CX 4220/CSE 6220 Introduction to High Performance Computing
Spring 2017
Programming Assignment 3
Due Tuesday April 25

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: $x \leftarrow \begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix}$
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.*

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.

Assignment

In this assignment, you will implement *Jacobi's* method using MPI. A framework is provided with this programming assignment, which declares all functions that you should implement. You are required to implement your solution within the provided framework hosted on Georgia Tech's Enterprise GitHub page at: github.gatech.edu/ucatalyurek7/cse6220-Sp17-src/tree/master/prog3-template. In the event that we have to update the framework, we will publish the updates in this repository and notify the class via T-Square. The framework comes with some pre-implemented unit tests. You are encouraged to add more test cases to make sure that your implementation works as expected.

First, you should implement matrix-vector multiplication and *Jacobi's* method sequentially. You will later use this as the comparison for measuring speedup and as reference implementation for testing your parallel code.

Implement the sequential code in the file `jacobi.cpp` according to the function declarations in `jacobi.h`. Unit tests for the sequential code are implemented in `seq_tests.cpp`. Add your own test cases in this file. To compile and run the tests, run `make test` either locally on your own machine, or in an interactive job. Do NOT run the tests on the login node of the cluster.

Next, you should implement the parallel algorithm as described above. For this, you'll have to implement functions to distribute the data among the grid of processors and gather results back to the processor with rank $(0, 0)$. Then, implement the parallel algorithms for matrix-vector multiplication and Jacobi's method. Implement your parallel code in the file `mpi_jacobi.cpp` according to the function declarations in `mpi_jacobi.h`. A few test cases are already provided in the file `mpi_tests.cpp`. You may also want to add your own test cases in this file as well. Some utility functions for block distributions are provided in `utils.h` and `utils.cpp`. You can make use of these function, and implement your own utility functions in these files.

Finally, test the performance of your parallel implementation against your sequential implementation for various input sizes. Use the executable `jacobi` for this and generate the input on-the-fly via the `-n` and `-d` (difficulty) parameters. Report your achieved speedups for different input sizes and difficulties. Increasing difficulty increases the number of iterations needed for *Jacobi's Method* to converge.

Submission Policies

This assignment is due April 25, 11:55pm EST on T-Square. You are expected to work in teams of two. No matter how you decide to share the work between yourselves, each student is expected to have full knowledge of the submitted program. **And only one member from each team should submit a zip file (file name : assign_3.zip) containing the following on T-Square**

1. A text file containing the names of all team members along with their contribution to the assignment (file name : names.txt)

2. 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 implementation does not rely on any extra files or functions implemented outside of the provided source files.

3. A PDF report containing the following (file name : report.pdf)

   - Short design description of your algorithm(s)
   - Runtime and speedup plots by varying problem size and difficulty. Give observations on how your algorithm behaves on varying the parameters. Show the results of your experiments, and the conclusions along with the evidence that led you to reach the conclusions.
   - Explain any ideas or optimizations of your own that you might have applied and how they might have affected the performance.

The program will be graded on correctness and use of the appropriate parallel programming features.

Notes

We will enforce the specified polices strictly. You will risk losing points if you do not adhere to them. We will check all submissions for plagiarism. Any cases of plagiarism will be taken seriously, and dealt in accordance with the GT academic honor code (honor.gatech.edu).