

محمد باربد 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:

```
e_sim_apy x e_sim_bpy x e_sim_cpy x k_means_apy x k_means_bpy
1 import random
2 import time
3 from decimal import Decimal, getcontext
4
5 NUM_ITERATIONS = 4_000_000
6
7 def monte_carlo_e_estimate():
8     counter = 0
9     getcontext().prec = 40
10    start_time = time.time()
11    for _ in range(NUM_ITERATIONS):
12        total = 0
13        while total <= 1:
14            total += random.uniform(0, 1)
15        counter += 1
16    elapsed_time = time.time() - start_time
17    e_estimate = Decimal(counter / NUM_ITERATIONS)
18    print(f"Euler's Number Estimate: {e_estimate}")
19    print(f"Elapsed Time: {elapsed_time} seconds")
20
21 if __name__ == '__main__':
22     monte_carlo_e_estimate()
```

```
1 #!/bin/bash
2 #SBATCH --job-name=e_serial_inic
3 #SBATCH --nodes=1
4 #SBATCH --ntasks-per-node=1
5 #SBATCH --partition=partition
6 #SBATCH --output=e_serial_inic.out
7 echo "Jobs are started ..."
```

```
1 Jobs are started ...
2 Euler's Number Estimate: 2.718238500000000091326742167427595694580078125
3 Elapsed Time: 13.5157639583479 seconds
4
```

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 :

```

1 from mpi4py import MPI
2 import random
3 import time
4 from decimal import Decimal, getcontext
5
6 comm = MPI.COMM_WORLD
7 my_rank = comm.Get_rank()
8 num_processes = comm.Get_size()
9
10 total_iterations = 4,000,000
11 local_counter = 0
12
13 getcontext().prec = 40
14
15 if my_rank == 0:
16     start = time.time()
17
18 for _ in range(int(total_iterations // num_processes)):
19     running_total = 0
20     while running_total <= 1:
21         running_total += random.uniform(0, 1)
22         local_counter += 1
23     total_count = comm.reduce(local_counter, root=0)
24
25 if my_rank == 0:
26     print(f"Euler: {Decimal(total_count / total_iterations)}")
27     print(f"time: {time.time() - start} (s)")
28

```

```

1 #!/bin/bash
2 #SBATCH --job-name=e_parallel_1n2c
3 #SBATCH --nodes=1
4 #SBATCH --ntasks-per-node=2
5 #SBATCH --partition=partition
6 #SBATCH --output=e_parallel_1n2c.out
7 echo "Jobs are started ..."
8 srun --mpi=pmix_v4 python3 e_sim_b.py
9

```

```

1 Jobs are started ...
2 Euler: 2.7181147500000000059372951000113971531391143798828125
3 time: 8.266072988510132 (s)
4

```

## قسمت ج)

کد این قسمت دقیقاً 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.

```

1 #!/bin/bash
2 #SBATCH --job-name=e_parallel_2n2c
3 #SBATCH --nodes=2
4 #SBATCH --ntasks-per-node=2
5 #SBATCH --partition=partition
6 #SBATCH --output=e_parallel_2n2c.out
7 echo "Jobs are started ..."
8 srun --mpi=pmix_v4 python3 e_sim_c.py
9

```

```

1 Jobs are started ...
2 Euler: 2.718545250000000000242152964347042143344879150390625
3 time: 6.322203636169434 (s)
4

```

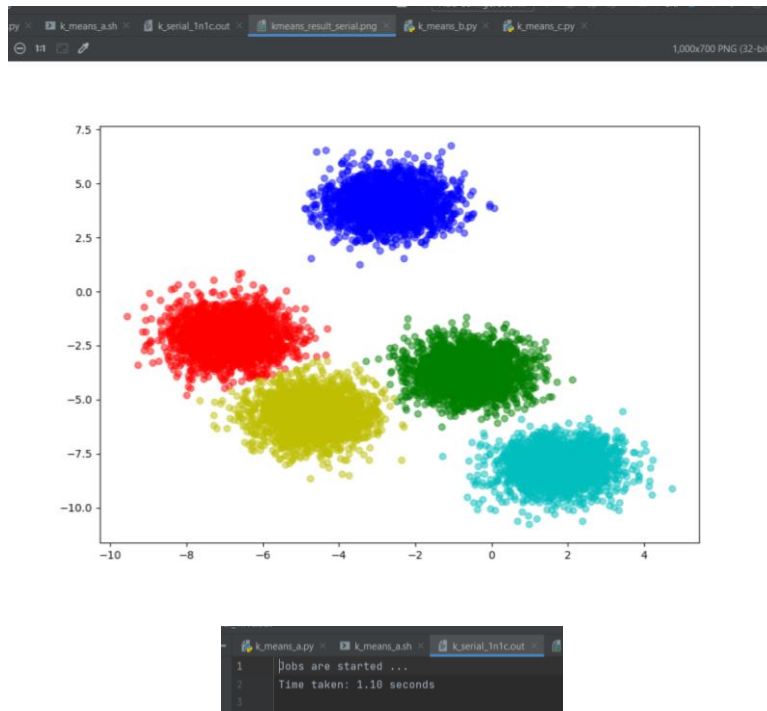
## سوال ۲:

### 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 لیل 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.

```
9 all_points = dataset.to_numpy()
10
11 num_clusters = 5
12 iterations = 100
13
14 initial_indices = np.random.choice(all_points.shape[0], num_clusters, replace=False)
15 cluster_centers = all_points[initial_indices]
16
17 for iteration in range(iterations):
18     distance = np.linalg.norm(all_points[:, np.newaxis] - cluster_centers, axis=2)
19     closest = np.argmin(distance, axis=1)
20     cluster_centers = [all_points[closest == cluster_idx].mean(axis=0) for cluster_idx in range(num_clusters)]
21
22 fig, ax = plt.subplots(figsize=(10, 7))
23 palette = ['r', 'g', 'b', 'y', 'c', 'm']
24
25 for idx in range(num_clusters):
26     ax.scatter(all_points[closest == idx].T[0], all_points[closest == idx].T[1], c=palette[idx], alpha=0.5)
27     ax.scatter(cluster_centers[idx][0], cluster_centers[idx][1], color=palette[idx], marker='x')
28
29 fig.savefig('kmeans_result_serial.png')
30
31 end_time = time.time()
32 elapsed_time = end_time - start_time
33 print(f"Time taken: {elapsed_time:.2f} seconds")
34
```

```
1 #!/bin/bash
2 #SBATCH --job-name=k_serial_init
3 #SBATCH --nodes=1
4 #SBATCH --ntasks-per-node=1
5 #SBATCH --partition=partition
6 #SBATCH --output=k_serial_init.out
7 echo "Jobs are started ..."
8 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 سائزی که هر هسته باید دریافت کند را محاسبه میکند سپس `all_points` را بین `local_points` ها تقسیم میکنیم، سپس مستر به صورت رندوم مراکز 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_in2c.png k_parallel

No Python interpreter configured for the project

1 from mpi4py import MPI
2 import pandas as pd
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import time
6
7 comm = MPI.COMM_WORLD
8 rank = comm.Get_rank()
9 size = comm.Get_size()
10
11 data_loading_start = time.time()
12
13 if rank == 0:
14     dataset = pd.read_csv('data.csv', header=None)
15     all_points = dataset.to_numpy()
16     partition_size = len(all_points) // size
17 else:
18     all_points = None
19     partition_size = None
20
21 partition_size = comm.bcast(partition_size, root=0)
22 local_points = np.empty((partition_size, 2))
23 comm.Scatter(all_points, local_points, root=0)
24
25 data_loading_end = time.time()
```

```
k_means_b.py k_means_b.sh kmeans_parallel_in2c.png k_parallel_in2cout k_means_c.py

No Python interpreter configured for the project Use Py

27 if rank == 0:
28     print(f"Time taken for data scattering: {data_loading_end - data_loading_start:.2f} seconds")
29
30 num_clusters = 5
31 iterations = 100
32
33 if rank == 0:
34     initial_indices = np.random.choice(all_points.shape[0], num_clusters, replace=False)
35     cluster_centers = all_points[initial_indices]
36 else:
37     cluster_centers = np.empty((num_clusters, 2))
38
39 kmeans_start = time.time()
40
41 for _ in range(iterations):
42
43     cluster_centers = comm.bcast(cluster_centers, root=0)
44     local_sum = np.zeros((num_clusters, 2))
45     local_count = np.zeros(num_clusters)
46
47     for point in local_points:
48         idx = np.argmin(np.linalg.norm(point - cluster_centers, axis=1))
49         local_sum[idx] += point
50         local_count[idx] += 1
51
52     global_sums = comm.reduce(local_sum, op=MPI.SUM, root=0)
53     global_counts = comm.reduce(local_count, op=MPI.SUM, root=0)
```

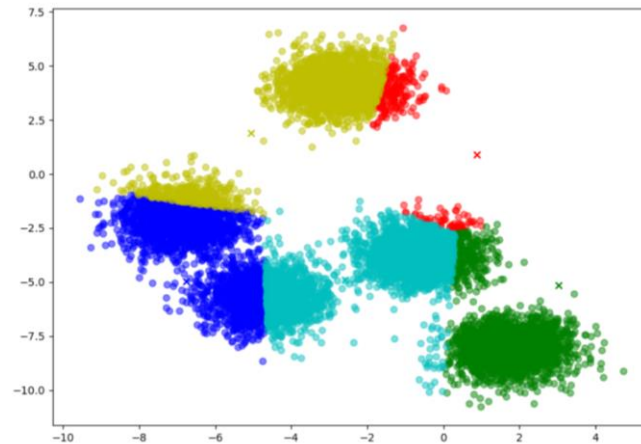
```
k_means_b.py k_means_b.sh kmeans_parallel_in2c.png k_parallel_in2cout k_means_c.py

No Python interpreter configured for the project Use Python 3.9 Configure Python interpreter

52 global_sums = comm.reduce(local_sum, op=MPI.SUM, root=0)
53 global_counts = comm.reduce(local_count, op=MPI.SUM, root=0)
54
55 if rank == 0:
56     cluster_centers = [global_sums[i] / global_counts[i] if global_counts[i] > 0 else cluster_centers[i] for i in range(num_
57
58 kmeans_end = time.time()
59
60 if rank == 0:
61     closest = np.array([np.argmin(np.linalg.norm(point - cluster_centers, axis=1)) for point in all_points])
62
63     fig, ax = plt.subplots(figsize=(10, 7))
64     palette = ['r', 'g', 'b', 'y', 'c', 'm']
65
66     for idx in range(num_clusters):
67         ax.scatter(all_points[closest == idx].T[0], all_points[closest == idx].T[1], c=palette[idx], alpha=0.5)
68         ax.scatter(cluster_centers[idx][0], cluster_centers[idx][1], color=palette[idx], marker='x')
69
70     fig.savefig('kmeans_parallel_in2c.png')
71
72 print(f"Time taken for k-means clustering: {kmeans_end - kmeans_start:.2f} seconds")
73 print(f"Total Time taken: {kmeans_end - data_loading_start:.2f} seconds")
74
```

```
k_means_bash x kmeans_parallel_1n2c.png x k_parallel_1n2c.out x k_means_cpy x
1 Would you like to install shellcheck to verify your shell scripts?
2 #!/bin/bash
3 #SBATCH --job-name=k_parallel_1n2c
4 #SBATCH --nodes=1
5 #SBATCH --tasks-per-node=2
6 #SBATCH --partition=partition
7 #SBATCH --output=k_parallel_1n2c.out
8 echo "Jobs are started ..."
9 srun --mpi=pmix_v4 python3 k_means_b.py
```

```
bash x kmeans_parallel_1n2c.png x k_parallel_1n2c.out x k_means_cpy x
14 1.000x700 PNG (32-bit)
```

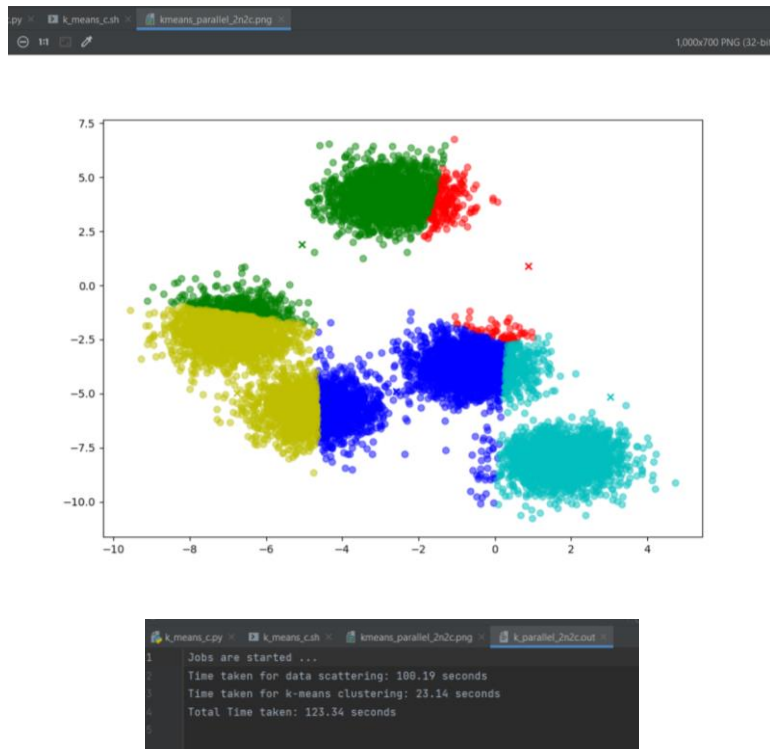


```
k_means_bash x k_parallel_1n2c.out x k_means_cpy x
1 Jobs are started ...
2 Time taken for data scattering: 0.02 seconds
3 Time taken for k-means clustering: 44.09 seconds
4 Total Time taken: 44.12 seconds
5
```

## قسمت ج)

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

```
k_means_cpy x k_means_cah
1 Would you like to install shellcheck to verify your shell scripts?
2 #!/bin/bash
3 #SBATCH --job-name=k_parallel_2n2c
4 #SBATCH --nodes=2
5 #SBATCH --tasks-per-node=2
6 #SBATCH --partition=partition
7 #SBATCH --output=k_parallel_2n2c.out
8 echo "Jobs are started ..."
9 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 .