```
#include <mpi.h>      /* Jacobi iteration using MPI */
#include <stdio.h>
#define MAXGRID 258   /* maximum grid, with boundaries */
#define COORDINATOR 0  /* rank of Coordinator */
#define TAG 0          /* not used */

static void Coordinator(int,int,int);
static void Worker(int,int,int,int,int);

int main(int argc, char *argv[]) {
  int myid, numIters;
  int numWorkers, gridSize; /* assume gridSize is     */
  int stripSize;            /* multiple of numWorkers */

  MPI_Init(&argc, &argv);           /* initialize MPI */
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);
  MPI_Comm_size(MPI_COMM_WORLD, &numWorkers);
  numWorkers--;  /* one coordinator, rest are workers */
  read gridSize and numIters; compute stripSize;
  if (myid == COORDINATOR)
    Coordinator(numWorkers, stripSize, gridSize);
  else
    Worker(myid,numWorkers,stripSize,gridSize,numIters);
  MPI_Finalize();  /* clean up MPI */
}

static void Coordinator(int numWorkers,
                     int stripSize, int gridSize) {
  double grid[MAXGRID][MAXGRID];
  double mydiff = 0.0, maxdiff = 0.0;
  int i, worker, startrow, endrow;
  MPI_Status status;

  /* get final grid values from Workers */
  for (worker = 1; worker <= numWorkers; worker++) {
    startrow = (worker-1)*stripSize + 1;
    endrow = startrow + stripSize - 1;
    for (i = startrow; i <= endrow; i++)
      MPI_Recv(&grid[i][1], gridSize, MPI_DOUBLE, worker,
            TAG, MPI_COMM_WORLD, &status);
  }
  /* reduce differences from Workers */
  MPI_Reduce(&mydiff, &maxdiff, 1, MPI_DOUBLE,
          MPI_MAX, COORDINATOR, MPI_COMM_WORLD);
  print results;
}
```

```
static void Worker(int myid, int numWorkers,
         int stripSize, int gridSize, int numIters) {
  double grid [2][MAXGRID][MAXGRID];
  double mydiff, maxdiff;
  int i, j, iters;
  int current = 0, next = 1; /* current and next grids */
  int left, right;           /* neighboring workers    */
  MPI_Status status;

  initialize my grids;  determine left and right neighbors;

  for (iters = 1; iters <= numIters; iters++) {

    /* exchange my boundaries with my neighbors */
    if (right != 0)  MPI_Send(&grid[next][stripSize][1],
       gridSize, MPI_DOUBLE, right, TAG, MPI_COMM_WORLD);
    if (left != 0)  MPI_Send(&grid[next][1][1], gridSize,
                  MPI_DOUBLE, left, TAG, MPI_COMM_WORLD);
    if (left != 0)  MPI_Recv(&grid[next][0][1], gridSize,
       MPI_DOUBLE, left, TAG, MPI_COMM_WORLD, &status);
    if (right != 0) MPI_Recv(&grid[next][stripSize+1][1],
          gridSize, MPI_DOUBLE, right, TAG,
          MPI_COMM_WORLD, &status);

    /* update my points */
    for (i = 1; i <= stripSize; i++)
      for (j = 1; j <= gridSize; j++)
        grid[next][i][j] = (grid[current][i-1][j] +
          grid[current][i+1][j] + grid[current][i][j-1] +
          grid[current][i][j+1]) * 0.25;
    current = next;  next = 1-next;  /* swap grids */
  }

  /* send my rows of final grid to the coordinator */
  for (i = 1; i <= stripSize; i++) {
    MPI_Send(&grid[current][i][1], gridSize, MPI_DOUBLE,
             COORDINATOR, TAG, MPI_COMM_WORLD);
  }
  compute mydiff;
  /* reduce mydiff with Coordinator */
  MPI_Reduce(&mydiff, &maxdiff, 1, MPI_DOUBLE,
             MPI_MAX, COORDINATOR, MPI_COMM_WORLD);
}
```

Figure 12.2 Jacobi iteration using MPI.