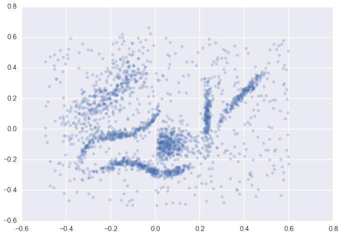
Theoretically Efficient Parallel Density-Peaks Clustering

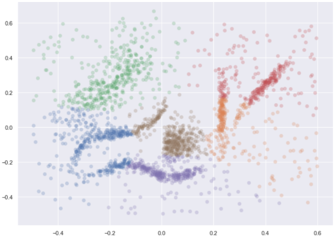
Michael Huang
Under the direction of
Shangdi Yu and Prof. Julian Shun

October 16, 2022

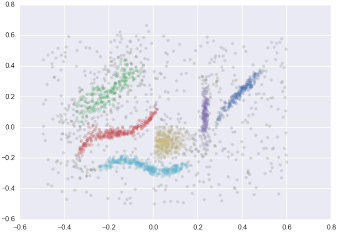
Density-based Clustering



Unclustered data



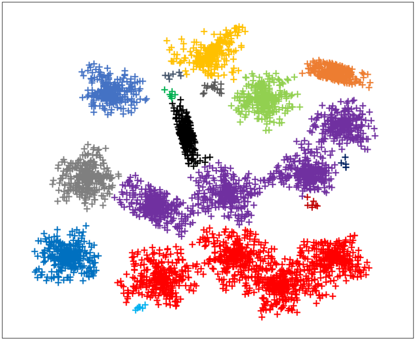
k-means clustering result[1]



DBSCAN clustering result[2]

[1] Everitt, Landau, and Leese 2009.
[2] Ester et al. 1996.

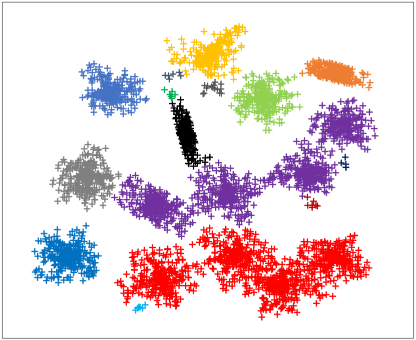
Why Density-Peaks Clustering (DPC) Algorithm



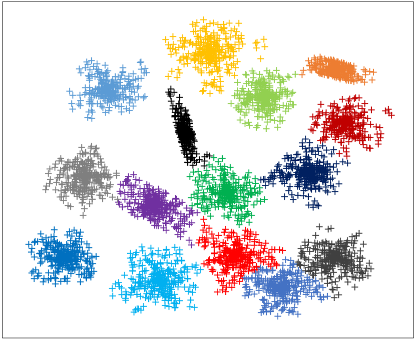
DBSCAN fails on datasets where clusters are close together[3]

[3] Amagata and Hara 2021.

Why Density-Peaks Clustering (DPC) Algorithm



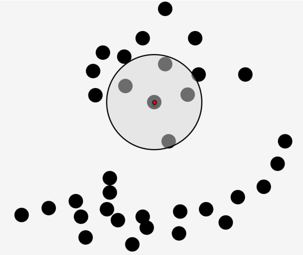
DBSCAN fails on datasets where clusters are close together[3]



DPC is able to separate close clusters[4]

[3] Amagata and Hara 2021.
[4] Amagata and Hara 2021.

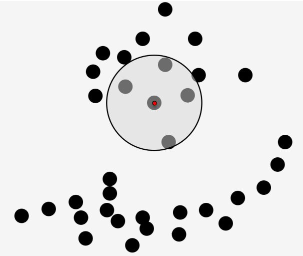
DPC Algorithm Procedure Description



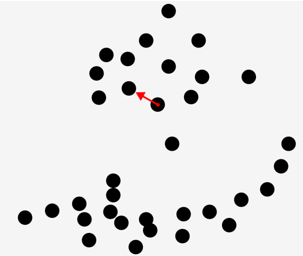
Compute density[5]

[5] Rodriguez and Laio 2014.

DPC Algorithm Procedure Description



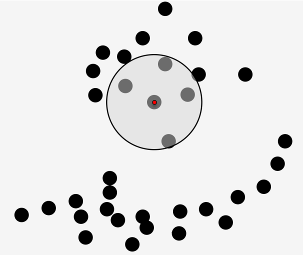
Compute density[5]



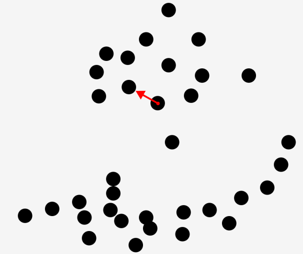
Find dependent point (the nearest neighbor with higher density)[6]

[5] Rodriguez and Laio 2014.
[6] Rodriguez and Laio 2014.

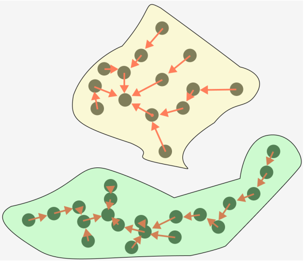
DPC Algorithm Procedure Description



Compute density[5]



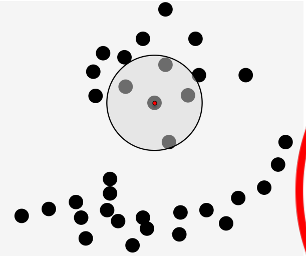
Find dependent point (the nearest neighbor with higher density)[6]



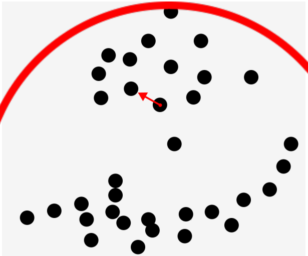
Separate into clusters[7]

[5] Rodriguez and Laio 2014.
[6] Rodriguez and Laio 2014.
[7] Rodriguez and Laio 2014.

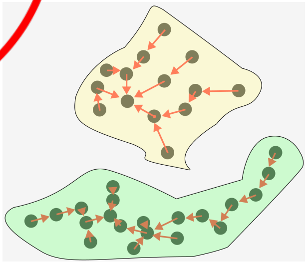
DPC Algorithm Procedure Description



Compute density[5]



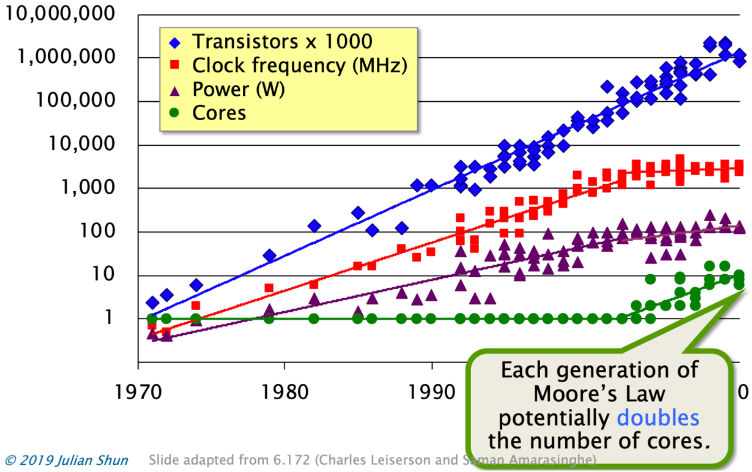
Find dependent point (the nearest neighbor with higher density)[6]



Separate into clusters[7]

[5] Rodriguez and Laio 2014.
[6] Rodriguez and Laio 2014.
[7] Rodriguez and Laio 2014.

Why Focus on Parallelism



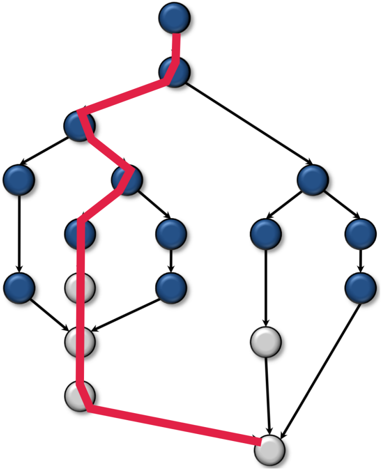
CPU clock-speed hits ceiling; #cores increases exponentially[8]

[8] Shun 2021.

T_p = runtime with p processors
T_1 = work
T_∞ = span

Brent's Law:

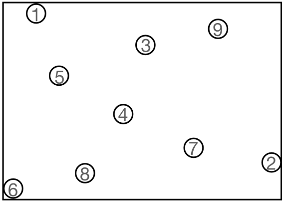
$$T_p \leq T_\infty + \frac{T_1 - T_\infty}{p}$$



Computational graph of a parallel algorithm[9]

[9] Shun 2021.

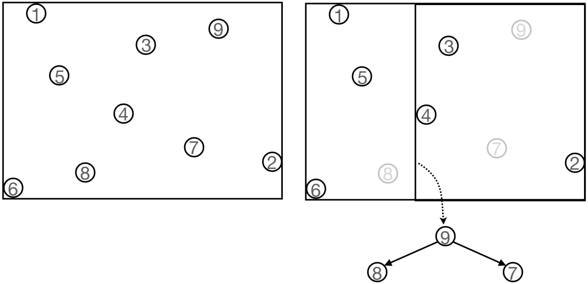
Parallel Dependent Point Finding with Priority Search *k*d-tree



Binary Space Partitioning Tree:

1. Divide points up equally
2. Satisfy heap property (higher in the tree $\implies$ higher density)

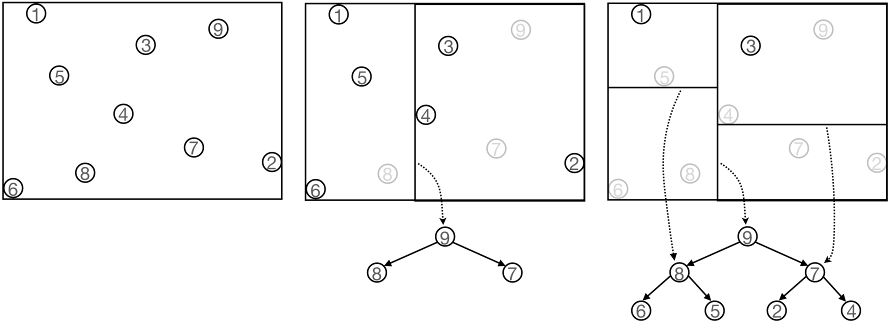
Parallel Dependent Point Finding with Priority Search *k*d-tree



Binary Space Partitioning Tree:

1. Divide points up equally
2. Satisfy heap property (higher in the tree $\implies$ higher density)

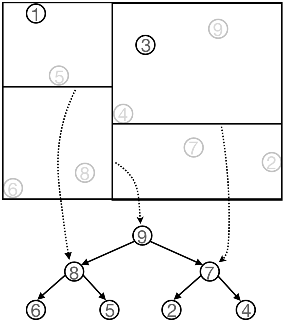
Parallel Dependent Point Finding with Priority Search *k*d-tree



Binary Space Partitioning Tree:

1. Divide points up equally
2. Satisfy heap property (higher in the tree $\implies$ higher density)

Parallel Dependent Point Finding with Priority Search *k*d-tree

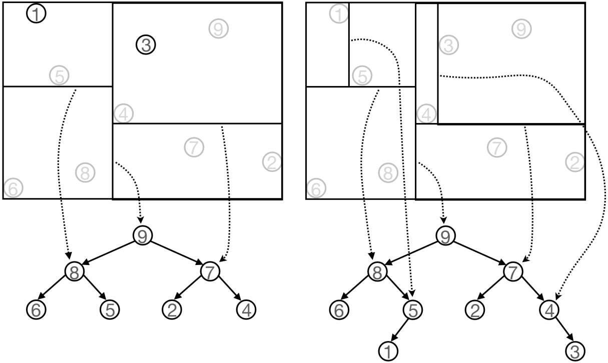


Binary Space Partitioning Tree:

1. Divide points up equally
2. Satisfy heap property (higher in the tree $\Longrightarrow$ higher density)

Reduce dependent point finding from $O(n)$ to Avg. $O(\log(n))$

Parallel Dependent Point Finding with Priority Search *k*d-tree

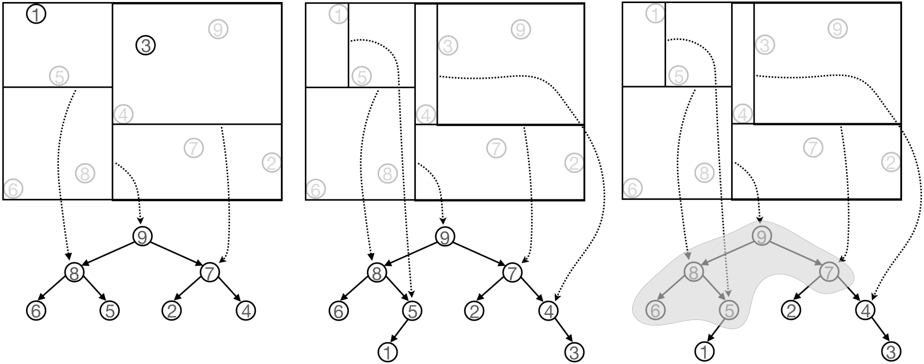


Binary Space Partitioning Tree:

1. Divide points up equally
2. Satisfy heap property (higher in the tree $\implies$ higher density)

Reduce dependent point finding from $O(n)$ to Avg. $O(\log(n))$

Parallel Dependent Point Finding with Priority Search *k*d-tree



Binary Space Partitioning Tree:

1. Divide points up equally
2. Satisfy heap property (higher in the tree $\implies$ higher density)

Reduce dependent point finding from $O(n)$ to Avg. $O(\log(n))$

Algorithmic Complexity

	Compute density		Find dependent point	
Algorithms	Avg. Work	Avg. Span	Avg. Work	Avg. Span
Previous SOTA[10]	$O(n^{2-\frac{1}{d}} + n\rho)$	$O(n^{1-\frac{1}{d}} + \rho)$	$O(n^2)$	$O(n)$
Our algorithm	$O(n^{2-\frac{1}{d}})$	$O(n^{1-\frac{1}{d}})$	$O(n \log(n))$	$O(\log(n))$

Complexity comparison

1. n: the number of points to be clustered
2. ρ: average density of points
3. d: the number of dimensions each point has

[10] Rodriguez and Laio 2014; Amagata and Hara 2021.

Experiment Setup

1. 30-core, 2-way hyperthreading, CPU @3.1 GHz
2. Implemented with ParlayLib[11] and ParGeo[12]

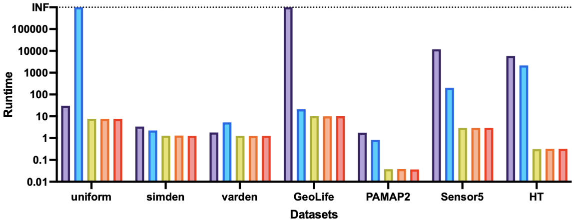
Google Cloud Platform

Dataset	n	d	synthetic
uniform	10M	2	yes
simden	10M	2	yes
varden	10M	2	yes
GeoLife	24.88M	3	no
PAMAP2	0.26M	4	no
Sensor	3.84M	5	no
HT	0.93M	8	no

Datasets

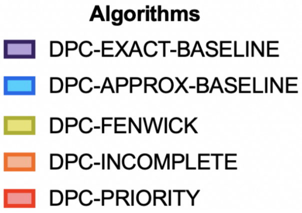
[11] Blelloch, Anderson, and Dhulipala 2020.
[12] Wang et al. 2022.

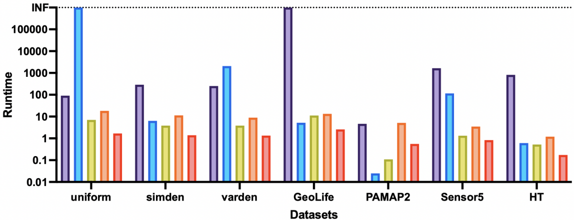
Runtime Comparison



Density computation

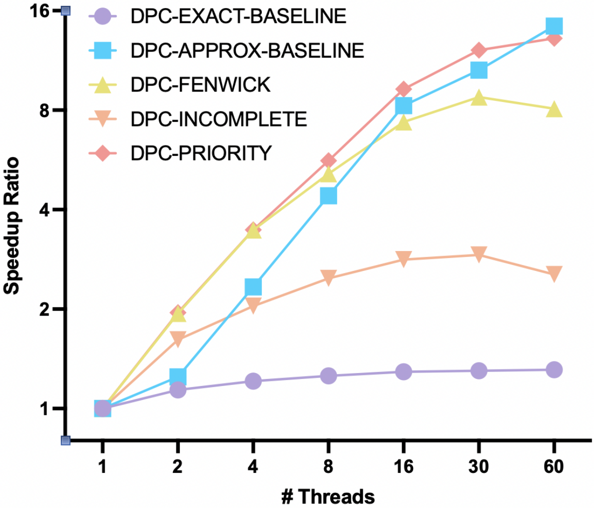
Overall speedups: 8.3–4666.3x





Dependent point finding

Parallel Scalability



13.2x self-relative speedup

Conclusion

1. Proposed the Priority Search *k*d-tree data structure and proved its avg. query complexity
2. Developed a theoretically efficient and practically fast DPC algorithm, with up to 4666x speedup compared to SOTA

Acknowledgements

- Shangdi Yu
- Prof. Julian Shun
- MIT PRIMES Program