



Programming Project 5: Parallelizing Jacobi Iteration

Due: 4:00 PM, Wednesday, October 21, 2026

In this programming project, parallelize the Jacobi iteration program using the SPMD parallel computational model, using message passing with MPI for communication.

You may use your own serial implementation as a starting point, or use my sample solution (available on request).

You may work alone or with a partner or two on this programming project. AI assistance is permitted, but must be documented and intermediate steps in the process must be committed to your repository. You are responsible for the code you submit and will be required to demonstrate your program and explain key aspects of your code.

Learning goals:

1. To parallelize a scientific computation using MPI for communication.

Getting Set Up

In Canvas, you will find a link to the starter GitHub repository for this programming project, and you should fork that repository into your own GitHub account and make sure it is shared with your instructor. Only one member of the group should fork the repository on GitHub, then others should request a link to be granted write access.

All GitHub repositories must be created with all group members having write access and all group member names specified in the `README.md` file by 4:00 PM, Friday, October 14, 2026. This applies to those who choose to work alone as well!

Parallelization of Jacobi Iteration

The parallelization of the Jacobi iteration program will closely follow that in the MPI version of Conway's Game of Life. Distribute the computational domain among the processors by groups of adjacent rows. As with the Game of Life simulation example, you may assume the number of processes evenly divides the number of grid rows.

Your program's functionality should remain the same, and your solution output should be identical to the serial version.

The printing of solutions to the terminal and to a file are naturally sequential operations. In the serial version, this was not a concern. But here, you will have a distributed data structure. There are two ways you might synchronize that output:

- Gather all solution data to one process, likely the rank 0 process, and that process is responsible for all output.
- Have the processes take turns printing/writing the solution data to the terminal/file.

For full credit, you should use the second approach above, as this will require much less message passing and the distributed nature of the data structure will be preserved.

Question 1: Find a problem size and a number of iterations that runs for about 10 minutes using a single process on noreaster. Run that same problem instance on 2, 4, 8, 16, and 32 processes and time each. Report your results in tabular form, including speedups and efficiencies. (10 points)

For the questions below, assume a problem size of 512x512 grid cells.

Question 2: How many rows would be allocated in memory (a) on each process and (b) across all processes if you run on 1, 16, 64, and 256 processes? (4 points)

Question 3: How many messages are sent to pass a complete row from one process to another on each iteration when running on 16, 64, and 256 processes? (3 points)

Extra Credit Opportunities

For 5 bonus points, make your simulation work when the number of processes does not evenly divide the number of rows.

Submission

Commit and push!

Grading

This assignment will be graded out of 85 points.

Feature	Value	Score
Correct Makefile	1	
Command-line parameters	2	
Distributed grid allocation	10	
Parallel Jacobi iteration	20	
Stops based on iteration limit	2	
Stops based on error tolerance	4	
Print simulation stats	4	
Synchronized terminal solution output	5	
Synchronized file solution output	5	
Documentation	10	
Code Style	5	
Question 1	10	
Question 2	4	
Question 3	3	
(Bonus) Works for any size grid and numprocs	0	
Total	85	