Parallel Data Processing Project Using MPI

Project Logic Explanation:
The idea behind this project is to leverage the capabilities of MPI (Message Passing Interface) to perform parallel data processing tasks efficiently. MPI enables multiple processes to work together by distributing data and computation across different processors or cores. This approach significantly reduces the time required to execute data-intensive tasks, especially when compared to sequential execution.

Each task in the project showcases a practical application of parallel computing. For instance:

- **Odd-Even Transposition Sort** is used to demonstrate how a sorting algorithm can be parallelized by dividing a list among processes.
- **Word Counting and Duplicate Removal** uses text partitioning to enable multiple processes to analyze a file simultaneously.
- **Image Processing** tasks like grayscale conversion and blurring are split by pixel rows, allowing for concurrent transformation.
- **Linear Regression Training** distributes the dataset so each process contributes to the final model training.
- **Keyword Search in Large Texts** divides the text file to search for a given keyword in parallel, improving search performance.
- **Statistical Analysis** splits CSV columns or rows for processes to independently calculate statistical measures.
- **Matrix Multiplication** is implemented by distributing rows and columns of the matrices for simultaneous computation.

Each of these examples highlights how dividing a task into smaller chunks and processing them in parallel can lead to faster and more scalable software solutions.

Challenges Faced During Development:
While the project was conceptually straightforward, integrating it into a web-based interface presented several issues:

1. **Streamlit Integration:** Streamlit, while excellent for building user interfaces, is not designed to handle MPI-style process spawning natively. MPI requires multiple processes to be launched at once, which conflicts with how Streamlit manages

application state and user sessions. This made it difficult to trigger mpiexec commands from within the web app.

2. **Command Prompt Execution:** Executing MPI programs via mpiexec from a Python-based interface introduced challenges in process control and output retrieval. Running parallel processes reliably from the command line and then capturing and displaying results within a web app was not as seamless as expected.
3. **Data Handling Between Processes:** Ensuring that file uploads, user inputs, and output collection were correctly managed across distributed processes involved significant debugging and synchronization effort.

Despite these issues, the project successfully demonstrated how core data processing tasks can benefit from distributed computing using MPI in Python. Although the full web-based interface integration faced limitations, the backend logic and MPI implementations are functional and can be run independently via command-line execution.

Conclusion:
This project was a valuable learning experience in understanding and applying parallel computing principles to real-world tasks. It also highlighted the technical boundaries of combining distributed systems with web development frameworks like Streamlit. In the future, solutions like job schedulers or dedicated backend servers may help bridge this gap more effectively.

Names: Mariam Taher IDS : 320220201

Noor Abdelsalam 320220191

Mariam Mostafa 320220187

Section 3