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

\[ \begin{align*}
\text{k-means clustering result}^1 & \quad \text{DBSCAN clustering result}^2
\end{align*} \]

---

1 Everitt, Landau, and Leese 2009.
2 Ester et al. 1996.
DBSCAN fails on datasets where clusters are close together\(^3\)

\(^3\)Amagata and Hara 2021.
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.
Compute density\textsuperscript{5}

\textsuperscript{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

1. Compute density
2. Find dependent point (the nearest neighbor with higher density)
3. Separate into clusters

---

DPC Algorithm Procedure Description

1. Compute density\(^5\)
2. Find dependent point (the nearest neighbor with higher density)\(^6\)
3. Separate into clusters\(^7\)

\(^5\) Rodríguez and Laio 2014.
\(^6\) Rodríguez and Laio 2014.
\(^7\) Rodríguez and Laio 2014.
Why Focus on Parallelism

CPU clock-speed hits ceiling; #cores increases exponentially\(^8\)

Each generation of Moore’s Law potentially doubles the number of cores.

\(^8\)Shun 2021.
Parallel Algorithm Background

\[ T_p = \text{runtime with } p \text{ processors} \]
\[ T_1 = \text{work} \]
\[ T_\infty = \text{span} \]

Brent’s Law:

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

Computational graph of a parallel algorithm\(^9\)

\(^9\)Shun 2021.
Binary Space Partitioning Tree:

1. Divide points up equally
2. Satisfy heap property (higher in the tree $\Rightarrow$ higher density)
Parallel Dependent Point Finding with Priority Search

kd-tree

Binary Space Partitioning Tree:

1. Divide points up equally
2. Satisfy heap property (higher in the tree $\Rightarrow$ higher density)
Binary Space Partitioning Tree:

1. Divide points up equally
2. Satisfy heap property (higher in the tree $\Rightarrow$ higher density)
Parallel Dependent Point Finding with Priority Search

kd-tree

Binary Space Partitioning Tree:
1. Divide points up equally
2. Satisfy heap property (higher in the tree $\Rightarrow$ higher density)

Reduce dependent point finding from $O(n)$ to Avg. $O(\log(n))$
Parallel Dependent Point Finding with Priority Search

kd-tree

Binary Space Partitioning Tree:
1. Divide points up equally
2. Satisfy heap property (higher in the tree $\Rightarrow$ higher density)

Reduce dependent point finding from $O(n)$ to Avg. $O(\log(n))$
Parallel Dependent Point Finding with Priority Search

kd-tree

Binary Space Partitioning Tree:

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

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

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Compute density</th>
<th>Find dependent point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous SOTA(^\text{10})</td>
<td>$O(n^{2-\frac{1}{d}} + n\rho)$</td>
<td>$O(n^{1-\frac{1}{d}} + \rho)$</td>
</tr>
<tr>
<td>Our algorithm</td>
<td>$O(n^{2-\frac{1}{d}})$</td>
<td>$O(n^{1-\frac{1}{d}})$</td>
</tr>
</tbody>
</table>

**Complexity comparison**

1. **n**: the number of points to be clustered
2. **\(\rho\)**: average density of points
3. **d**: the number of dimensions each point has

---

\(^{10}\text{Rodriguez and Laio 2014; Amagata and Hara 2021.}\)
Experiment Setup

1. 30-core, 2-way hyperthreading, CPU @3.1 GHz
2. Implemented with ParlayLib\textsuperscript{11} and ParGeo\textsuperscript{12}

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$n$</th>
<th>$d$</th>
<th>synthetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniform</td>
<td>10M</td>
<td>2</td>
<td>yes</td>
</tr>
<tr>
<td>simden</td>
<td>10M</td>
<td>2</td>
<td>yes</td>
</tr>
<tr>
<td>varden</td>
<td>10M</td>
<td>2</td>
<td>yes</td>
</tr>
<tr>
<td>GeoLife</td>
<td>24.88M</td>
<td>3</td>
<td>no</td>
</tr>
<tr>
<td>PAMAP2</td>
<td>0.26M</td>
<td>4</td>
<td>no</td>
</tr>
<tr>
<td>Sensor</td>
<td>3.84M</td>
<td>5</td>
<td>no</td>
</tr>
<tr>
<td>HT</td>
<td>0.93M</td>
<td>8</td>
<td>no</td>
</tr>
</tbody>
</table>

\textsuperscript{11}Bleloch, Anderson, and Dhulipala 2020.
\textsuperscript{12}Wang et al. 2022.
Runtime Comparison

Density computation

Overall speedups: 8.3–4666.3x

Algorithms
- DPC-EXACT-BASELINE
- DPC-APPROX-BASELINE
- DPC-FENWICK
- DPC-INCOMPLETE
- DPC-PRIORITY

Dependent point finding
13.2x self-relative speedup
Conclusion

1. Proposed the Priority Search $kd$-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