Data Mining Algorithms
Parallelizing K-Means with MPI

Eleonora Renz, 239020

1. Introduction

Objectives

- Parallelize the most popular [1] clustering algorithm using MPI
- Reduce computational time
- Gain deeper understanding of MPI, HPC, and C

[1] P. Berkhin. 2006. A Survey of Clustering Data Mining Techniques. Springer Berlin Heidelberg, Berlin, Heidelberg, 25–71.

State of the Art

- Well studied algorithm
- Parallelization through MPI & OpenMP
  → No code, nor data found to compare this solution to

2. Problem Analysis

K-Means

Pseudocode

Algorithm 1 Serial k-means

```
1: Set initial centroids k = <k_1, k_2, ..., k_k>
2: Calculate distance of each point to the k centroids
3: Assign each point to a cluster based on the shortest distance
4: Calculate new centroids based on cluster members
5: If k and k' are different
6: Set k to be k' and repeat from step 2
7: Else stop
```

3. Parallel Solution Design

K-Means

Pseudocode

Algorithm 3 Parallel k-means

```
Set initial centroids k = <k_1, k_2, ..., k_k>
Split data amongst s processes into s subgroups of equal size
for every subgroup do
    Calculate distance of each point to the k centroids
    Assign each point to a cluster based on the shortest distance
end for
Gather all points with their cluster associations
Calculate new centroids based on cluster members
If k and k' are different
Set k to be k' and repeat from step 3
Else stop and return all points with their cluster association
```

Parallelize with MPI

In the root process:

- Read data
- Set k initial centroids

Use collective communication calls to:

- Scatter data points across different processes with MPI_Scatter
  → custom MPI_Datatype was defined for sending struct data
- Broadcast initial centroids to all processes

```
typedef struct point
{
    double x[6]; // data points
    int cluster; // cluster association
} point;
```

In each process:

- Find the euclidean distance between all data points in the subset with the k centroids
- Assign points to a cluster based on shortest distance to a centroid

Use collective communication call to:

- Gather data points with their new cluster association in the root process with MPI_Gather

Parallelize with MPI

In the root process:

- Re-calculate centroids based on cluster members of gathered data
- Check if centroids have moved

If centroids have moved, use collective communication calls to:

- Broadcast new centroids with MPI_Bcast
- Repetition of previous points for all processes up until a max iteration limit

Else, if centroids have not moved, use collective communication calls to:

- Break the loop in each process by broadcasting an indicating variable with MPI_Bcast

4. Benchmarking

K-Means

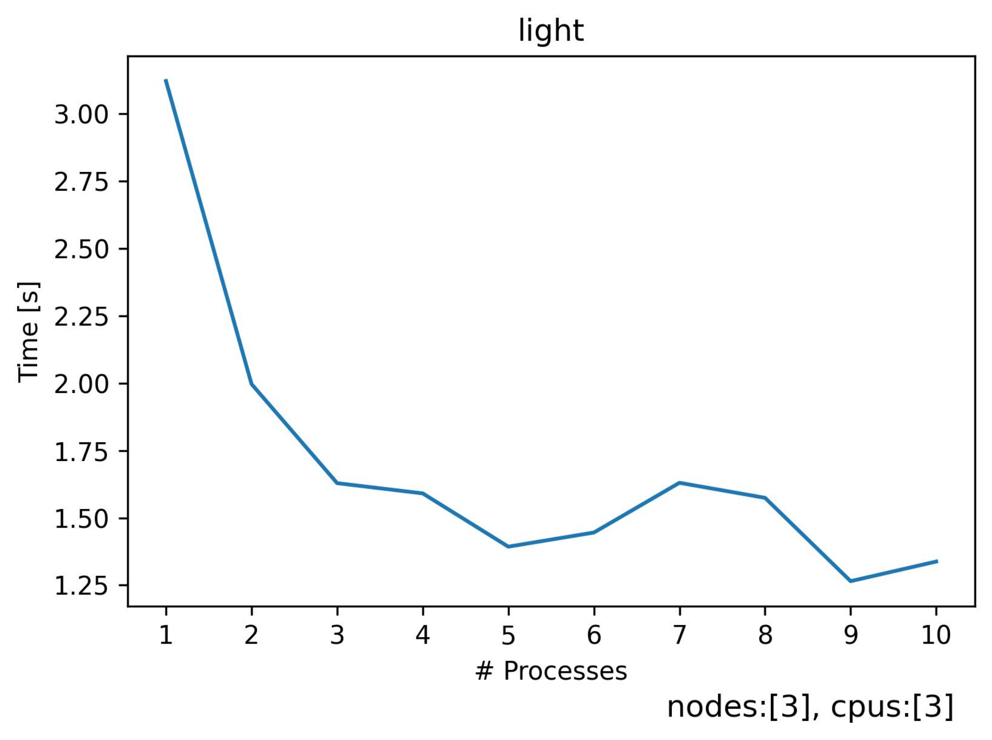
Benchmark Experiments

1. Light data set size (8950 rows, 6 columns) with a fixed size of nodes and CPUs while increasing the number of processes used.
2. Heavy data set size (17950 rows, 6 columns) with a fixed size of nodes and CPUs while increasing the number of processes used.
3. Heavy data set size with increasing nodes along number of processes while keeping CPUs fixed.
4. Heavy data set size with increasing CPUs along number of processes while keeping nodes fixed.

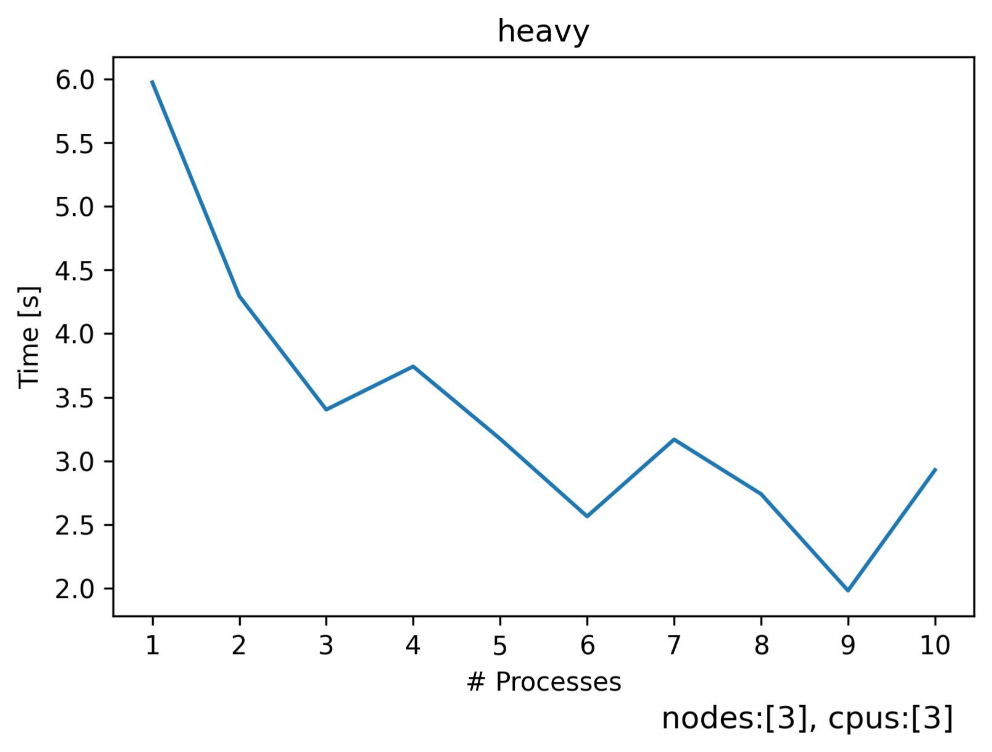
- No benchmarking on I/O
- Do 10 runs of every job and take median result

Experiment	Dataset size	Nodes	CPUs	Processes
Light	501200B	3	3	1–30
Heavy	1005200B	3	3	1–30
Heavy Nodes	1005200B	1-30	1	1–30
Heavy CPUs	1005200B	1	1-30	1–30

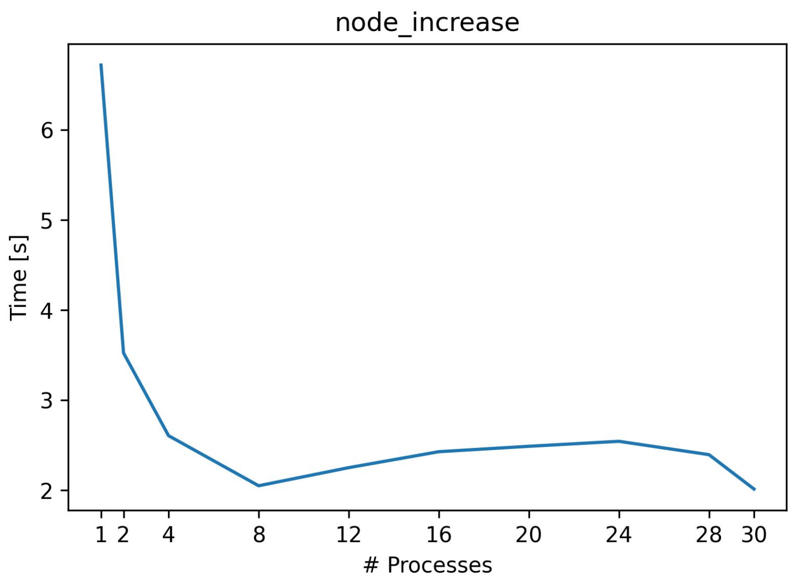
Results Experiment 1



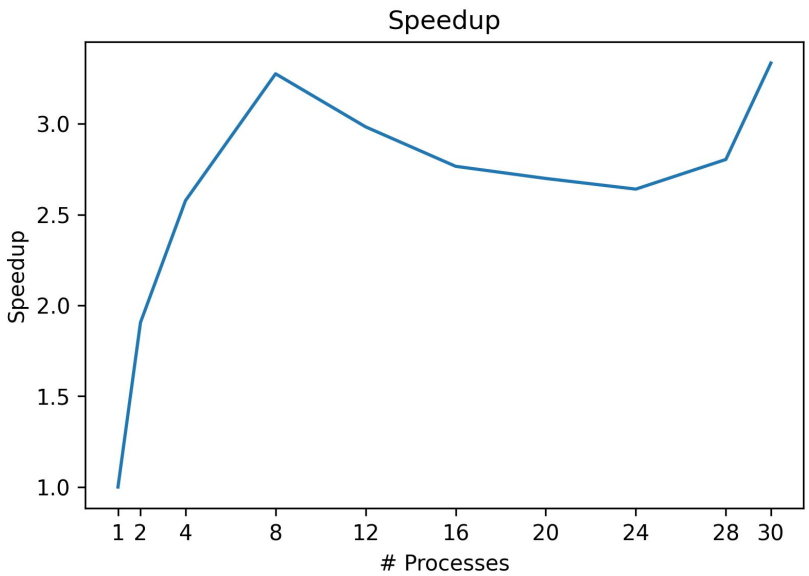
Results Experiment 2



Results Experiment 3

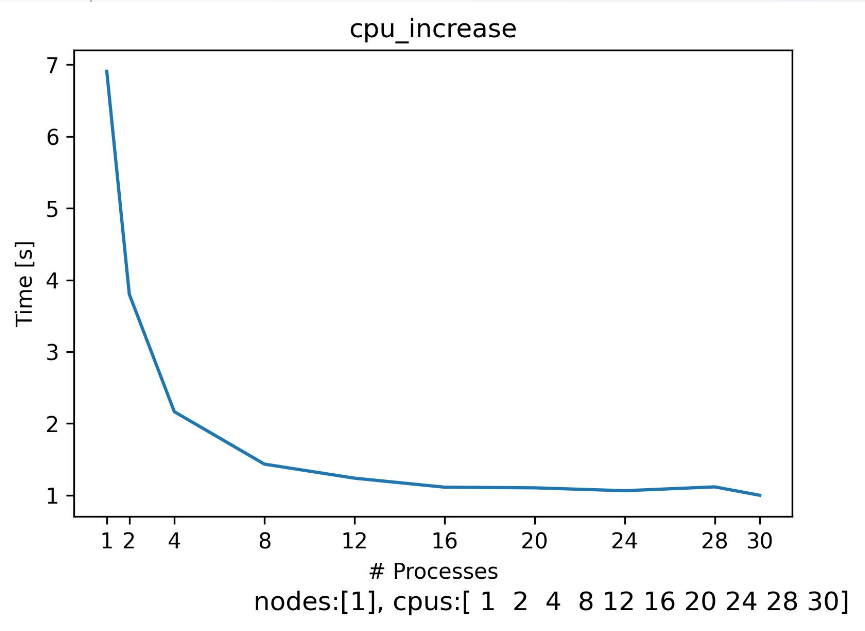


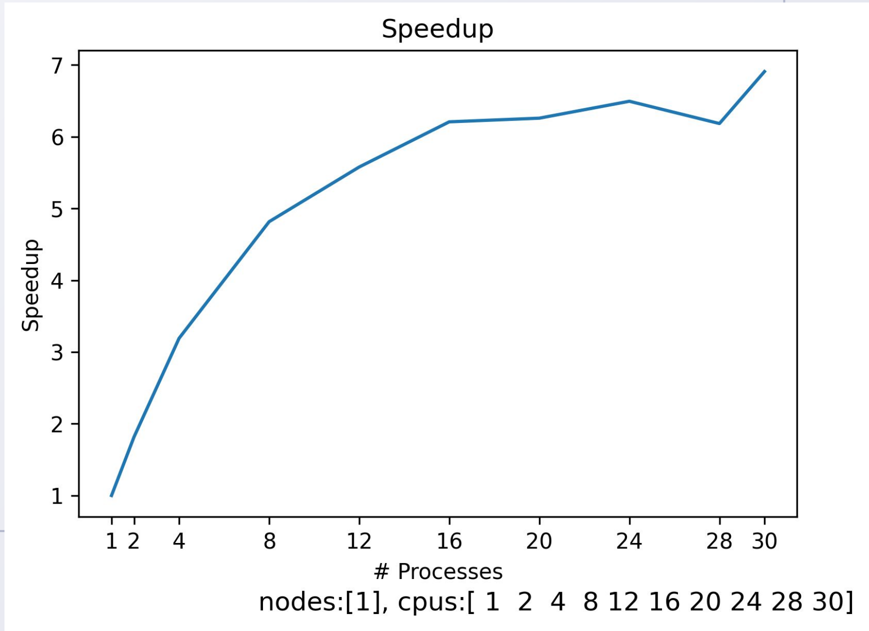
nodes:[1 2 4 8 12 16 20 24 28 30], cpus:[1]



nodes:[1 2 4 8 12 16 20 24 28 30], cpus:[1]

Results Experiment 4





5. Conclusion

K-Means

Conclusion

Learning Outcomes

- Understanding of MPI, HPC, and C

Room for Improvement

- Improve existing parallelization
  → calculate sub-centroids in each process to then reduce to one centroid in root
  with custom MPI_Op for MPI_Redcue
- Use bigger dataset

Further Research

- Try different parallelization approach
- Implement OpenMP version