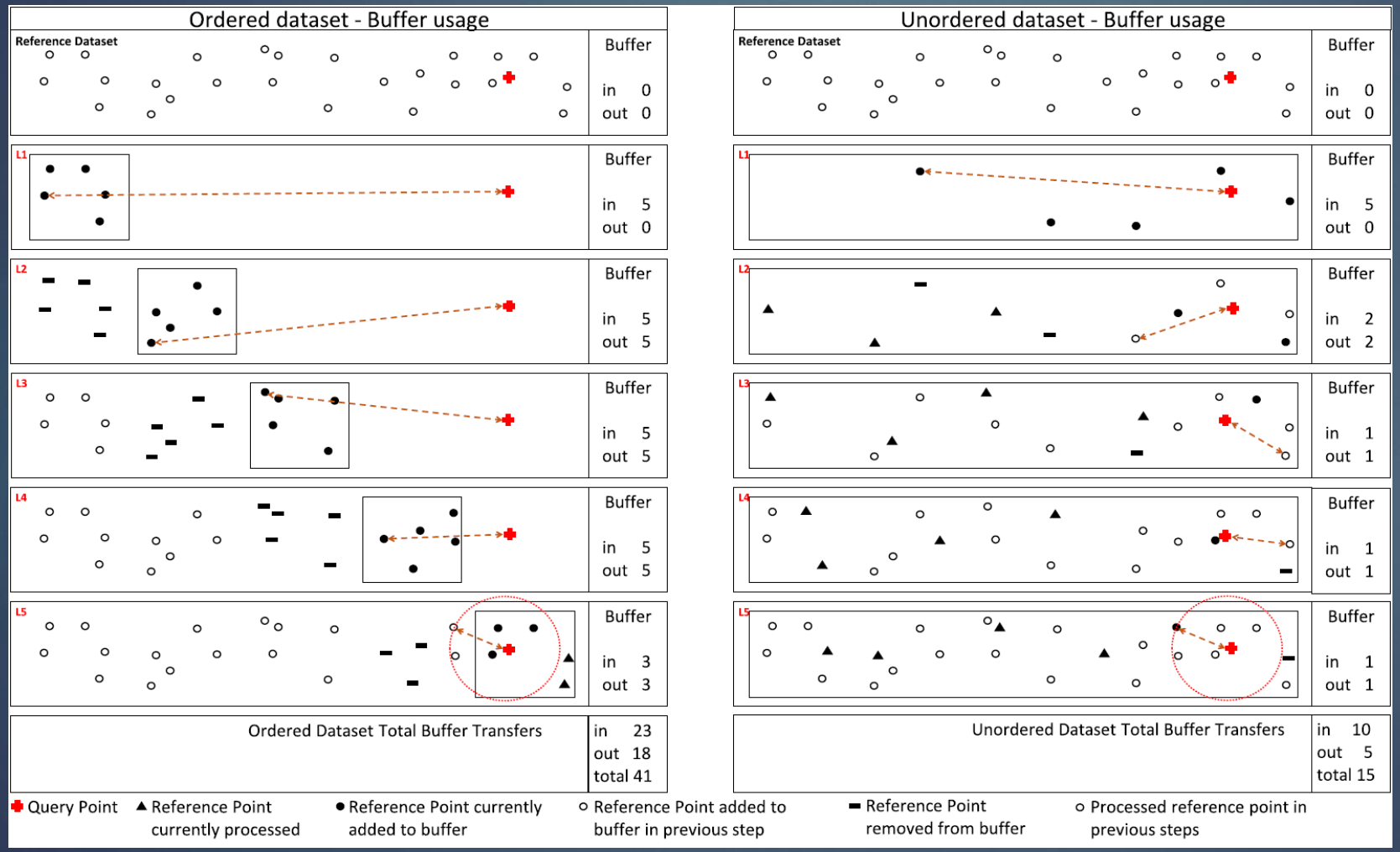
Experiments: k scaling



	DBF k=10	DBF Heap k=10	DPS k=10	DPS Heap k=10	DBF k=20	DBF Heap k=20	DPS k=20	DPS Heap k=20	DBF k=50	DBF Heap k=50	DPS k=50	DPS Heap k=50	DBF k=100	DBF Heap k=100	DPS k=100	DPS Heap k=100
Presorted	172	171	181	178	174	173	185	184	181	178	190	189	180	181	210	208
Unsorted	158	156	68	63	160	159	69	69	161	160	72	71	168	166	98	93

- All the methods performed better with the unsorted dataset
- Plane-sweep methods performed once again exceptionally better than the others for the presorted dataset
- Plane-sweep methods were about 2 times faster than Brute-force ones, in the unsorted dataset experiments

Results Interpretation:
Presorted vs Unsorted



- When the reference dataset is presorted each partition contains points that fall within a limited x-range
- When the reference dataset is unsorted, each partition contains points that cover a wide x-range
- The maximum k-NN distance (red dotted line) is decreasing faster when using unordered datasets
- This results to less total buffer transfers (15 unordered vs 41 presorted)

Conclusions

- Our algorithms exploit the numerous GPU cores, utilize the device memory as much as possible and take advantage of the speed and storage capacity of SSDs, thus process efficiently big reference datasets
- Plane-sweep on unsorted reference data (with either an array or a max-Heap buffer for organizing the current k-NNs) is a clear performance winner

Future work

- Development of k-NN GPU-based algorithms for big SSD resident data which exploit the use of indexes to further speed-up processing
- Implementation of other queries (like k-closest pairs), based on techniques utilized in this paper

Thank you for your attention