HPC OPENMP TUTORIAL 7 REPORT

CS22B2015 – HARSHITH B

The parallel code for Mean Shift Clustering Algorithm is provided in main.c file. The key section of the code performs the following steps:

1. Threads are allocated in multiples of 2 (1, 2, 4, 8, 16, 32, 64, 128) within a loop, and the sum is calculated using the OpenMP pragma reduction. The serial code runs by default in 1 thread and therefore OpenMP is not being used.
2. Dataset containing approx 10^5 data points are being input to implement the clustering.
3. The threads versus time data is recorded into a text file. This data is then processed, and the respective results are plotted and analyzed using results.py.

1. Parallel Code Segment

```
#pragma omp parallel for num_threads(thread_count)
for (int i = 0; i < numPoints; i++) {
    newCentroids[i].x = 0.0;
    newCentroids[i].y = 0.0;
    weightsSum[i] = 0.0;
}

#pragma omp parallel for num_threads(thread_count)
for (int i = 0; i < numPoints; i++) {
    for (int j = 0; j < numPoints; j++) {
        double distance = sqrt(pow(centroids[i].x - data[j].x, 2) + pow(centroids[i].y - data[j].y, 2));
        double weight = gaussianKernel(distance, bandwidth);

        newCentroids[i].x += data[j].x * weight;
        newCentroids[i].y += data[j].y * weight;
        weightsSum[i] += weight;
    }

    if (weightsSum[i] > 0) {
        newCentroids[i].x /= weightsSum[i];
        newCentroids[i].y /= weightsSum[i];
    }
}

double maxShift = 0.0;
#pragma omp parallel for reduction(max:maxShift) num_threads(thread_count)
for (int i = 0; i < numPoints; i++) {
    double shift = sqrt(pow(newCentroids[i].x - centroids[i].x, 2) + pow(newCentroids[i].y - centroids[i].y, 2));
    if (shift > maxShift) {
        maxShift = shift;
    }
}

free(centroids);
```

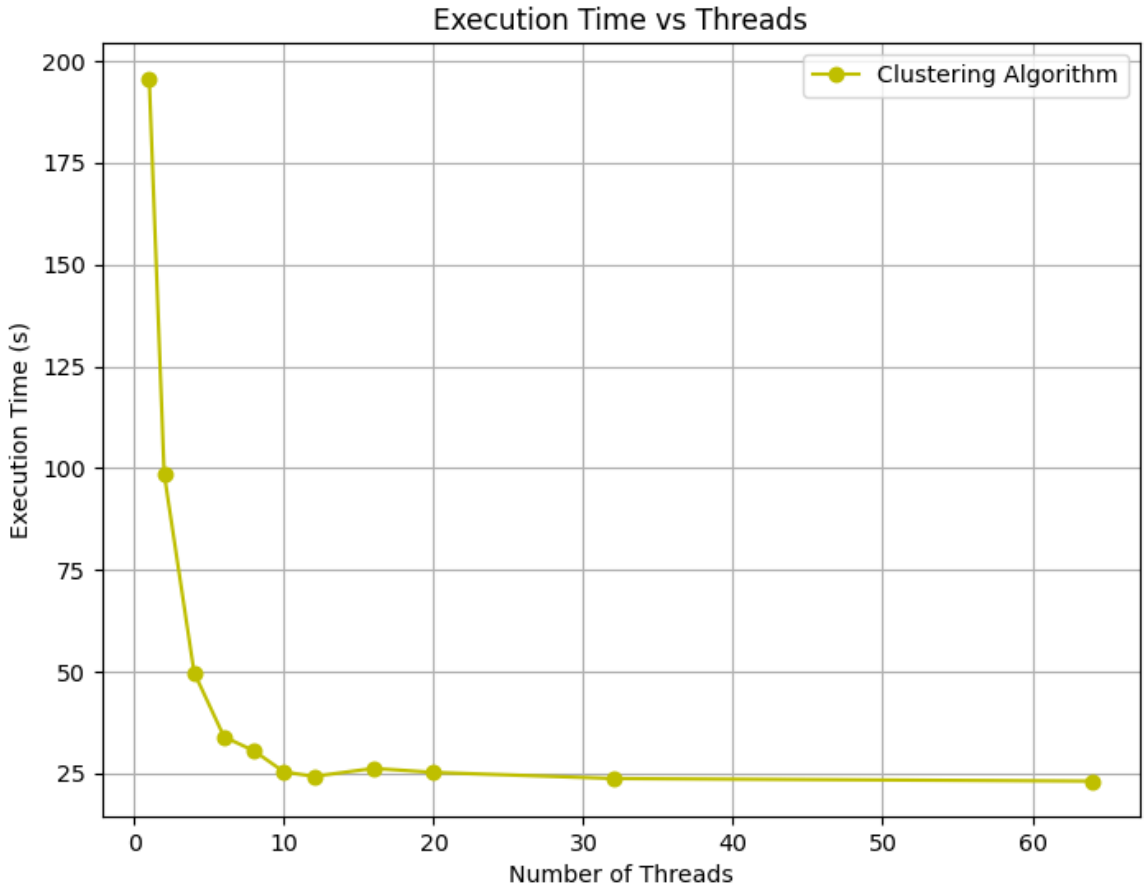
2. Terminal Output (Output of 10 threads)

```
Threads: 10, Time: 25.399304 seconds, Number of Centroids: 41
Centroid 0: (93.814496, 6.296693)
Centroid 1: (93.758127, 6.398689)
Centroid 2: (93.895221, 6.150731)
Centroid 3: (94.028312, 5.910349)
Centroid 4: (94.160556, 5.671831)
Centroid 5: (94.111891, 5.759566)
Centroid 6: (93.956708, 6.039635)
Centroid 7: (93.709455, 6.486807)
Centroid 8: (94.272122, 5.470871)
Centroid 9: (94.345598, 5.338659)
Centroid 10: (94.427393, 5.191615)
Centroid 11: (94.211857, 5.579393)
Centroid 12: (68.792432, 92.198536)
Centroid 13: (24.761575, 38.846405)
Centroid 14: (72.205184, 27.359281)
Centroid 15: (0.000000, 0.000000)
Centroid 16: (0.000000, 0.000000)
Centroid 17: (0.000000, 0.000000)
Centroid 18: (0.000000, 0.000000)
Centroid 19: (0.000000, 0.000000)
Centroid 20: (-nan, 0.000000)
Centroid 21: (0.000000, 0.000000)
Centroid 22: (0.000000, 0.000000)
Centroid 23: (0.000000, 0.000000)
Centroid 24: (0.000000, 0.000000)
Centroid 25: (0.000000, 0.000000)
Centroid 26: (0.000000, 0.000000)
Centroid 27: (0.000000, 0.000000)
Centroid 28: (0.000000, 0.000000)
Centroid 29: (0.000000, 0.000000)
Centroid 30: (0.000000, 0.000000)
Centroid 31: (0.000000, 0.000000)
Centroid 32: (0.000000, 0.000000)
Centroid 33: (0.000000, 0.000000)
Centroid 34: (-nan, 0.000000)
Centroid 35: (0.000000, 0.000000)
Centroid 36: (0.000000, 0.000000)
Centroid 37: (0.000000, 0.000000)
Centroid 38: (0.000000, 0.000000)
Centroid 39: (0.000000, 0.000000)
Centroid 40: (0.000000, 0.000000)
```

3. Thread vs Time Plots and Observations

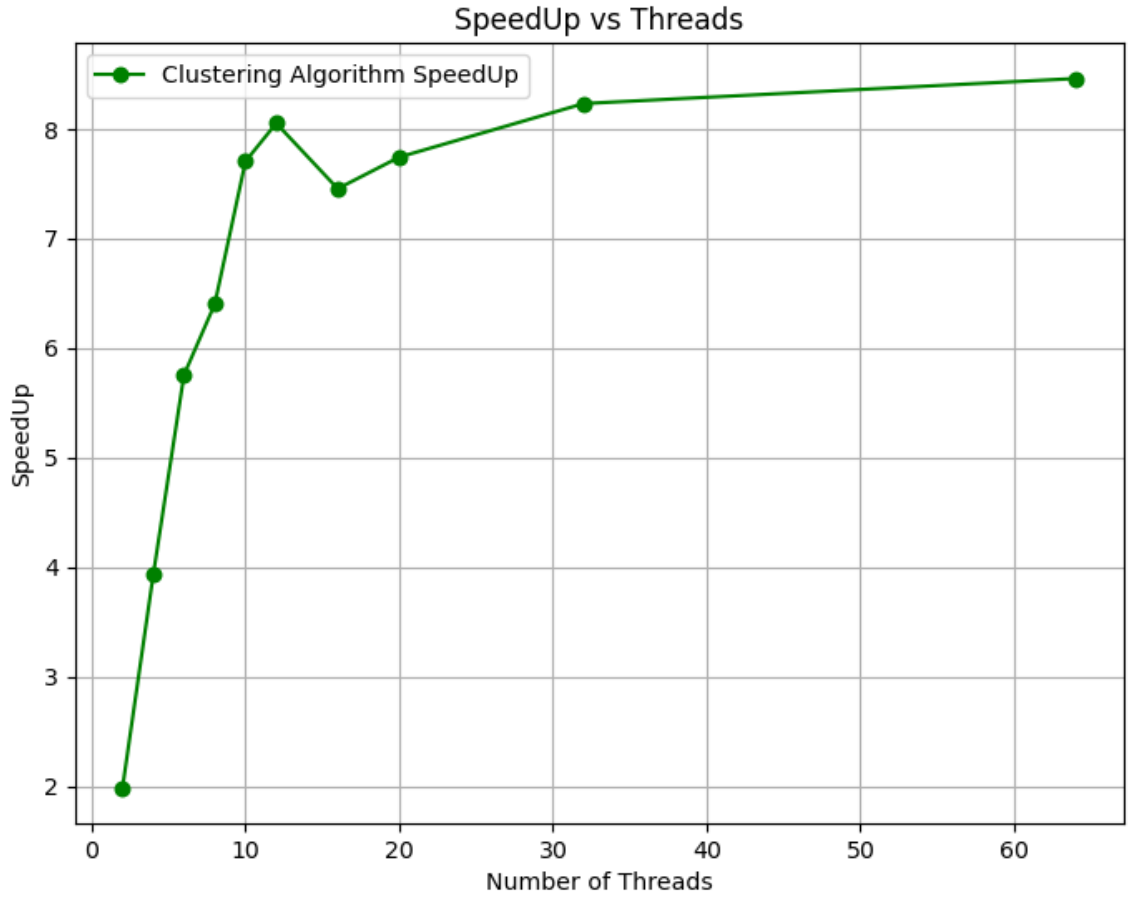
Observations :

- **Continuous Improvement with some deviations: We can observe that there are some deviations with mostly reducing time vs threads with peak performance always by increasing threads.**
- **Optimal Thread Count:** The best performance is observed at a 64 threads.

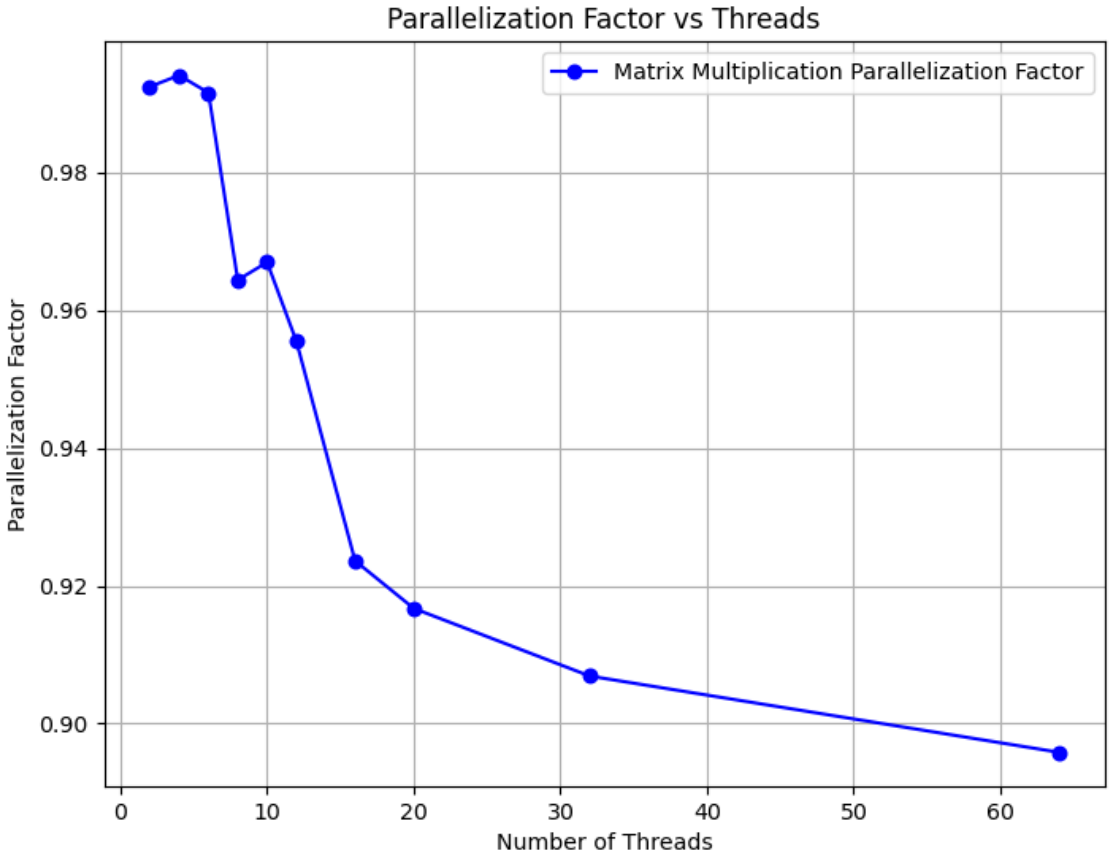


---

4. SpeedUp vs Processors (SpeedUp == T(1) / T(n))



5. Parallelization Fraction and Inference



Parallelisation Factor estimation : Range is 0.89 - 0.99 across different threads with 0.89 being 64 threads and 0.99 being at approx 4 threads.