CSE 6220/CX 4220
Introduction to High Performance Computing
Spring 2023
Programming Assignment 3
Due Friday April 21st

1 Jacobi's Method

Jacobi's method is an iterative, numerical method for solving a system of linear equations. Formally, given a full rank $n \times n$ matrix $A \in \mathbb{R}^{n \times n}$ and a vector $b \in \mathbb{R}^n$, *Jacobi's method* iteratively approximates $x \in \mathbb{R}^n$ for:

$$Ax = b$$

Given the matrix A,

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

we first separate A into its diagonal elements D and the remaining elements R such that $A = D + R$ with:

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}, \qquad R = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}$$

Jacobi's method then follows the following steps till convergence or termination:

```
1. Initialize $x$: $x \leftarrow \begin{bmatrix} 0 & 0 & \ldots & 0 \end{bmatrix}$
2. $D = diag(A)$
3. $R = A - D$
4. while $||Ax - b|| > l$ do
    (a) update $x \leftarrow D^{-1}(b - Rx)$
```

where $||x||$ is the L2-norm, and l is a parameter for the termination accuracy.

A matrix A is *diagonally dominant*, if its diagonal elements are larger than the sum of absolutes of the remaining elements in the corresponding rows, i.e., iff:

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|.$$

It can be shown that for *diagonally dominant* matrices, *Jacobi's method* is guaranteed to converge. The goal of this assignment is to develop a parallel algorithm for *Jacobi's method.*

2 Parallel Algorithm

Data distribution We use a 2-dimensional grid/mesh as the communication network for this problem. We assume that the number of processors is a perfect square: $p = q \times q$ with $q = \sqrt{p}$, arranged into a mesh of size $q \times q$. The inputs are distributed on the grid as follows:

- The n-by-n matrix A is block distributed onto the grid. The rows of A are distributed onto the rows of the processor-grid such that either $\lceil \frac{n}{q} \rceil$ or $\lfloor \frac{n}{q} \rfloor$ rows of A are distributed onto each row of the grid. More specifically, the first ($n \mod q$) rows of the grid contain $\lceil \frac{n}{q} \rceil$ rows of A, and the remaining rows of the grid contain $\lfloor \frac{n}{q} \rfloor$ rows of A. The same applies to the distribution of columns. A processor with coordinates (i, j) thus has the following size local matrix:

$$\begin{cases} \lceil \frac{n}{q} \rceil^2 & \text{if } i < (n \mod q) \text{ and } j < (n \mod q) \\ \lceil \frac{n}{q} \rceil \times \lfloor \frac{n}{q} \rfloor & \text{if } i < (n \mod q) \text{ and } j \geq (n \mod q) \\ \lfloor \frac{n}{q} \rfloor \times \lceil \frac{n}{q} \rceil & \text{if } i \geq (n \mod q) \text{ and } j < (n \mod q) \\ \lfloor \frac{n}{q} \rfloor^2 & \text{if } i \geq (n \mod q) \text{ and } j \geq (n \mod q) \end{cases}$$

- The size n vectors b and x are equally block distributed only along the first column of the processor-grid, i.e., only among processors with indexes $(i, 0)$. Processor $(i, 0)$ will thus have the following local size:

$$\begin{cases} \lceil \frac{n}{q} \rceil & \text{if } i < (n \mod q) \\ \lfloor \frac{n}{q} \rfloor & \text{if } i \geq (n \mod q) \end{cases}$$

Parallel Matrix-Vector Multiplication Let A be a n-by-n matrix and let x and y be n dimensional vectors. We wish to compute $y = Ax$. Assume that the square matrix A and the vector x are distributed on the processor grid as explained above. The answer y will again be distributed in the same fashion as the vector x. To do so, we will first "transpose" the vector x on the grid, such that a processor with index (i, j) will end up with the elements that were on processor $(j, 0)$ according to the above distribution. We do this by first sending the elements from a processor $(i, 0)$ to its corresponding diagonal processor (i, i), using a single `MPI_Send` and the sub-communicator for the row of processors. We then broadcast the received elements from (i, i) along each column of the grid using a sub-communicator for the column of processors.

Now, each processor has the elements it needs for its local matrix-vector multiplication. We multiply the local vector with the local matrix and then use a parallel reduction to sum up the resulting vectors along the rows of the grid back onto the processors of the first column. The final result of the multiplication thus ends up distributed among the first column in the same way that the input x was.

Parallel Jacobi For parallelizing *Jacobi's method*, we distribute A and R as described above and use parallel matrix-vector multiplication for calculating Rx. The vectors x, and b, are distributed among the first column of the processor grid. The diagonal elements D of A need to be collected along the first column of the processor grid. We can thus update x by $x_i \leftarrow \frac{1}{d_i}(b_i - (Rx)_i)$ using only local operations.

In order to detect termination, we calculate Ax using parallel matrix-vector multiplication, and then subtract b locally on the first column of processors (where the result of the multiplication and b are located). Next, we calculate the L2-norm $||Ax - b||$ by first calculating the sum of squares locally and then performing a parallel reduction along the first column of processors. We can now determine whether to terminate or not by checking the L2-norm against the termination criteria and then broadcasting the result along rows. Now, every processor knows whether or not to continue with further iterations. In your implementation, you should not only check for the termination criteria, but also terminate after a pre-set maximum number of iterations.

3 Task

In this assignment, you will implement parallel *Jacobi's* method using MPI. The program should terminate when it hits the termination criteria of 10^{-9}, or reaches the maximum iteration of $1,000,000$.

As it was, you can form a team up to three members. No matter how you decide to share the work between yourselves, each student is expected to have full knowledge of the submitted program.

3.1 Input & Output Format

The input to the algorithm is two files containing the matrix A and vector b. The output of the program is the vector x. File format is described in more details below. The program should take 3 command line arguments: `input_mat.txt`, `input_vec.txt` and `output.txt`. You have to structure the program such that one processor reads the input file and distributes the data among all the processors. We are doing this for convenience sake, and because we do not have a machine that supports truly parallel I/O. Note that when you time your algorithm, you should start the timer after the data has been block distributed and end it before you write the vector x to the file.
Sample command line input:

```
$ mpiexec -np 4 ./pjacobi input_mat.txt input_vec.txt output.txt
```

3.2 File Format

Sample input and output files are also provided for reference in the *Programming Assignments* folder on Canvas.

- `input_mat.txt`: contains $n + 1$ lines for an $n \times n$ matrix A
  - **First line**: value of n
  - **Following n lines**: space-delimited elements of the matrix
    (**Datatype - Doubles**)

- `input_vec.txt`: contains 1 line for the vector b
  - **First line**: n space-delimited elements of vector b
    (**Datatype - Doubles**)
- `output.txt`: contains 1 line for the vector x
  - **First line**: n space-delimited elements of vector x
    (**Datatype - Doubles, upto 16 decimal points**)

4 Deliverables

1. Source files: All C/C++ program source files. You should use good programming style and comment your program so that it is easy to read and understand. Make sure that your submission is self-contained.
2. Makefile: A makefile for the program. Make sure the output executable is named *pjacobi*.
3. Report: A PDF report containing the following
   - Names and GT usernames of your team members at the very beginning
   - Short design description of your algorithm(s)
   - Experimentally evaluate the performance of your program. For instance, fix a problem size, vary the number of processors, and plot the run-time as a function of the number of processors. Fix the number of processors and see what happens as the problem size is varied. Try to come up with some important observations on your program. What is the minimum problem size per processor to get good parallel efficiency? We are not asking that specific questions be answered but you are expected to think on your own, experiment and show the conclusions along with the evidence that led you to reach the conclusions. At least TWO plots are required.
   - Explain any ideas or optimizations of your own that you might have applied and how they might have affected the performance
   - Distribution of team effort. Members with significantly less effort will get fewer points.

Submit (a) all the source files and the Makefile in "Programming Assignment 3 Code" on Gradescope. Make sure you are NOT uploading the executable file in your submission, and (b) submit your report in "Programming Assignment 3 Report" on Gradescope. Only ONE student needs to make the submission for the whole group. If students of the same group makes two different submissions, there would be a minor penalty applied.

5 Grading

Your program will be graded on correctness and use of the appropriate parallel programming features. You should also use good programming style and comment your program so that it is understandable. Grading consists of the following components:

1. Correctness (60 pts)
    - Automated tests on Gradescope that will test your program functionality for various type of input matrices.
2. Performance (20 pts)
    - We will go through your code to make sure appropriate parallel programming practices discussed in the class are being followed along with the right MPI functions being called. Particularly, we will check codes for:
        - Appropriate distribution of data over processors
        - Parallel implementation of matrix-vector multiplication
        - Check for termination criteria

      `Note`: Points will be deducted for ignoring performance protocols; **serialization in the program will lead to a zero on the whole assignment.**
3. Report (20 pts)
    - Design description (5 pts)
    - Evaluation of performance (15 pts)

6 Resources

1. What is a Makefile and how does it work?: https://opensource.com/article/18/8/what-how-makefile
2. PACE ICE cluster guide: https://docs.pace.gatech.edu/ice_cluster/ice-guide/. Documentation for writing a PBS script: https://docs.pace.gatech.edu/software/PBS_script_guide/
3. Jacobi's method: https://en.wikipedia.org/wiki/Jacobi_method