# Hierarchical Density-Based Clustering based on GPU Accelerated Data Indexing Strategy

Danilo Melo[1] Sávyo Toledo[1] **Guilherme Andrade**[2]
Fernando Mourão[1] Aniket Chakrabarti Renato Ferreira[2]
Srinivasan Parthasarathy[3] Leonardo Rocha[1]

1. Federal University of Minas Gerais, Brazil
2. Federal University of São João del Rei, Brazil
3. Dept. of Computer Science and Engineering, The Ohio-State University

June - 2016

# Summary

# Summary

# Motivation

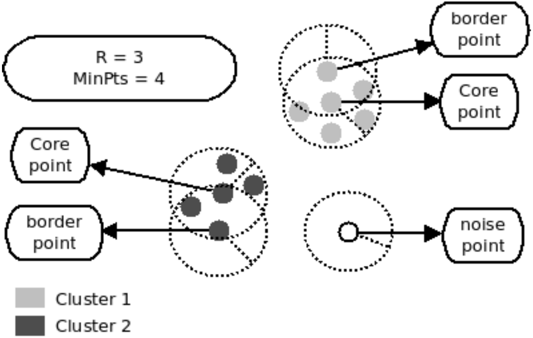
## Motivation Scenario

- The large volume of data generated has emerged in recent years a challenging scenario for several applications.
- New proposals for models and algorithms that are able to handle this data efficiently and effectively are emerging every moment.
    - Data Mining area!
    - Clustering algorithms.
- Social networks, recommendation systems, bioinformatics..

# Motivation

## Density-based Clustering

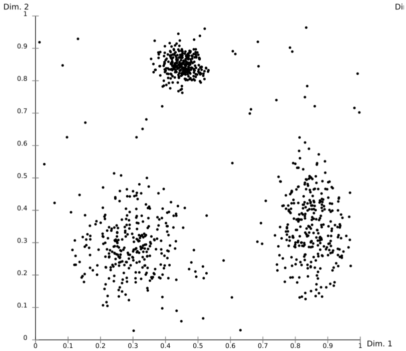
- clusters are areas with **high density** separated by areas with **low density**
- Dense area: if there are more them *MinPts* with a distance between them lower than $\epsilon - distance$.
- Density-based spatial clustering of applications with noise (DBSCAN)



# Motivation

## OPTICS Algorithm

- Same idea of DBSCAN
- Addresses one of DBSCAN's major weaknesses
- **The problem of detecting meaningful clusters in data of varying density**



# Motivation

## OPTICS Problems

- When comes to big volume of data, this strategies is time-demanding.
- Strategies has been proposed to make these applications feasible.
    - Data Indexation
    - Parallel Computing
- GPU has been given considerable importance, since these are able of providing a higher level of parallelism than multicore CPU's, associated with a lower energy consumption.

# Contribution

## Contribution

**In this work we present a new approach to make OPTICS feasible based on data indexing strategy parallelized using GPU.**

# Contribution

## Contribution

**In this work we present a new approach to make OPTICS feasible based on data indexing strategy parallelized using GPU.**

## Contribution Characteristics

- Representation as a graph $G(V, E)$.
- Compact Adjacent List
- Graph Construction completely parallel
- OPTICS algorithm becomes very fast

# Summary

# Optics Overview: Main Idea

## Main Idea

- Points of the database are (linearly) ordered
- Points which are spatially closest become neighbors in the ordering.
- Special distance is stored for each point that represents the density that needs to be accepted for a cluster in order to have both points belong to the same cluster.

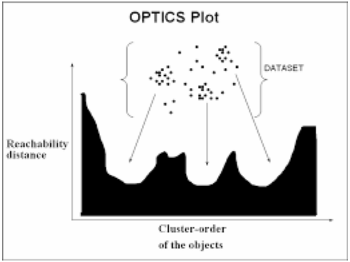



Figure: Image from: OPTICS on Text Data: Experiments and Test

# Optics Overview: Main Concepts

### $\epsilon$-*neighborhood*

- The neighborhood of an object $p$ is the set of objects $s$ so that $distance(p, s) \leq \epsilon$.

# Optics Overview: Main Concepts

### *$\epsilon$-neighborhood*

- The neighborhood of an object $p$ is the set of objects $s$ so that $distance(p, s) \leq \epsilon$.

### *core-distance of p*

- Smallest distance that makes $p$ a core point. The distance from $p$ to the $minPts - th$ neighbor.

# Optics Overview: Main Concepts

### *$\epsilon$-neighborhood*

- The neighborhood of an object $p$ is the set of objects $s$ so that $distance(p, s) \leq \epsilon$.

### *core-distance of p*

- Smallest distance that makes $p$ a core point. The distance from $p$ to the $minPts - th$ neighbor.

### *reachability-distance (p,o)*

- Smallest distance from $p$ to $o$ if $o$ is a core object.

# Optics Overview: Algorithm Steps

## Algorithm Steps

1. OPTICS maintains a priority queue

# Optics Overview: Algorithm Steps

## Algorithm Steps

1. OPTICS maintains a priority queue
2. Take object $p$ and determines its core distance.

# Optics Overview: Algorithm Steps

## Algorithm Steps

1. OPTICS maintains a priority queue
2. Take object $p$ and determines its core distance.
3. for all its neighbours $q \in N_\epsilon(p)$ are calculated a new reachability distance

# Optics Overview: Algorithm Steps

## Algorithm Steps

1. OPTICS maintains a priority queue
2. Take object $p$ and determines its core distance.
3. for all its neighbours $q \in N_\epsilon(p)$ are calculated a new reachability distance
4. Insert or update $q$ in priority queue

# Optics Overview: Algorithm Steps

## Algorithm Steps

1. OPTICS maintains a priority queue
2. Take object $p$ and determines its core distance.
3. for all its neighbours $q \in N_\epsilon(p)$ are calculated a new reachability distance
4. Insert or update $q$ in priority queue
5. Repeat until Seeds is empty and there is no unprocessed object

# Optics Overview: Algorithm Steps

## Algorithm Steps

1. OPTICS maintains a priority queue
2. Take object $p$ and determines its core distance.
3. for all its neighbours $q \in N_\epsilon(p)$ are calculated a new reachability distance
4. Insert or update $q$ in priority queue
5. Repeat until Seeds is empty and there is no unprocessed object
6. Based on the output of OPTICS algorithm we can extract any density-based clustering.

# Optics Overview: Algorithm Analysis

## Algorithm Analysis

- OPTICS is heavily dominated by the runtime to get or consult $\epsilon$-*neighborhood* for *core-distance and reachability-distance operations. For each object, OPTICS could be* $O(n^2)$
- **Data indexing techniques can be used***s*

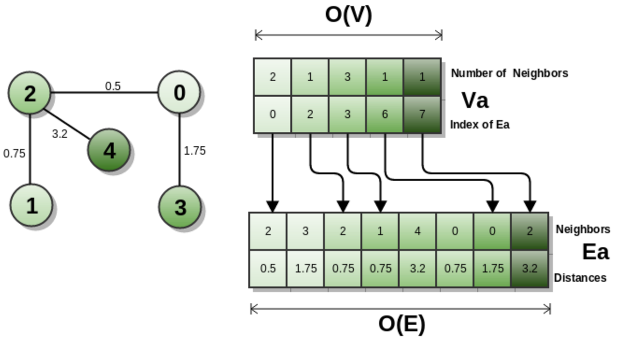
# Summary

# G-OPTICS: Data Representation

## Data Representation

- Represent the data as a graph $G(V, E)$
  - $V$ represents the objects to be clustered
  - $E$ the edges connecting the objects that are within the minimum distance radius of each other (smaller than $\epsilon$)
  - Edges weighted: distance between two objects
- This distance can be calculated by metrics of similarity
- Compact adjacency list.

# Data Representation




## Considerations

- $\epsilon$ and *MinPts* as Parameters
- calculate the distance to other objects
- Insert edge if distance is lower than $\epsilon$
- Sort adjacent lists (QuickSort algorithm)

# Important Consideration

## Optics Complexity

- $\epsilon$-*neighborhood* of a object is a $O(1)$ operation.
- The **sort step** is very important as it makes the complexity to search the $MinPts^{th}$ neighbour $O(1)$, which corresponds to the process to find the *core distance*
- heap structure to represent the priority queue *Seeds*
- Total Optics Complexity $O(E * logV)$

# Important Consideration

## Optics Complexity

- $\epsilon$-*neighborhood* of a object is a $O(1)$ operation.
- The **sort step** is very important as it makes the complexity to search the $MinPts^{th}$ neighbour $O(1)$, which corresponds to the process to find the *core distance*
- heap structure to represent the priority queue *Seeds*
- Total Optics Complexity $O(E * logV)$

## Graph Construction Complexity

- $O(V^2)$, since, in the worst case, it will require a comparison between each pair of objects
- **Parallelization on GPU!**

# Parallel implementation

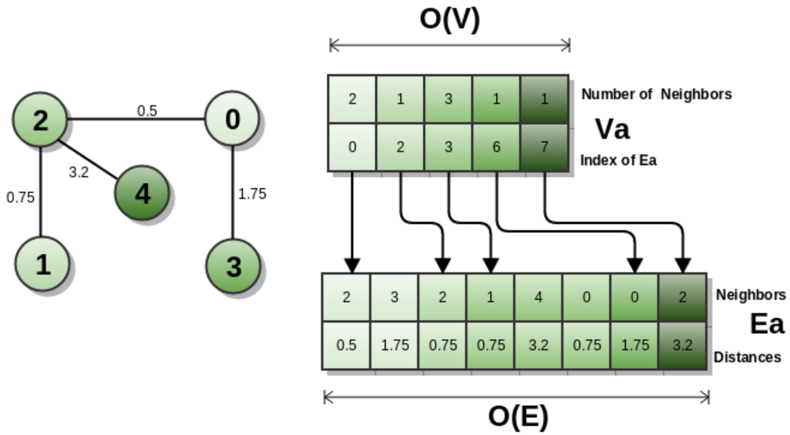
## Parallelization only Graph Construction

- Higher complexity than the OPTICS process
- Optics algorithm is based on dependent iterations which limiting parallel opportunities.

## Parallelization Steps

- Vertice degree calculation (first Va)
- Adjacency index calculation (second Va)
- Adjacency lists assembly (Ea)
- Adjacency lists sorting

# Parallelization Steps



# Vertice degree calculation:

## Vertice degree calculation:

- Multiple cores of the GPU to process multiple vertices in parallel
- Thread to each vertex
- Each GPU thread will count how many adjacent vertices has under its responsibility, filling the first value on the vector $Va$ ($Va_1$).
- **No dependencies!**

# Adjacency index calculation:

## Adjacency index calculation:

- The second value in *Va* is related to the start index in *Ea* of the adjacency list of a particular vertex
- $Va_2[i] = Va_2[i-1] + Va_1[i-1]$
- efficiently done in parallel using an **exclusive_scan** operation filling the first value on the vector *Va*
- For this operation, we used the *thrust* library

# Adjacency lists assembly

## Adjacency lists assembly

- For each vertex, we know its degree and the start index of its adjacency list, calculated in the two previous steps
- Simply mount the compact adjacency list ($Ea$)
- $Ea_1$ vertice id and $Ea_2$ vertice distance
- We assign a GPU thread to each vertex
- Each of these threads will fill the adjacency list of its associated vertex with all vertices adjacent to it.

# Adjacency lists sorting

## Adjacency lists sorting

- Having the vector $Ea_1$ and $Ea_2$ been completely filled, we can now simply sort each adjacent list.
- Following the logic of the Third step, we assign a GPU thread to each vertex
- Each of these threads will sort the adjacency list of its associated vertex

# Adjacency lists sorting

## Adjacency lists sorting

- Having the vector $Ea_1$ and $Ea_2$ been completely filled, we can now simply sort each adjacent list.
- Following the logic of the Third step, we assign a GPU thread to each vertex
- Each of these threads will sort the adjacency list of its associated vertex

## Consideration

We adopt the **Selection Sort**, since the complexity of this algorithm is always $O(n^2)$, in the worst, average and best cases, consequently avoiding an unbalanced workload between the gpu threads

# Completly process

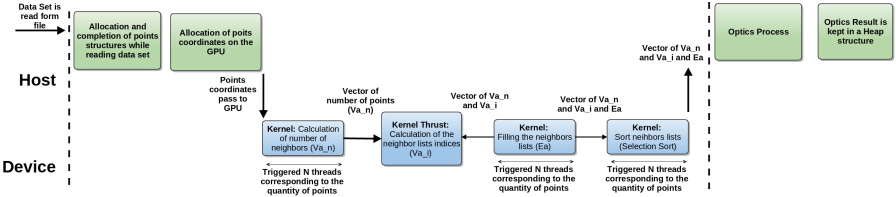



Figure: Computation and data transfers illustration.

# Summary

# Experimental Setup

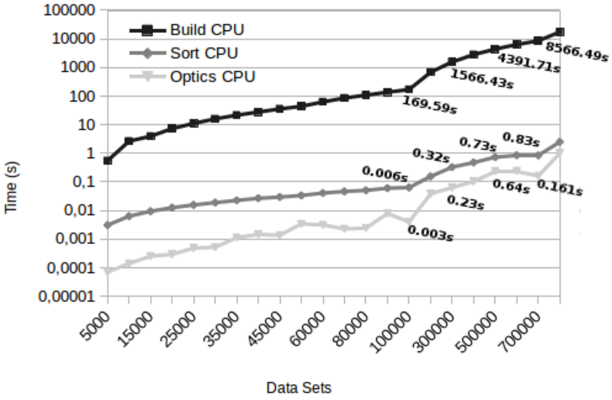
## Experimental Setup

- Input data set between $5,000$ and $700,000$ objects (2D).
- Execution times: Construction of the graph, Sorting process, OPTICS process. (CPU and GPU).
- Data sets with 20 randomly generated Gaussian clusters
- Parameters are fixed for all tests being $MinPts = 4$ and $\epsilon = 0.05$.
- Experimental Setup based in [**?**].

## Experimental Setup

- C and CUDA
- Intel Core i7-4930K 3.40GHz processor with 32GB of memory.
- Tesla K40c 12GB, with 2,880 CUDA cores.

# Profiling Execution

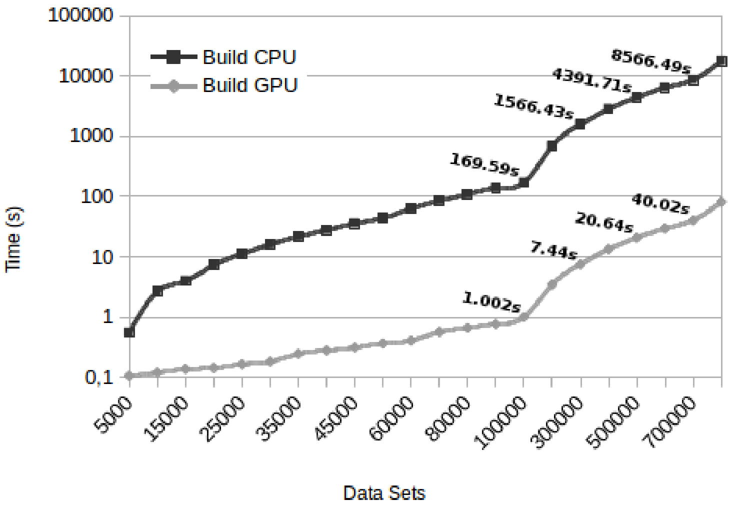



## Considerations

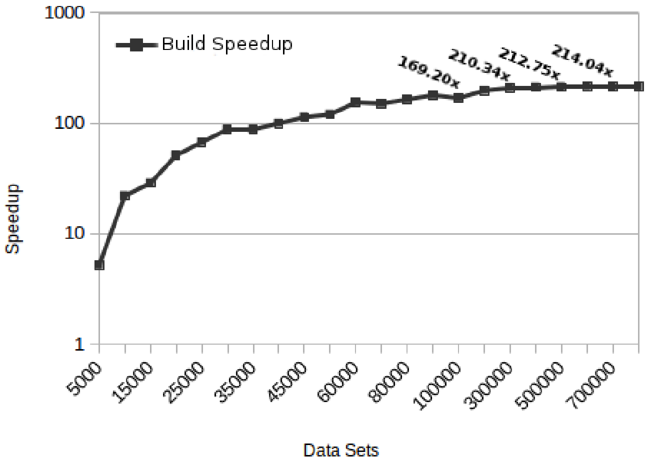
- It is easy to see that the graph construction dominates the execution time for all tested datasets, being 99,97% of the total time for the 700, 000 objects dataset

# Summary

# Graph construction evaluation



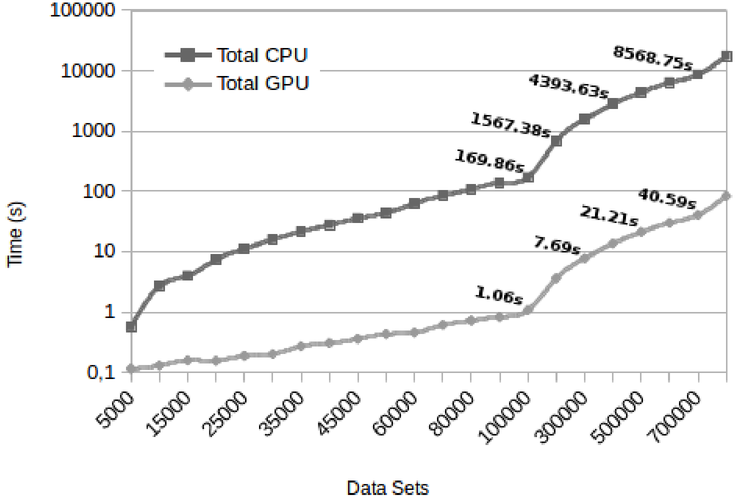
# Graph construction evaluation



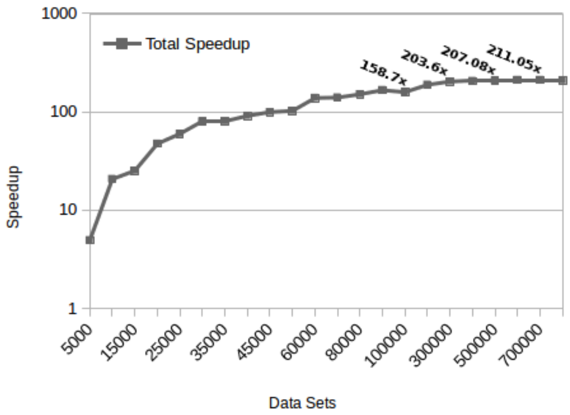

## Considerations

- For values of $N$ greater than $100,000$ the growth is less pronounced stabilizing around $N = 600,000$ with a $214\times$ speedup.

# Total time evaluation



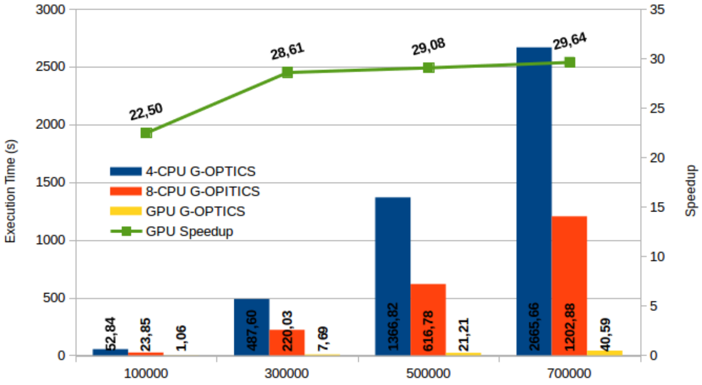
# Total time evaluation




## Considerations

- We can see that the the maximum speedup achieved was $211x$, decreasing the execution time from $8,568.75s$ on CPU to $40.59s$ on GPU, with $700,000$ objects

# Parallel Comparison




(a) Parallel Comparison

# Summary

# Conclusion

## Conclusions

- Presented G-OPTICS
- Efficient use of indexation for OPTICS
- Paralleliazation to speedup indexation construction.

## Future Works

- New parallel proposals for Optics
- PRIM's Minimum Spanning Tree algorithm
- Multiple GPUs
- Evaluate all these proposals on real data scenarios

# Thank you!

Questions?