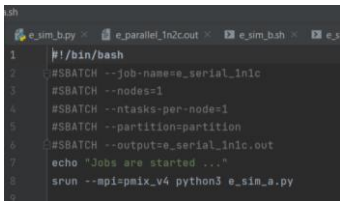
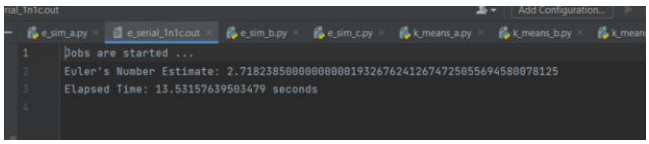
محمدباربد امیرمزلقانی - ۸۱۰۱۰۲۳۴۸

سوال ۱:

قسمت الف)

ابتدا الگوریتم را به صورت سریال پیاده‌سازی میکنیم، دقیقا همان تولید عدد رندوم بین صفر تا یک تا وقتی که بیشتر از یک شود(به کمک while) سپس میانگین گیری بین این تعدادها که کد و خروجی را در ادامه میبینیم:





قسمت ب)

در این قسمت کد تا حد بسیار زیادی مانند قبل است با این تفاوت که این عمل شمردن در بین هسته های پردازشی پخش میشود و هر کدام بخشی از محاسبات را انجام میدهند همانطور که مشاهده میشود (و انتظار میرود) زمان پردازش کاهش یافته است :

```python
from mpi4py import MPI
import random
import time
from decimal import Decimal, getcontext

comm = MPI.COMM_WORLD
my_rank = comm.Get_rank()
num_processes = comm.Get_size()

total_iterations = 4_000_000
local_counter = 0

getcontext().prec = 40

if my_rank == 0:
    start = time.time()

for _ in range(int(total_iterations // num_processes)):
    running_total = 0
    while running_total <= 1:
        running_total += random.uniform(0, 1)
        local_counter += 1
total_count = comm.reduce(local_counter, root=0)

if my_rank == 0:
    print(f"Euler: {Decimal(total_count / total_iterations)}")
    print(f"time: {time.time() - start} (s)")
```

```
#!/bin/bash
#SBATCH --job-name=e_parallel_1n2c
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=2
#SBATCH --partition=partition
#SBATCH --output=e_parallel_1n2c.out
echo "Jobs are started ..."
srun --mpi=pmix_v4 python3 e_sim_b.py
```

```
Jobs are started ...
Euler: 2.71831675000000005937295100011397153139114379882812 5
time: 8.266072988510132 (s)
```

قسمت ج)

کد این قسمت دقیقا برابر با کد قسمت ب سوال است ولی در bash script تغییراتی ایجاد میکنیم تا روی ۲ نود اجرا شود و طبق انتظار رانتایم برنامه پایینتر میاید.

```
#!/bin/bash
#SBATCH --job-name=e_parallel_2n2c
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --partition=partition
#SBATCH --output=e_parallel_2n2c.out
echo "Jobs are started ..."
srun --mpi=pmix_v4 python3 e_sim_c.py
```

```
Jobs are started ...
Euler: 2.718545250000000024215296434704214334487915039062 5
time: 6.322283636169434 (s)
```

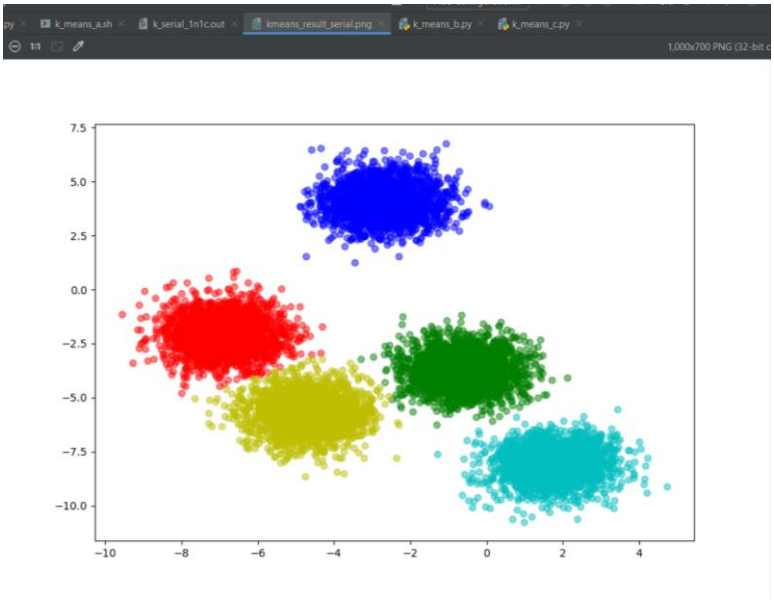
سوال ۲:

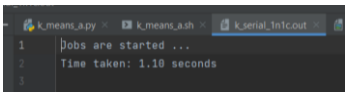
قسمت الف)

ابتدا الگوریتم را به صورت سریال مینویسیم، در ادامه ابتدا کمی راجب صحبت میکنیم بعد از آن به سراغ نتایج و خوشه ها میرویم. اولین چالشی که با آن روبرو بودیم انتقال دیتاست بود که با اندکی جست و جو به دستور scp رسیدیم پس از آن ابتدا دیتا را لود میکنیم و سپس به صورت رندوم مراکز را انتخاب میکنیم (رندوم یک سری اندیس ایجاد کردیم) پس از آن در هر iteration فاصله اقلیدسی نقاط از هر مرکز را حساب میکنیم به کوتاه ترین فاصله assign میکنیم و بعد از آن مرکز جدید را میانگین نقاط با لیبل آن مرکز قرار میدهیم.(این سه خط داخل for هستند) و در آخر نمودار را رسم و برای هر خوشه یک رنگ خاص انتخاب میکنیم زمان را هم خروجی میدهیم.

```python
all_points = dataset.to_numpy()

num_clusters = 5
iterations = 100

initial_indices = np.random.choice(all_points.shape[0], num_clusters, replace=False)
cluster_centers = all_points[initial_indices]

for iteration in range(iterations):
    distance = np.linalg.norm(all_points[:, np.newaxis] - cluster_centers, axis=2)
    closest = np.argmin(distance, axis=1)
    cluster_centers = [all_points[closest == cluster_idx].mean(axis=0) for cluster_idx in range(num_clusters)]

fig, ax = plt.subplots(figsize=(10, 7))
palette = ['r', 'g', 'b', 'y', 'c', 'm']

for idx in range(num_clusters):
    ax.scatter(all_points[closest == idx].T[0], all_points[closest == idx].T[1], c=palette[idx], alpha=0.5)
    ax.scatter(cluster_centers[idx][0], cluster_centers[idx][1], color=palette[idx], marker='x')

fig.savefig('kmeans_result_serial.png')

end_time = time.time()
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time:.2f} seconds")
```

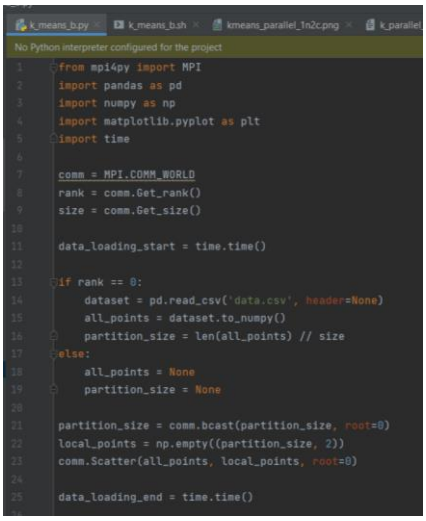
```
#!/bin/bash
#SBATCH --job-name=k_serial_1n1c
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --partition=partition
#SBATCH --output=k_serial_1n1c.out
echo "Jobs are started ..."
srun --mpi=pmix_v4 python3 k_means_a.py
```

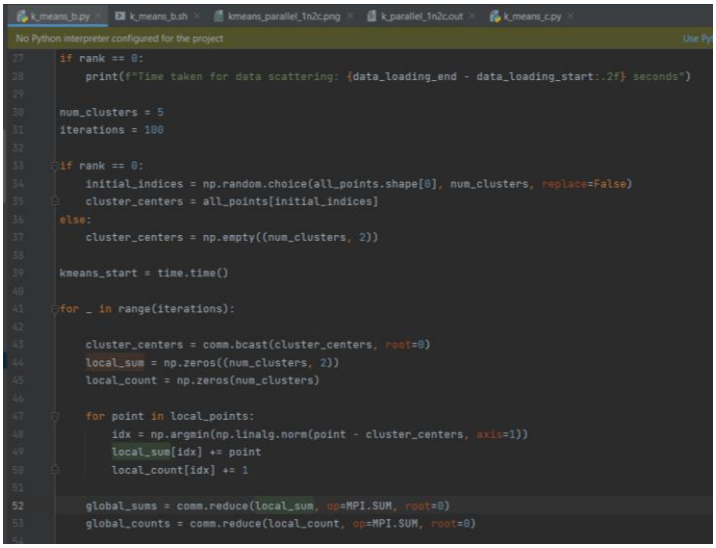


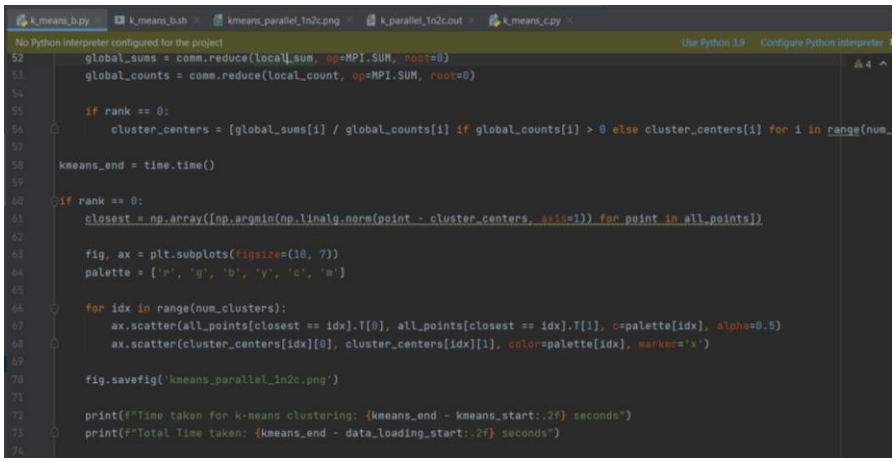


قسمت ب)

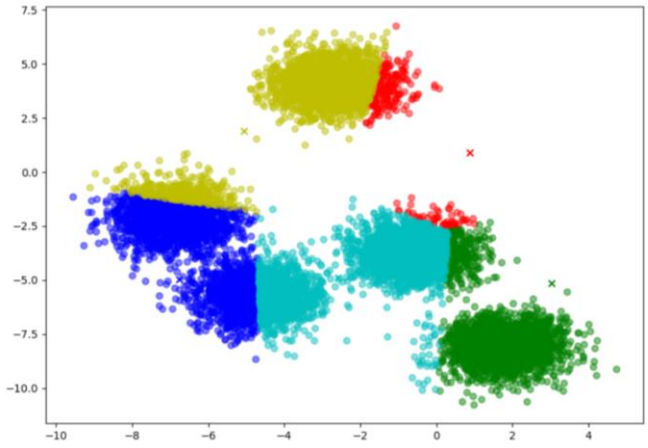
در این قسمت سراغ پیاده سازی آن به صورت موازی طبق توضیحات داک میرویم، تفاوتی که داریم این است که نود master دیتارا میخواند سایزی که هر هسته باید دریافت کند را محاسبه میکند سپس all_points را بین local_points ها تقسیم میکنیم،سپس مستر به صورت رندوم مرکزها مشخص میکند در ادامه kmean مانند مرحله قبل است با این تفاوت که طبق خواسته سوال برای هر هسته و هر مرکز خوشه جمع و count را جدا حساب میکنیم(خط ۴۷ تا ۵۳)، سپس مجموع این ها دوباره به نود مستر، در آنجا دوباره محاسبه مراکز جدید را داریم(خط ۵۶) و ارسال دوباره همگانی پس از آن تا هنگامی که تعداد تکرار به ۱۰۰ برسد. زمان اجرا هم تا حد زیادی بالا رفت دلیل آن شامل موارد زیر است : overhead زیاد در ارتباطات برای مثال ما با ۱۰۰ iteration برنامه ران کردیم و در هر iteration نیاز به ارسال و جمع آوری داده از نود های کارگر و ارسال به مستر همچنین دوباره باز ارسال به نود های کارگر میباشیم(مثلا اگر طبق خواسته سوال با ۱۰ iteration ران میکردیم این اختلاف کمتر بود)


k_means_b.py
k_means_b.sh
kmeans_parallel_1n2c.png
No Python interpreter configured for the project
from mpi4py import MPI
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
data_loading_start = time.time()
if rank == 0:
dataset = pd.read_csv('data.csv', header=None)
all_points = dataset.to_numpy()
partition_size = len(all_points) // size
else:
all_points = None
partition_size = None
partition_size = comm.bcast(partition_size, root=0)
local_points = np.empty((partition_size, 2))
comm.Scatter(all_points, local_points, root=0)
data_loading_end = time.time()


k_means_b.py
k_means_b.sh
kmeans_parallel_1n2c.png
k_parallel_1n2c.out
k_means_c.py
No Python interpreter configured for the project
if rank == 0:
print(f"Time taken for data scattering: {data_loading_end - data_loading_start:.2f} seconds")
num_clusters = 5
iterations = 100
if rank == 0:
initial_indices = np.random.choice(all_points.shape[0], num_clusters, replace=False)
cluster_centers = all_points[initial_indices]
else:
cluster_centers = np.empty((num_clusters, 2))
kmeans_start = time.time()
for _ in range(iterations):
cluster_centers = comm.bcast(cluster_centers, root=0)
local_sum = np.zeros((num_clusters, 2))
local_count = np.zeros(num_clusters)
for point in local_points:
idx = np.argmin(np.linalg.norm(point - cluster_centers, axis=1))
local_sum[idx] += point
local_count[idx] += 1
global_sums = comm.reduce(local_sum, op=MPI.SUM, root=0)
global_counts = comm.reduce(local_count, op=MPI.SUM, root=0)


k_means_b.py
k_means_b.sh
kmeans_parallel_1n2c.png
k_parallel_1n2c.out
k_means_c.py
No Python interpreter configured for the project
Use Python 3.9
Configure Python interpreter
global_sums = comm.reduce(local_sum, op=MPI.SUM, root=0)
global_counts = comm.reduce(local_count, op=MPI.SUM, root=0)
if rank == 0:
cluster_centers = [global_sums[i] / global_counts[i] if global_counts[i] > 0 else cluster_centers[i] for i in range(num_
kmeans_end = time.time()
if rank == 0:
closest = np.array([np.argmin(np.linalg.norm(point - cluster_centers, axis=1)) for point in all_points])
fig, ax = plt.subplots(figsize=(10, 7))
palette = ['r', 'g', 'b', 'y', 'c', 'm']
for idx in range(num_clusters):
ax.scatter(all_points[closest == idx].T[0], all_points[closest == idx].T[1], c=palette[idx], alpha=0.5)
ax.scatter(cluster_centers[idx][0], cluster_centers[idx][1], color=palette[idx], marker='x')
fig.savefig('kmeans_parallel_1n2c.png')
print(f"Time taken for k-means clustering: {kmeans_end - kmeans_start:.2f} seconds")
print(f"Total Time taken: {kmeans_end - data_loading_start:.2f} seconds")

```
#!/bin/bash
#SBATCH --job-name=k_parallel_1n2c
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=2
#SBATCH --partition=partition
#SBATCH --output=k_parallel_1n2c.out
echo "Jobs are started ..."
srun --mpi=pmix_v4 python3 k_means_b.py
```

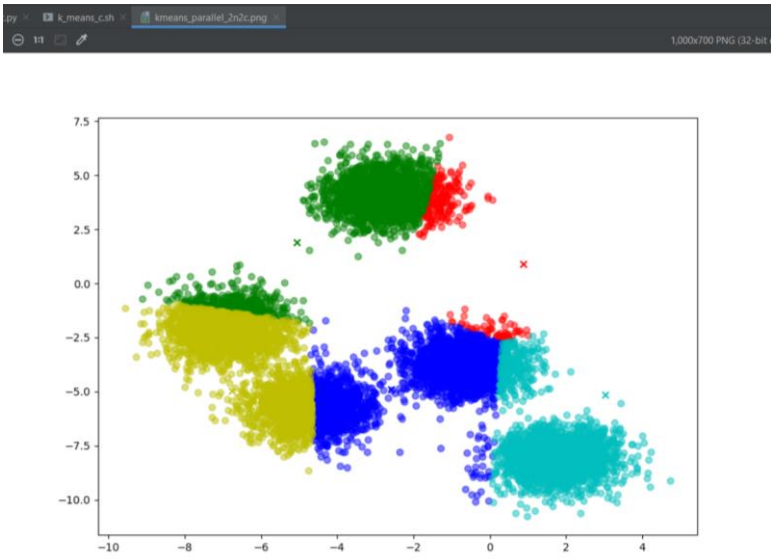


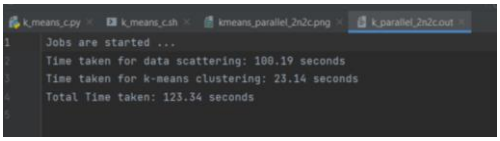

```
Jobs are started ...
Time taken for data scattering: 0.02 seconds
Time taken for k-means clustering: 44.09 seconds
Total Time taken: 44.12 seconds
```

قسمت ج)

کد دقیقا مانند مرحله قبل است ولی bash script را به گونه ای تغییر دادیم تا ۲ نود درگیر شوند :

```
#!/bin/bash
#SBATCH --job-name=k_parallel_2n2c
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --partition=partition
#SBATCH --output=k_parallel_2n2c.out
echo "Jobs are started ..."
srun --mpi=pmix_v4 python3 k_means_c.py
```





نتایج در قسمت ب و ج این سوال به شدت جذاب شد دیدیم وقتی دو نود شد زمان پخش داده به شدت رفت بالا ولی از طرفی رانتایم خود الگوریتم kmeans با توجه به زیاد شدن منابع محاسباتی کاهش داشت .