**Parallel Jacobi Iteration Solver for 2D Poisson Equation**

1. **Problem Description**

The code implements a parallel solution to the 2D Poisson equation using the Jacobi iterative method with MPI (Message Passing Interface) for distributed memory parallelization. The specific problem involves solving the equation ∇²u = f on a 2D square domain with prescribed boundary conditions.

**2. Numerical Method**

**2.1 Discretization**

* The domain is discretized into a uniform grid of size N × N
* Grid spacing h = 1 / (N - 1)
* Jacobi iteration used for solving the discrete Poisson equation
* Exact solution u(x,y) = xy used for boundary conditions and error checking

**2.2 Parallel Decomposition**

* Domain is divided among processes in a 2D grid layout
* Number of processes must be a perfect square
* Local subdomains distributed using a square root of total processes
* Each process manages a local grid with ghost points for neighboring exchanges

**3. Key Implementation Components**

**3.1 Initialization**

* Command-line argument for grid size N
* Calculates local grid dimensions for each process
* Handles uneven distribution of rows/columns for non-perfect divisions
* Initializes local grids with exact boundary conditions

**3.2 Parallel Communication**

* Uses MPI\_Sendrecv for ghost point exchanges
* Handles communication for top, bottom, left, and right neighbors
* Custom MPI datatype created for column exchanges
* Careful handling of edge cases and boundary processes

**3.3 Jacobi Iteration**

* Performs in-place updates on local grid
* Incorporates ghost point values from neighboring processes
* Calculates local maximum difference for convergence check

**3.4 Convergence Criteria**

* Iterates until:
  1. Maximum difference between iterations is below tolerance (1e-6)
  2. Maximum number of iterations (10,000) is reached

**4. Parallel Algorithm Steps**

1. Initialize MPI and determine process grid
2. Distribute domain among processes
3. Set up boundary conditions
4. Prepare communication buffers
5. Main iteration loop:
   * Prepare ghost point send buffers
   * Exchange ghost points with neighbors
   * Perform local Jacobi iteration
   * Reduce maximum difference across all processes
6. Calculate and report global error
7. Clean up and finalize

**6. Compilation and Execution**

mpicc CMPE478-fall-2024-hw2-part-a.c -o CMPE478-fall-2024-hw2-part-a

mpirun -np <number\_of\_processes> ./ CMPE478-fall-2024-hw2-part-a <grid\_size>

**7. Complexity Analysis**

* Time Complexity: O(N²/P \* iterations), where N is grid size and P is number of processes
* Space Complexity: O(N²/P) per process

**8. Conclusion**

The implemented parallel Poisson solver demonstrates an effective approach to distributed numerical computing using MPI. It provides a scalable solution for solving 2D partial differential equations across multiple computational nodes.