1. **Main Entry Point:**

-The program first prompts the user to choose the implementation type: Distributed (MPI) or **T**hreaded/Task-Based.

-Based on user input, it runs either RunMPIImplementation or RunThreadedImplementation.

1. **Matrix Class:**

The Matrix class represents the current state of the puzzle, including the positions of the tiles and the empty space 0. It also tracks the number of steps taken to reach the current state.

This class includes methods to:

-Load a matrix from a file (FromFile).

-Calculate the Manhattan Distance heuristic (helper function) for easier puzzle solving.

-Generate valid moves by sliding a tile into the empty space.

-Check if the puzzle is solvable using the **inversion count method**.

1. **Distributed (MPI) Implementation (RunMPIImplementation):**

-The MPI version of the solution divides the work across multiple processes (one master and several worker processes).

-The master process (rank 0) loads the puzzle, checks if it’s solvable, and then distributes puzzle configurations to worker processes.

-Each worker process searches for a solution, and results are sent back to the master.

-The search is done using the iterative deepening approach, where each process explores the state space up to a certain depth (bounded by Manhattan Distance).

-Communication: Uses SendObject and ReceiveObject to send and receive data (states and results) between master and worker processes.

-The worker processes return the best solution found, or a bound value if no solution is found within their assigned depth.

1. **Threaded Implementation (RunThreadedImplementation):**

-This version uses multi-threading with Task Parallelism to explore puzzle configurations concurrently.

-Worker threads fetch states from a shared queue, explore possible moves, and add new states back to the queue.

-The search process is terminated when a solution is found or all states have been explored. It uses Manhattan Distance as the heuristic for pruning.

* + **Task Queue**: A ConcurrentQueue<Matrix> is used to store the states to be explored by threads.

1. **Helper Classes & Methods Pair and SendObject/ReceiveObject:**

-Pair<T1, T2>: A helper class to hold a pair of values, used to store results like the search depth and matrix state.

-SendObject and ReceiveObject: These methods serialize and deserialize objects to and from JSON format for communication between processes or threads.

**Main**

-Displays a menu for the user to choose between the **Distributed (MPI)** or **Threaded/Task-Based** implementations.

-Based on the choice, it either calls RunMPIImplementation or RunThreadedImplementation.

**RunMPIImplementation**

-Initializes the MPI environment.

-The **master process (rank 0)**:

-Loads the puzzle from the file mat.in, checks solvability, and proceeds with the search by calling SearchMaster.

**-Worker processes (rank > 0)**:

-Each worker listens for puzzle configurations, processes them using a recursive depth-limited search (SearchWorker), and returns the results to the master.

**RunThreadedImplementation**

-Initializes the root puzzle state and checks solvability.

-A task queue (ConcurrentQueue<Matrix>) is used to store states, which are dequeued by worker threads (SolveUsingThreads method).

-The search process is parallelized by spawning multiple worker threads, each of which explores the puzzle state space and looks for a solution.

**SearchMaster and SearchWorker (MPI-based Search):**

**-Master Process**:

-Generates initial puzzle states and assigns them to workers.

-Collects results from workers, compares found solutions, and updates the search depth (bound).

**-Worker Process**:

-Receives a puzzle state and search bound from the master.

-Searches for a solution recursively, and sends results back to the master.

**Search (Depth-First Search with Bounding):**

-Performs a recursive depth-first search (DFS) for a solution.

-Prunes the search by using the Manhattan Distance as a heuristic to limit the search depth (estimation > bound).

**Matrix Class (Puzzle State Representation):**

-Represents a 4x4 puzzle state (with tiles, free space position, number of steps, etc.).

-Provides methods to:

**-Generate moves**: Generates possible states resulting from sliding tiles.

**-Manhattan Distance**: Calculates the Manhattan Distance heuristic for a given puzzle state, used to determine how far the current state is from the goal state.

**-IsSolvable**: Checks whether the puzzle configuration is solvable using the inversion count method.

**Serialization/Deserialization (SendObject and ReceiveObject):**

-Used for sending and receiving objects over MPI communication. The objects are serialized into JSON format before being sent, and deserialized on the receiving side.