

## Project #2 – MPI

### Overview

For this project you will analyze the performance characteristics of different decomposition schemes. In particular, you will implement a program using MPI that overlays a “space” with points and that computes information for each point based on its neighbors. This is often referred to as a Jacobi iteration.

### The Technique

Your program will repeatedly compute new values for each point based on the values of its neighbors. As such, each point will store two values: the “old” and the “new” value.

Execution proceeds in phases (or iterations). During the current iteration the “new” value at a point is the *average* of the “old” values at its four neighbors (diagonal neighbors are not considered; boundary cases are discussed below). Every point must complete the current phase before any point moves to the next phase. (Note that one may do more interesting computations at each point, but taking the average of the neighbors is a good place to start.)

In a typical approach, these iterations would continue until the values at all points converge (within an epsilon). For this assignment, we will limit the number of iterations to a specific value to control execution time.

Points on the boundary of this “space” will never change value. They serve only to respond to their neighbors.

### Parameters

Your program must define a square “space” of size  $N \times N$  (this defines the number of points). To simplify the decompositions,  $N$  will be a power of two.

Your program will take a value specifying the number of iterations to perform.

To initialize the values in the “space”, your program will take a value  $M$  as an argument. Select  $M$  points (boundary points are valid) at random and initialize each to a random value. (Note that this is strictly to make the result more interesting. It will not affect the work done since the computation will be bounded by the specified number of iterations.)

### Decompositions

You will need to compare three different decompositions.

- Serial. Use a single process to compute the iterations. This is a baseline measure of the computation.
- Blocks. Divide the “space” into blocks of points. Each block is assigned to a single process (this process computes the iterations for its set of points). If taken to the extreme, this decomposition will result in a process per point. Do not do this as it will require more processes than you will be able to spawn (except on the smallest test cases).
- Rows. Divide the “space” into rows of points. Each set of rows is assigned to a single process (i.e., a process can manage multiple rows).

The communication requirements for these three decompositions are different. You may choose to create a separate program for each.

### Experimentation

Compare the different decompositions on various test sizes. In particular, vary the dimensions of the “space” and the number of iterations; the number of initialized points should have no effect on the performance. Be sure to vary the numbers of blocks and rows.

Once you have a feel for the relative performance, modify the computation at each point. Instead of a simple average, increase the computational complexity (the time taken to compute the “new” value). How does this change the results of the previous experiments?

### Report

Write and submit a report that details the experiments performed and their results.

### Handin

Submit, using `handin`, your source code and paper to `458_hw2` on vgon.