A Parallel approach to K-Means with OMP and MPI

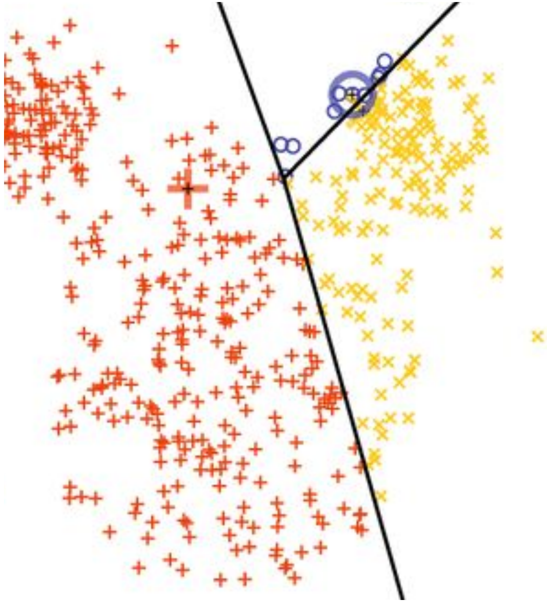
Introduction to Parallel Computing

Authors: Elia Zonta, Alex Pegoraro

What is K-Means

A geometric clustering [1] algorithm with applications such as:

- Data analysis
- Pattern recognition
- Image segmentation
- …

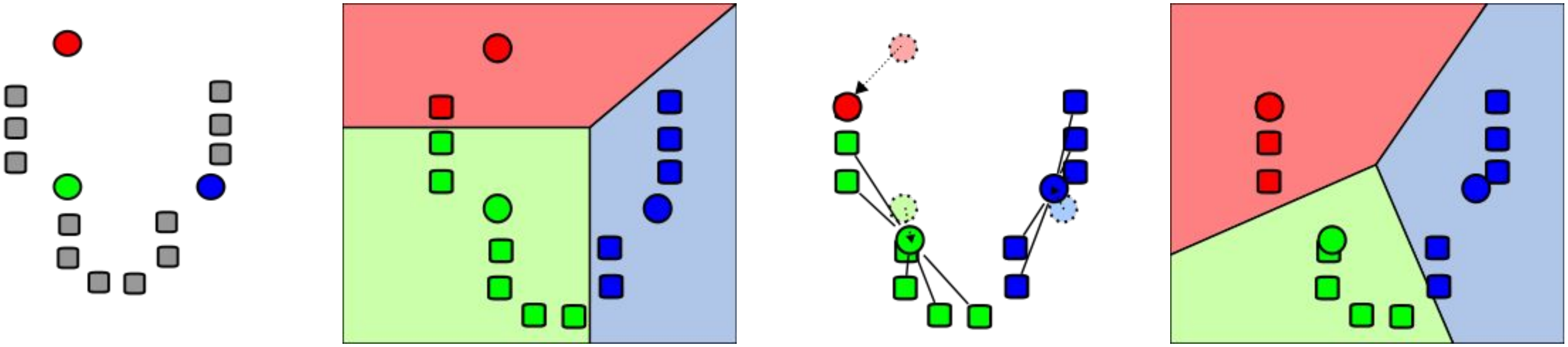


Authors: Elia Zonta, Alex Pegoraro

Objective

Our goal is to apply different parallelization techniques to K-Means, studying the effects, benefits and drawbacks.
Every experiments has been done on the UniTN-HPC cluster

The Algorithm [1]



The Algorithm [2]

```
random_init_centroids(points, centroids)

for e in epochs:
	assign_points_to_cluster(points, clusters)
	cumulative = sum_points_in_cluster(clusters)
	compute_centroids(cumulative)
	if delta < tolerance:
		break // convergence reached
```

The Dataset

- Synthetic data generator script
- Uniform distribution
- Arbitrary number of features
- Arbitrary number of entries
- Arbitrary upper and lower bounds
- Both data points and initial centroids generated with it, to ensure fairness

```
std::uniform_real_distribution<double> uniform();
```

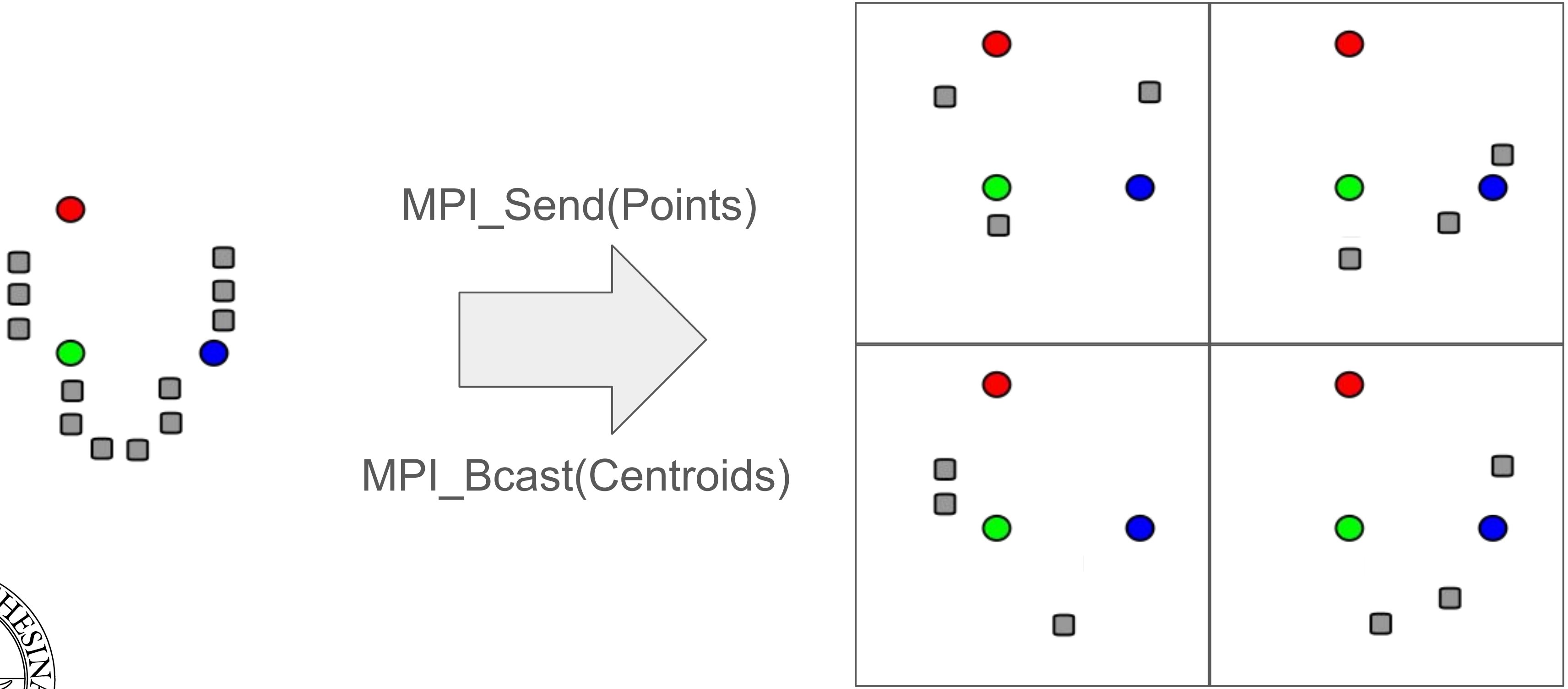
The Parallel Approach (OpenMP)

```
random_init_centroids(points, centroids)

for e in epochs:
    #pragma omp parallel for schedule(static, static_cast<int>(...)) reduction(+:delta)
    assign_points_to_cluster(points, clusters)
    #pragma omp parallel for schedule(static, static_cast<int>(...)) reduction(+:delta)
    cumulative = sum_points_in_cluster(clusters)
    #pragma omp parallel for schedule(static, n_features)
    compute_centroids(cumulative)
    if delta < tolerance:
        break // convergence reached
```

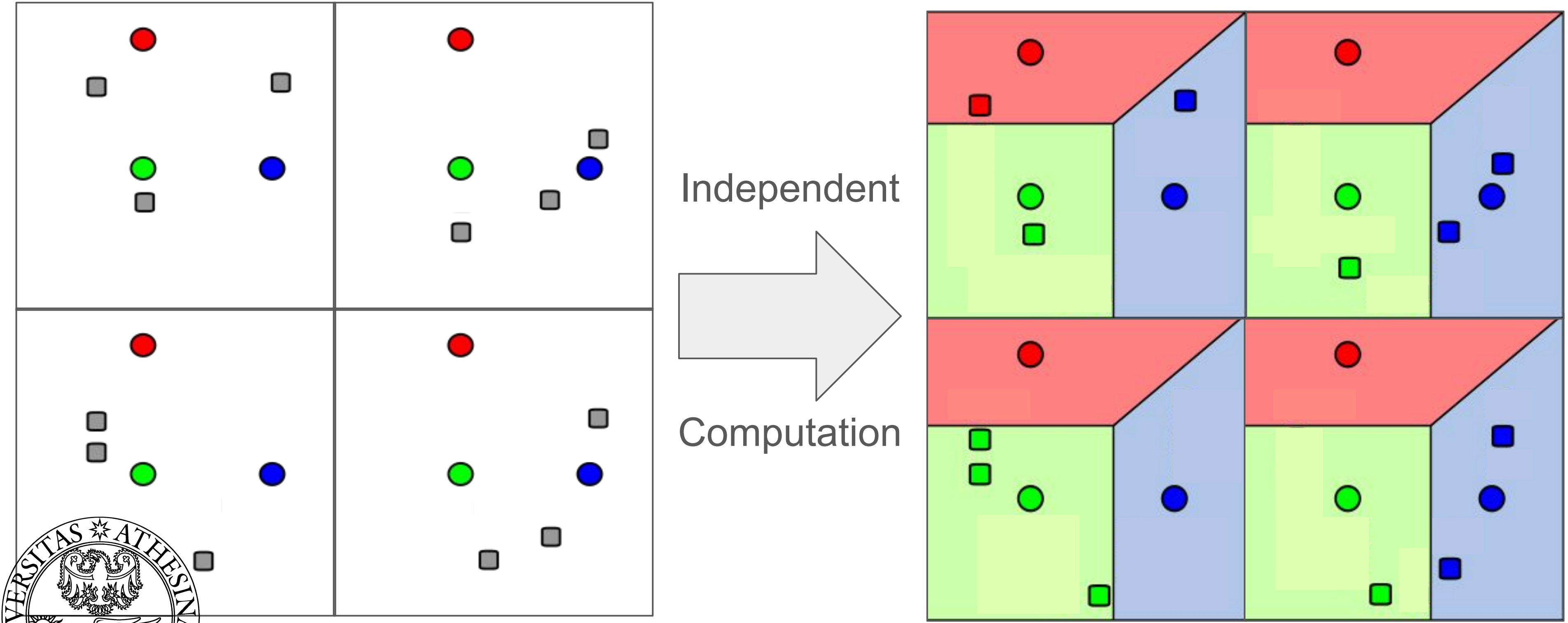
The Parallel Approach (MPI) [1]

```
random_init_centroids(points, centroids)
if (Master) MPI_Send(points) else  MPI_Recv(points)
MPI_Bcast(centroids)

for e in epochs:
        assign_poits_to_cluster(points, clusters)
        cumulative = sum_points_in_cluster(clusters)
        MPI_Allreduce(cumulative)
        compute_centroids(cumulative)
        if delta < tolerance:
                break // convergence reached
```

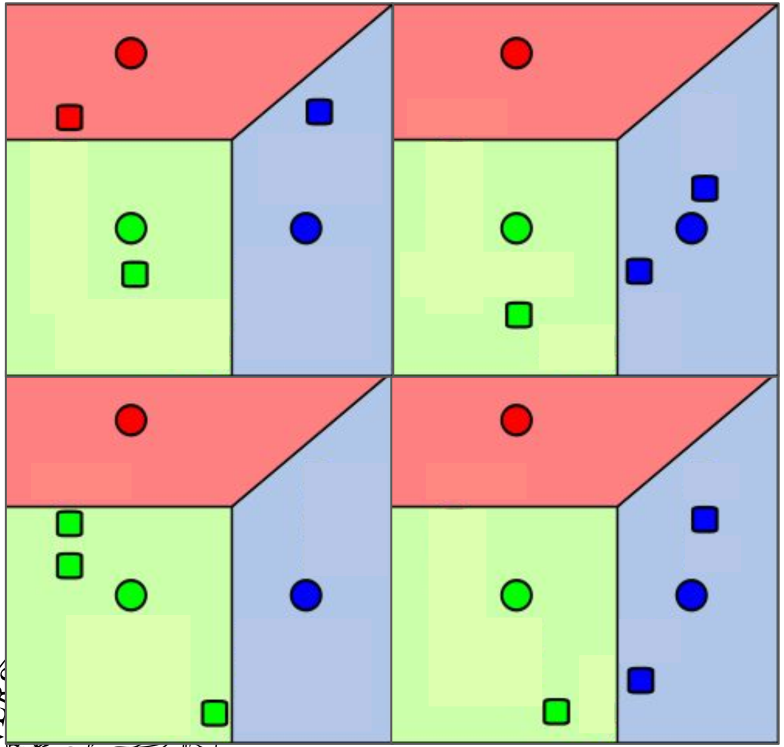
The Parallel Approach (MPI) [2]



The Parallel Approach (MPI) [3]

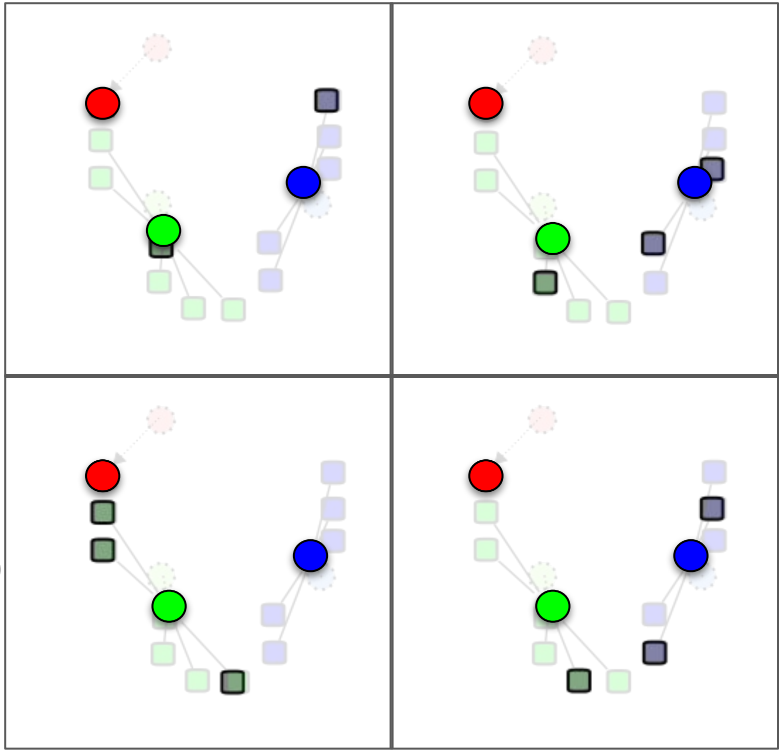


The Parallel Approach (MPI) [4]



MPI_Allreduce (cumulative)

MPI_Allreduce (point_counter)



Authors: Elia Zonta, Alex Pegoraro

Correctness

```
Base file: out/serial_out.csv
n_points: 65536, n_clusters: 64, n_dimensions: 8, tolerance: 0.001
Comparison with files in: out

mpi_asynch_strong_out_1.csv: points OK, centroids OK
mpi_asynch_strong_out_128.csv: points OK, centroids OK
mpi_asynch_strong_out_16.csv: points OK, centroids OK
mpi_asynch_strong_out_2.csv: points OK, centroids OK
mpi_asynch_strong_out_256.csv: points OK, centroids OK
mpi_asynch_strong_out_32.csv: points OK, centroids OK
mpi_asynch_strong_out_4.csv: points OK, centroids OK
mpi_asynch_strong_out_64.csv: points OK, centroids OK
mpi_asynch_strong_out_8.csv: points OK, centroids OK
mpi_asynch_weak_out_1.csv: LESS points
mpi_asynch_weak_out_128.csv: MORE points, 64509 DIFFERENT points, 64 DIFFERENT centroids
mpi_asynch_weak_out_16.csv: LESS points
mpi_asynch_weak_out_2.csv: LESS points
mpi_asynch_weak_out_256.csv: MORE points, 64510 DIFFERENT points, 64 DIFFERENT centroids
mpi_asynch_weak_out_32.csv: LESS points
mpi_asynch_weak_out_4.csv: LESS points
mpi_asynch_weak_out_64.csv: 64543 DIFFERENT points, 64 DIFFERENT centroids
```

Computing System

UniTrento@HPC Cluster:

- 142 CPU nodes for a total of 7.674 cores
- 10 GPU nodes for a total of 48.128 CUDA cores
- 2 frontend nodes
- 65 TB of Ram
- Total theoretical peak performance: 478,1 TFLOPs
- Theoretical peak performance CPU: 422,7 TFLOPs
- Theoretical peak performance GPU: 55,4 TFLOPs
- 10Gb/s network
- Both nodes with Infiniband and Omni-Path connectivity.
- Linux CentOS 7 and PBS as the cluster workload manager.

Benchmark

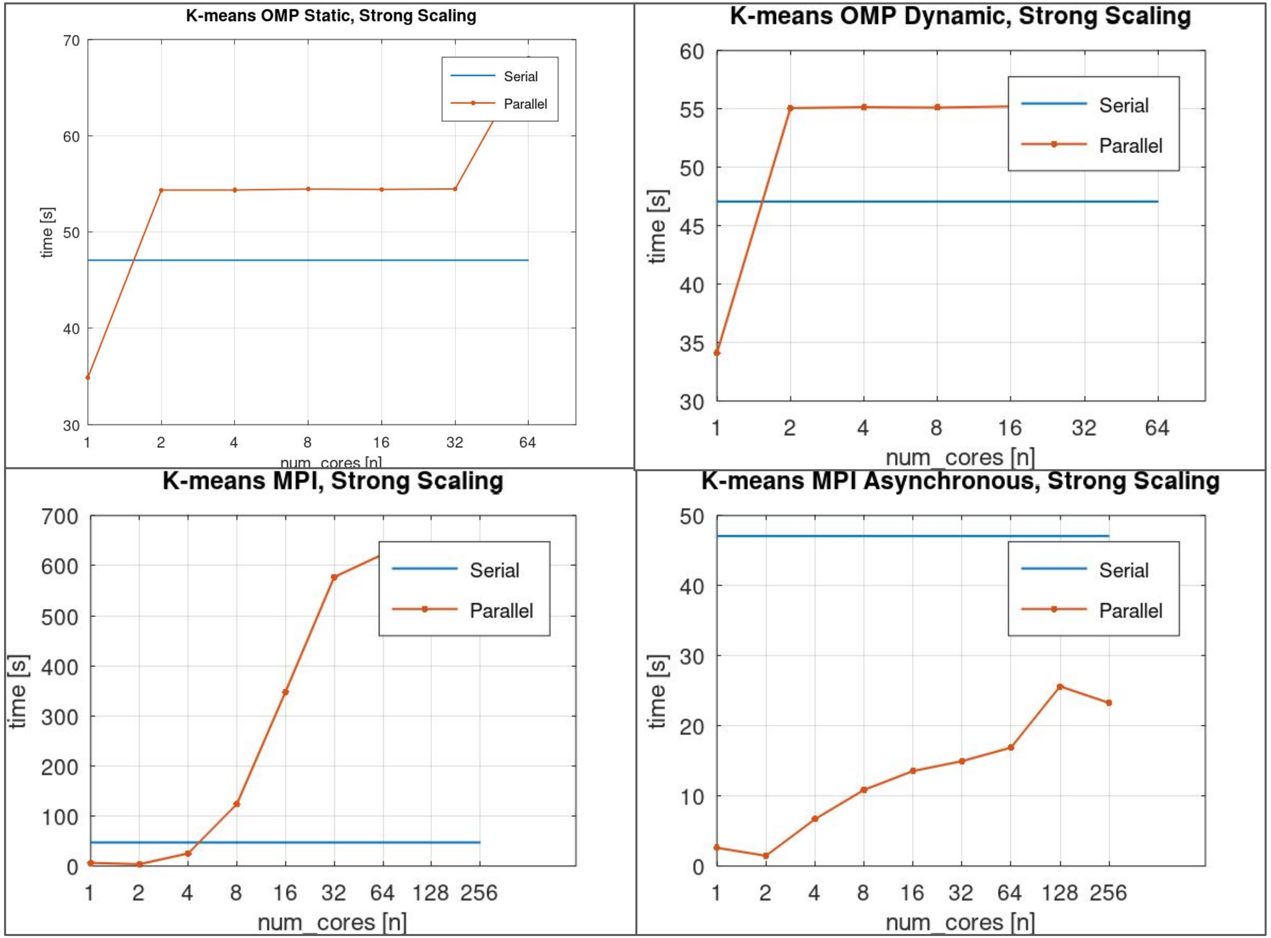
For strong scaling and serial algorithm:

- 65536 data points
- 64 centroids
- 8-dimensional feature space
- 128 epochs
- up to 64 OMP threads
- up to 256 MPI processors

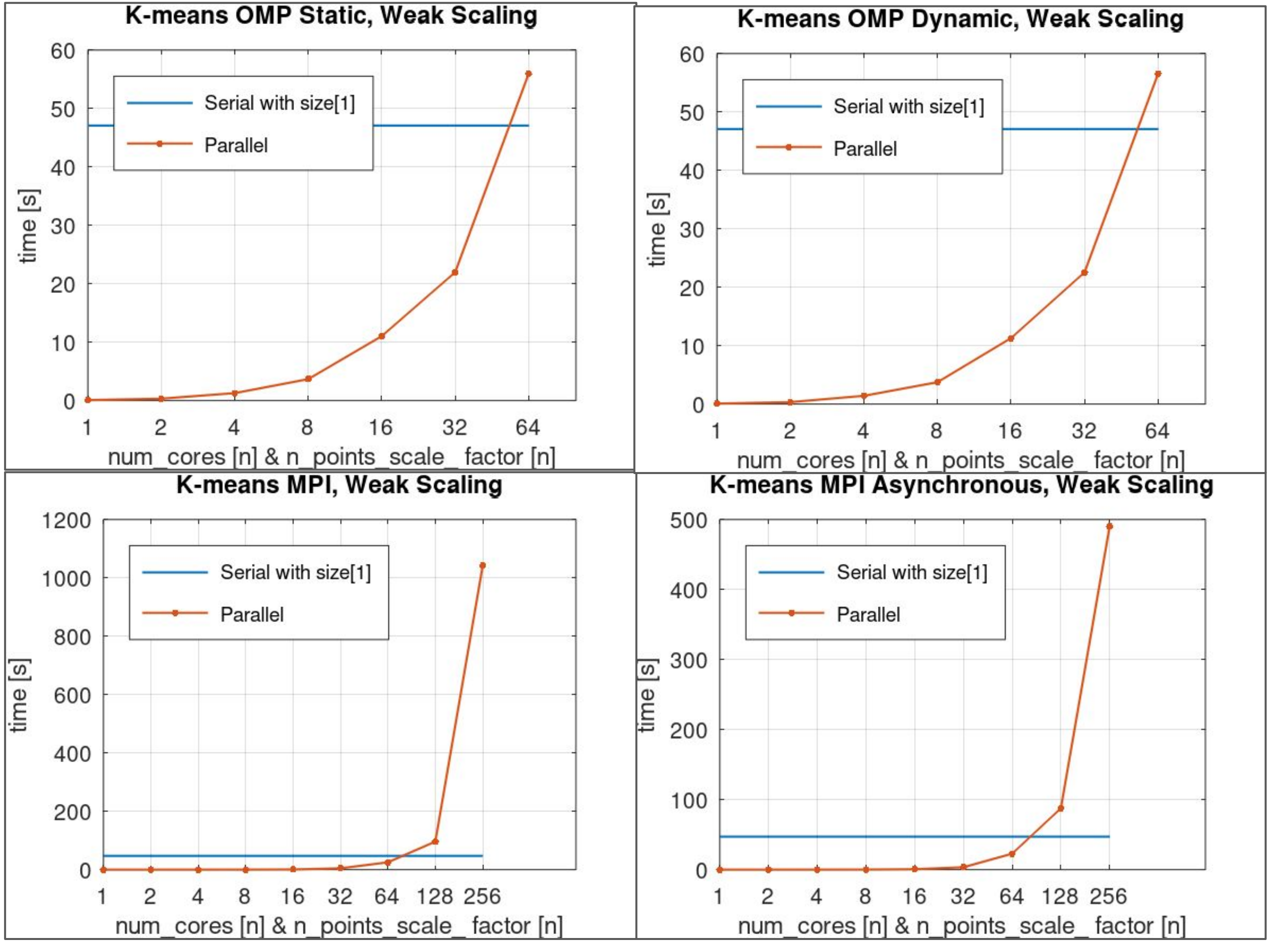
For weak scaling:

- baseline of 1024 data points, increased accordingly to core size
- other parameters identical to strong scaling

Results[1]


K-means OMP Static, Strong Scaling
K-means OMP Dynamic, Strong Scaling
K-means MPI, Strong Scaling
K-means MPI Asynchronous, Strong Scaling
Serial
Parallel
time [s]
num_cores [n]

Results[2]



K-means OMP Static, Weak Scaling
time [s]
num_cores [n] & n_points_scale_ factor [n]
Serial with size[1]
Parallel
K-means OMP Dynamic, Weak Scaling
time [s]
num_cores [n] & n_points_scale_ factor [n]
Serial with size[1]
Parallel
K-means MPI, Weak Scaling
time [s]
num_cores [n] & n_points_scale_ factor [n]
Serial with size[1]
Parallel
K-means MPI Asynchronous, Weak Scaling
time [s]
num_cores [n] & n_points_scale_ factor [n]
Serial with size[1]
Parallel

Conclusion

The main bottleneck of K-Means parallelization is represented by the **initial messages** required to **distribute points**.

Pictures and References

Slide 1:[1] https://theory.stanford.edu/~sergei/papers/kMeans-socg.pdf
Slide 2:
https://commons.wikimedia.org/wiki/File:K-means_convergence.gif
Slide 4:
https://commons.wikimedia.org/wiki/File:K_Means_Example_Step_1.svg
https://commons.wikimedia.org/wiki/File:K_Means_Example_Step_2.svg
https://commons.wikimedia.org/wiki/File:K_Means_Example_Step_3.svg
https://commons.wikimedia.org/wiki/File:K_Means_Example_Step_4.svg
Slides 9,10,11: Same four pictures of slide 4, but crafted.

Thanks for the attention !

A Parallel approach to K-Means with OMP and MPI

Introduction to Parallel Computing

Authors: Elia Zonta, Alex Pegoraro