Parallel Computing Approaches for Enhancing Agglomerative Hierarchical Clustering and Dijkstra's Algorithm

Subject: Distributed Computing -21ES634

C.Naga Sai Varun Kumar Reddy - CB.EN.P2EBS4006
K.sreenivas - CB.EN.P2EBS24023

Amrita School of Engineering, Coimbatore
Amrita Vishwa Vidyapeetham

April 15, 2025

Outline

Problem statement

Challenges in AHC

- High execution time due to sequential merging of clusters.
- Poor scalability when handling large datasets.
- Need for efficient distance matrix computation and takes more time for computation.

Challenges in Dijkstra's Algorithm

- Computational complexity increases with graph size.
- Need for faster shortest path computation in real-time applications.
- less efficient in case of large scale applications when you are executing them in serial

Outline

Literature Review

Implementing Parallel Computing to Enhance the Performance of K-Means Algorithm
Authors: Ranyah Taha, Sara Alshakrani, Abdallah Alqaddoumi (2021)
This paper describes about implementing parallel computing to enhance the performance of the K-Means algorithm by dividing the dataset into smaller chunks. These chunks are processed concurrently across multiple processors, with each processor calculating local centroids. In this paper they used different metrics to calclate the efficiency like speedup,Time and compared with the parallelised k means performance.

Distributed Hierarchal Clustering Algorithm Utilizing a Distance Matrix
Authors: Gavriel Yarmish, Philip Listowsky and Simon Dexter (2017)
In this paper they described different clustering methods like average link, Complete linkage, Centroids Method and their performance tables are mentioned to increase the quality of cluster,also described a distributed algorithm for the Lance–Williams and also discribed how to parallelise to increase the performance .

Literature Review

ParChain: A Framework for Parallel Hierarchical Agglomerative Clustering using Nearest-Neighbor Chain
Authors: Shangdi Yu, Yiqiu Wang, Yan Gu (2022)
This paper gives, a parallel framework for Hierarchical Agglomerative Clustering (HAC) using the Nearest-Neighbour Chain (NNC) method. Traditional HAC algorithms are computationally expensive and difficult to parallelize. optimizes nearest-neighbour searches and workload distribution, achieving significant speedups and scalability on multi-core architectures. The framework outperforms existing parallel HAC methods and is useful for large-scale data clustering applications in bioinformatics, social networks, and image processing.
.

A novel parallelization approach for hierarchical clustering
*Authors: Z. Du * , F. Lin (2005)*
A parallel method for hierarchical clustering using MPI is presented in this paper. This work highlight the distribution of the computationally expensive tasks, such as distance matrix calculation and cluster merging, among multiple processors. This is done by splitting the dataset into smaller subsets, performing partial results in parallel, and then computing the final results to accelerate the process and scale it up significantly. The MPI-based parallelization permits hierarchical clustering to process larger data sets more effectively, reducing the execution time, and making it more practical in real-world applications

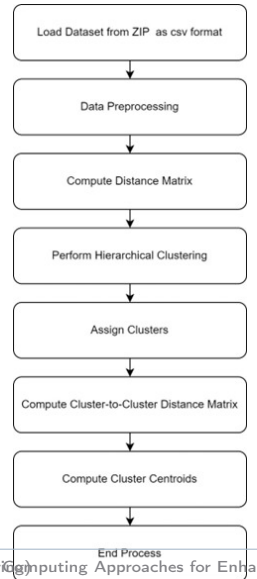
Outline

Objectives

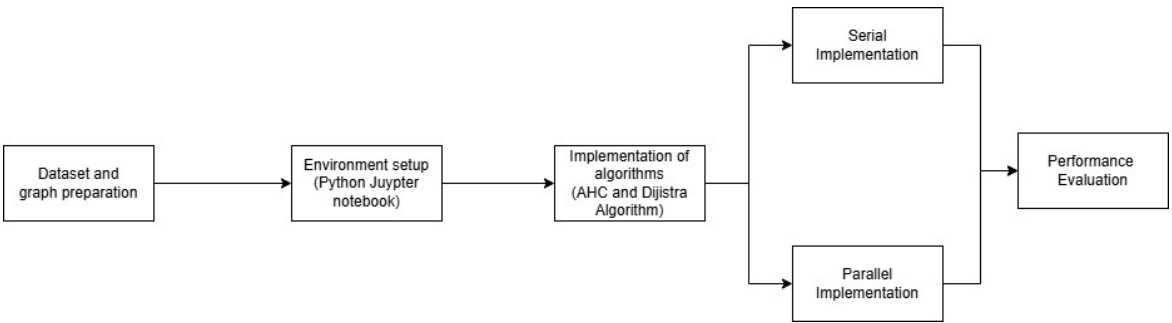
•To improve Agglomerative Hierarchical Clustering (AHC) quality by implementing techniques like ward method and centroid method.
•To optimize the shortest path computation by implementing a parallel version of Dijkstra's algorithm using MPI.
•To reduce execution time by using the MPI4py parallel processing library for both AHC and Dijkstra's algorithm.
•To analyze performance by evaluating metrics such as execution time, speedup, Silhouette score (for AHC),and path computation accuracy (for Dijkstra's algorithm) in both sequential and parallel implementations.

Outline

Block Diagram



Block diagram



Outline

Methodology

1. **Dataset Preparation:** Using the Dry Bean Dataset from UCI.
2. **Environment Setup:** Set up Python using (jupyter Notebook) with libraries such as `mpi4py`, `NumPy`, `csv`, `math`,`StandardScaler`, `pandas` and `time`.
3. **Agglomerative Hierarchical Clustering Implementation:**
   - 3.1 **Serial :** Perform Euclidian distance calculations, Constructing a hierarchical clustering tree using the Wards linkage method, and calculation of Silhouette Score (0.601)
   - 3.2 **Parallel :** Use MPI to distribute data across processes, compute distances in parallel, aggregate results and Time taken to perform.
4. **Performance Evaluation:** Measure execution time, speedup,Silhouette Score for both serial and parallel approaches to analyze cluster quality.

.

Methodology Contd..

Dijikstra Algorithm Methodology

1.Data Preparation:Construct a graph using an adjacency matrix from a Arff file

2. Environment Setup:Set up Python (Jupyter Notebook) with the required libraries like mpi4py, NumPy, Arff, math, time

3.Dijkstra's Algorithm Implementation:

Serial Approach:

3.1.1 Initialize distance array and visited set

3.1.2 Use a priority queue (Min-Heap)

3.1.3 Compute shortest paths from the source node

3.1.4 Measure execution time

Parallel Approach:

3.2.1 Use MPI to distribute graph data across multiple processes

3.2.2 Each process calculates partial shortest paths

3.2.3 Aggregate results and update global distance array and Measure Execution time

Methodology Contd...

4.Performance Evaluation: Compare execution time for serial vs. parallel approaches
Analyze speedup check that the values of distances in both serial and parallel execution are same

Result AHC without MPI contd..

```
Normalization completed in 0.0000 seconds
Distance Matrix computed in 0.2048 seconds
Hierarchical clustering using Ward's method completed in 0.1917 seconds
Silhouette analysis completed in 1.6580 seconds

Using Ward's Method:
Optimal number of clusters: 3
Silhouette score: 0.601

Cluster Sizes:
1    2025
2     152
3    1323
Name: count, dtype: int64

Linkage Matrix (First 5 Rows):
[[1.57500000e+03 1.58000000e+03 1.03396364e-02 2.00000000e+00]
 [5.65000000e+02 5.82000000e+02 1.06569969e-02 2.00000000e+00]
 [8.60000000e+02 8.94000000e+02 1.10758855e-02 2.00000000e+00]
 [8.55000000e+02 9.04000000e+02 1.13626905e-02 2.00000000e+00]
 [1.26800000e+03 1.30000000e+03 1.22639541e-02 2.00000000e+00]]
```

Figure: linkage matrix

```
Final Cluster Centroids:
          0         1         2         3         4         5         6   \
1  0.087290  0.122500  0.130304  0.140187  0.236941  0.571899  0.086236
2  0.905314  0.882704  0.860273  0.830508  0.529802  0.824696  0.900165
3  0.316177  0.456428  0.437881  0.315600  0.562704  0.838587  0.317204

          7         8         9         10        11        12        13  \
1  0.133183  0.656218  0.942186  0.883534  0.669482  0.728795  0.613973
2  0.930427  0.660799  0.909610  0.707495  0.353791  0.086330  0.073847
3  0.411819  0.546422  0.843479  0.520952  0.330482  0.496021  0.217261

         14        15
1  0.635717  0.973766
2  0.318727  0.878419
3  0.297372  0.923872

Cluster-to-Cluster Distance Matrix:
          0         1         2
0  0.000000  2.156422  1.092997
1  2.156422  0.000000  1.351273
2  1.092997  1.351273  0.000000
Visualization saved in 0.5711 seconds

========================================
Total execution time: 2.6573 seconds
========================================
```

Figure: finalclusters

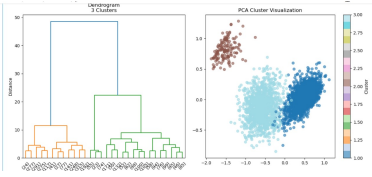


Figure: Dendogram and PCA visualization

Result AHC with MPI contd..

```
[Root] Hierarchical clustering completed in 0.1951 sec

[Root] Best number of clusters: 3
[Root] Best silhouette score: 0.6006

[Root] Cluster sizes:
1    2025
2     152
3    1323
Name: count, dtype: int64

[Root] Cluster centroids:
          0         1         2         3         4         5         6   \
1  0.087290  0.122500  0.130104  0.140187  0.236941  0.571899  0.086216
2  0.905314  0.882704  0.860273  0.830308  0.529802  0.824696  0.900165
3  0.316177  0.456428  0.437881  0.315600  0.562704  0.838587  0.317204

          7         8         9         10        11        12        13  \
1  0.133183  0.656218  0.942186  0.883534  0.669482  0.728795  0.613973
2  0.930427  0.660799  0.909610  0.707495  0.353791  0.086330  0.073847
3  0.411819  0.546422  0.843479  0.520952  0.330482  0.496021  0.217261

         14        15
1  0.635717  0.973766
2  0.318727  0.878419
3  0.297372  0.923872
```

Figure: final Clusters

```
[Root] Cluster-to-cluster distance matrix:
          0         1         2
0  0.000000  2.156422  1.092997
1  2.156422  0.000000  1.351273
2  1.092997  1.351273  0.000000
[Root] Visualization saved in 0.6990 sec

[Root] ========================================
[Root] Total execution time: 2.5299 seconds
[Root] ========================================
```

Figure: linkage matrix and Execution time

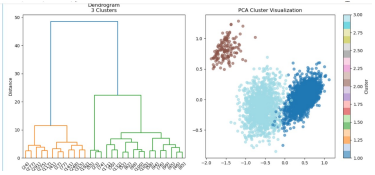


Figure: Dendogram and PCA visualization

Outline

Results

```
Final Shortest Paths from Node 0:
Node 0: 0.0000
Node 1: 0.3531
Node 2: 0.0700
Node 3: 0.2514
Node 4: 0.4610
Node 5: 0.0972
Node 6: 0.2669
Node 7: 0.1020
Node 8: 0.1038
Node 9: 0.1017
Node 10: 0.1277
Node 11: 0.2492
Node 12: 0.1204
Node 13: 0.1434
Node 14: 0.2019
Node 15: 0.1680
Node 16: 0.1170
...
```

Figure: Shortest path values

```
[Validation] Comparing Parallel and Serial Results...
[Validation]  PASS: Parallel and Serial results match.

Serial Execution Time: 2.7380 seconds
Parallel Execution Time: 1.9292 seconds
Speedup Achieved: 1.42x
```

Figure: Performance metrics

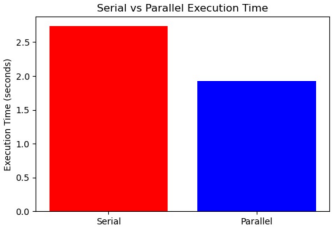


Figure: execution time comparison

Result contd..

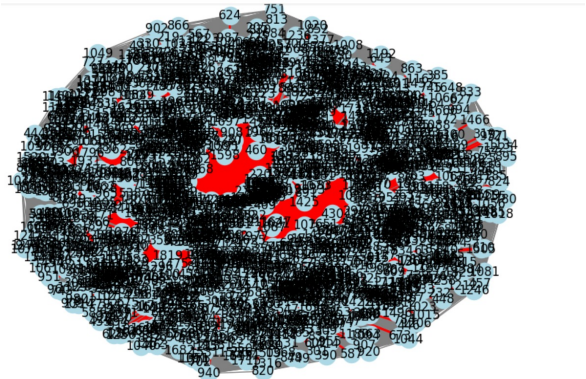


Figure: Graph visualization with shortest paths

Result contd..

```
Speedup achieved: 1.05x
```

Figure: Speedup in AHC algorithm

Algorithm	Parallel execution time	Serial execution time
Dijkstra Algorithm	1.92	2.73
AHC Algorithm	2.5	2.65

Figure: Parallel vs Serial Execution Time

Outline

Conclusion

This project successfully demonstrates how parallel computing with MPI can improve Agglomerative Hierarchical Clustering (AHC) performance. By leveraging mpi4py, we achieved a noticeable speedup (5 percent), although the gain was limited due to several MPI overheads. Communication overhead, load imbalance, memory duplication, and the single-process clustering bottleneck in linkage() all contributed to reduced efficiency. Despite these challenges, the parallel implementation maintained accuracy and scalability, making it suitable for larger datasets and real-time applications. The 5 percent speedup highlights the potential of parallelism, though further optimizations are needed to fully capitalize on MPI's capabilities.

Outline

Time Frame

Sl.No	Task Name	Project				
		DEC	JAN	FEB	MAR	April
1	Selection of topic& Literature Survey					
2	Data Pre processing					
3	Performing AHC And Dijkstra algorithms sequentially					
4	Parallelizing the Algorithm using MPI					
5	Comparing results of sequential as well as parallel					
6	Report writing					

Outline

References

- Taha, R., Alshakrani, S., Alqaddoumi, A. (2021). Implementing Parallel Computing to Enhance the Performance of K-mean Algorithm. 2021 International Conference on Data Analytics for Business and Industry (ICDABI), pp. 140-143. doi:10.1109/ICDABI53623.2021.9655811
- Distributed Hierarchal Clustering Algorithm Utilizing a Distance Matrix December 2017 DOI:10.1109/CSCI.2017.282
- Shangdi Yu, Yiqiu Wang, Yan Gu, Laxman Dhulipala, and Julian Shun, "ParChain: A Framework for Parallel Hierarchical Agglomerative Clustering using Nearest-Neighbor Chain," Proceedings of the 39th International Conference on Machine Learning (ICML), 2022
- Du, Z., Lin, F. (2005). A novel parallelization approach for hierarchical clustering. BioInformatics Research Centre, Nanyang Technological University, Singapore
- Awari, R. (2017). Parallelization of Shortest Path Algorithm Using OpenMP and MPI. Proceedings of the 2017 IEEE (ICCIC).