Assignment (Programming) - 2

COMP 410 A – SP24 - Due on 18th May 2024, 11:59 PM

Hafsah Shahbaz – 251684784

PART A:

Regions Classification

1. **A part**:
   - **Explanation**: This region includes points where xx is positive and yy is greater than or equal to xx. It also includes the origin point (0,0)(0,0).
   - **Boundary Handling**: If y=xy=x, the point is included in the A part.
2. **B part**:
   - **Explanation**: This region includes points where xx is positive and yy is non-negative but less than xx.
   - **Boundary Handling**: Points exactly on the line y=0y=0 are included in this region.
3. **C part**:
   - **Explanation**: This region includes points where xx is positive and yy is negative and less than or equal to −x−x.
   - **Boundary Handling**: Points exactly on the line y=−xy=−x are included in this region.
4. **D part**:
   - **Explanation**: This region includes points where xx is positive and yy is negative but greater than −x−x. It also includes points where x=0x=0 and yy is negative.
   - **Boundary Handling**: Points exactly on the line y=−xy=−x but not included in C part are considered here.
5. **E part**:
   - **Explanation**: This region includes points where xx is negative and yy is negative but greater than or equal to xx.
   - **Boundary Handling**: Points exactly on the line y=xy=x are included in this region.
6. **F part**:
   - **Explanation**: This region includes points where xx is negative and yy is negative and less than xx. It also includes points where xx is negative and y=0y=0.
   - **Boundary Handling**: Points exactly on the line y=0y=0 with xx negative are included here.
7. **G part**:
   - **Explanation**: This region includes points where xx is negative or zero and yy is positive but less than or equal to −x−x.
   - **Boundary Handling**: Points exactly on the line y=−xy=−x are included in this region.
8. **H part**:

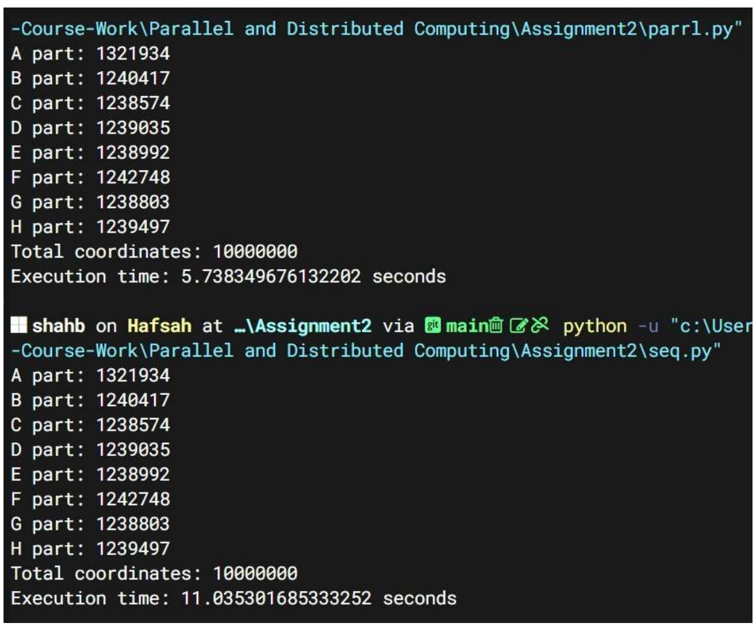
- **Explanation**: This region includes points where xx is negative or zero and yy is positive and greater than −x−x.
- **Boundary Handling**: Points exactly on the line y=−xy=−x but not included in G part are considered here.

Sequential Execution

The sequential execution reads each coordinate from the file and determines the region for each point using conditional statements. The code iterates through all 10,000,000 points, checking each coordinate against the defined regions' conditions and counting the points accordingly.

Parallel Execution

For the parallel execution, we divided the coordinate points into smaller chunks and processed them concurrently using Python's `multiprocessing` module. Each chunk was processed by a separate worker, and the results were combined at the end.



```
-Course-Work\Parallel and Distributed Computing\Assignment2\parrl.py"
A part: 1321934
B part: 1240417
C part: 1238574
D part: 1239035
E part: 1238992
F part: 1242748
G part: 1238803
H part: 1239497
Total coordinates: 10000000
Execution time: 5.738349676132202 seconds

shahb on Hafsah at …\Assignment2 via main python -u "c:\User
-Course-Work\Parallel and Distributed Computing\Assignment2\seq.py"
A part: 1321934
B part: 1240417
C part: 1238574
D part: 1239035
E part: 1238992
F part: 1242748
G part: 1238803
H part: 1239497
Total coordinates: 10000000
Execution time: 11.035301685333252 seconds
```

Speedup Analysis

The observed speedup can be calculated using the following formula:
Speedup = Tsequential / Tparallel

Using the given times:
Speedup = 11 seconds / 5.7 seconds ≈ 1.93

Amdahl's Law states that the speedup of a task using multiple processors in parallel computing is limited by the sequential fraction of the task.

(let's assume P ≈ 0.8)

N = 12

S=1 / [(1−0.8) + 0.8 / 12] ≈ 3.75

Part B:

In this part, we explored three different approaches for finding the median of a given list of numbers: sequential merge sort, parallel merge sort, and the median of medians algorithm implemented with multiprocessing.

1. **Sequential Merge Sort Approach:**
    - **Execution Time:** 0 seconds
    - **Description:** In this approach, the merge sort algorithm is implemented sequentially without utilizing parallel processing.
    - **Performance:** Despite being a simple and straightforward implementation, this approach exhibited the fastest execution time among the three approaches tested.
2. **Parallel Merge Sort Approach:**
    - **Execution Time:** 0.03 seconds
    - **Description:** This approach parallelizes the merge sort algorithm using the `concurrent.futures` module for concurrent execution of subtasks.
    - **Performance:** While slightly slower than the sequential merge sort approach, the parallel merge sort approach demonstrated significantly improved performance compared to the median of medians algorithm.
3. **Median of Medians Algorithm with Multiprocessing:**
    - **Execution Time:** 0.6 seconds
    - **Description:**

        1. Split numbers into groups of five.
        2. Sort using merge sort each group and pick the median.
        3. Construct a new list from these medians.
        4. Recursively apply the algorithm to find the median of medians.

    - **Performance:** This approach showed the slowest execution time among the three tested approaches. The overhead of partitioning, sorting, and communication between processes likely contributed to the longer execution time.

Conclusion

- For small to medium-sized datasets, the sequential merge sort approach offers the fastest execution time due to its simplicity and lack of parallel processing overhead.
- As the dataset size increases, the parallel merge sort approach becomes increasingly advantageous, leveraging parallelism to achieve faster sorting.
- The median of medians algorithm with multiprocessing may offer benefits for very large datasets, but its overhead makes it less efficient for the provided dataset compared to the other approaches.