LINMA2710 – Scientific Computing – 2021-2022

Project 3

Deadline: Friday 19th May 2023 18:15

Statement

Consider the Poisson equation:

$$\nabla^2 u = f(x, y)$$

where ∇^2 is the Laplace operator. We consider a square domain $[0, 1]^2$ with $(n + 1) \times (n + 1)$ equally spaced grid points and $n \times n$ cells of dimension (h, h), i.e. $h = 1/n$. Given a function f, and Dirichlet boundary conditions ($u^0_{i,j}$ is given for $i = 1, n + 1$, $j = 1, n + 1$), our aim is to find the value of u that solves this equation using the Jacobi iteration

$$\begin{aligned}
&u^0_{i,j} \text{ given} && i = 1, n+1 \text{ or } j = 1, n+1 \\
&u^0_{i,j} = 0 && 1 < i < n+1, 1 < j < n+1 \\
&u^{k+1}_{i,j} = u^k_{i,j} && i = 1, n+1 \text{ or } j = 1, n+1 \qquad (1) \\
&u^{k+1}_{i,j} = \frac{u^k_{i-1,j} + u^k_{i+1,j} + u^k_{i,j-1} + u^k_{i,j+1} - h^2 f_{i,j}}{4} && 1 < i < n+1, 1 < j < n+1 \qquad (2)
\end{aligned}$$

We use the stopping criterion

$$\sum_{i=1}^{n+1} \sum_{j=1}^{n+1} (u^k_{i,j} - u^{k-1}_{i,j})^2 = \sum_{i=2}^{n} \sum_{j=2}^{n} (u^k_{i,j} - u^{k-1}_{i,j})^2 < \varepsilon. \qquad (3)$$

The equality between the sum with and without the boundary points is due to (1), it means that we can use either sum for the criterion.

We ask you to solve the Poisson equation with the Jacobi iteration using several processors communicating with MPI. You are tasked to implement a 2D version which is developed in paragraph 4.2 of the MPI book [1]. To help you, we already provide a portion of the implementation; you have one class and two functions left to implement.

1 Distributed matrix

For a given k, the large matrix $u^k_{i,j}$ is distributed across the processes arranged in a grid. Each process has a block of the matrix in memory in addition to the adjacent adjacent rows and columns of their neighbors. Each time the matrix is updated, the adjacent processes need to communicate rows and columns as illustrated in Figure 1.

$$\begin{array}{ccccc}
0 & \leftrightarrow & 1 & \leftrightarrow & 2 \\
\updownarrow & & \updownarrow & & \updownarrow \\
3 & \leftrightarrow & 4 & \leftrightarrow & 5 \\
\updownarrow & & \updownarrow & & \updownarrow \\
6 & \leftrightarrow & 7 & \leftrightarrow & 8
\end{array}$$

Figure 1: Illustration for the the distribution of the matrix for processes with ranks from 0 to 8. The symbol $\leftrightarrow$ denotes column exchange and the symbol $\updownarrow$ denotes row exchange between the processes.

The `DistributedMatrix` class emulates the full matrix but in each process, only the part of the matrix stored on the process is accessible.

Each block is represented as a `SubMatrix` which is described next.

2 Tasks

2.1 SubMatrix class

The `SubMatrix` class implements a matrix whose rows and columns can be communicated with MPI. It implements

- A constructor with `int numRows` and `int numCols` as argument creating a `numRows` × `numCols` zero matrix.
- A destructor freeing the memory.
- A `Size` method.
- A `Read` method for 1-based read-only access to its entries.
- A `operator()` method for 1-based read-write access to its entries.
- A `void SendReceiveRows(int rowsend, int ranksend, int rowrecv, int rankrecv, int tag, MPI_Comm comm)` method that sends the row with (1-based) index `rowindex` to the process with rank `ranksend` and receive the row with (1-based) index `rowrecv` from the process with rank `rankrecv` using the tag `tag` and the communicator `comm`.
- A `void SendReceiveColumns(int colsend, int ranksend, int colrecv, int rankrecv, int tag, MPI_Comm comm)` method that sends the column with (1-based) index `colindex` to the process with rank `ranksend` and receive the column with (1-based) index `colrecv` from the process with rank `rankrecv` using the tag `tag` and the communicator `comm`.

Hints Represent the matrix using a single array `double *` containing all columns (or rows) stacked so that the full matrix is stored in a contiguous memory. Depending on how you represent the matrix (row wise or column wise), either synchronizing rows or columns will be easier as it is represented as a contiguous block of memory. For the other one, consider using strides (google it!) to tell MPI that the elements of the row (or column) are separated by a certain space in memory.

The following MPI elements can help you fulfill the tasks `MPI_Type_vector`, `MPI_Type_commit`, `MPI_Type_free`, `MPI_Datatype`, `MPI_Sendrecv`.

2.2 poisson file

The `poisson.cpp` (resp. `poisson.hpp`) files should contain the implementation (resp. signature) of the following two functions.

2.2.1 jacobi_iteration function

The `void jacobi_iteration(const DistributedMatrix &a, const DistributedMatrix &f, DistributedMatrix &b)` function implements the Jacobi iteration (see (1) and (2)) where the current iterate u^k is read in `a`, the f function is `f` and the next iterate u^{k+1} is written in `b`.

2.2.2 sum_squares function

The `double sum_squares(const DistributedMatrix &a, const DistributedMatrix &b)` function returns the left-hand side of the criterion (3).

Here the function `MPI_Allreduce` can help.

2.3 Benchmark

Benchmark your code with different numbers of processes. What speed-up do you observe ? Does that agree with your expectation and with the theory ? Detail your findings in the report.

For this question you will have to connect to the CÉCI clusters ("Consortium des Équipements de Calcul Intensif") which are computing facilities[1]. They consists in several clusters managed by

[1] Connecting to the clusters is a requirement for this project and is a part of the grade.

several universities in Wallonia and Bruxelles, you will use "Lemaitre3". Connections are made using `ssh` and computing resources are managed with `slurm`, a detailed documentation on how to do it is here https://support.ceci-hpc.be/doc/. (When you create your account refer to prof. Absil as the academic of reference, it will be asked. Note that it might take a day or two for your account to become active.)

The computing resources are also used by researchers. We kindly ask you to be reasonable with the number of processors and time of computing. Limit yourself to medium scale problems: sufficiently large to see improvement due to parallelization but not more (maximum 1min. problems are largely enough). Do not ask for more than 32cpus (we will not consider numerical results with strictly more than 32cpus).

Note that the clusters run on a version of linux (as a wide majority of servers in the world) and that you will access them through the command line.

If you are not acquainted with the command line please check one of the many online resources for beginners.

You can launch the Makefile we furnish to compile and run the codes, in our Makefile "make run" to compile and run, and "make runtests" to compile and run the tests, see inside the Makefile what is being done with a text editor).

An issue you will encounter is how to change the code files on the server. There are several options, one that works well for us is to use a text editor with the capability to connect in ssh. For exemple `VSCode` with the `ssh` extension. Another recommended option is the use of `sshfs` to synchronize a local directory with the server.

If you have an issue with the servers please refer to the documention of the CÉCI, to our Google doc with the FAQ and to references online. If this does not work contact us.

Instructions

1. **Fraud:** As always for this course, you must do all the writing (report, code, user manual...) individually. Never share your production. However, you are allowed, and even encouraged, to exchange ideas on how to address the assignment.
2. **Plagiarism:** As always, you must cite all your sources.
3. **Submission:** Using the Moodle assignment activity, submit your report in a file called `Report_Project 3_FirstName LastName.pdf`. The report should be short (maximum 2 pages) and should include

   - Your benchmark.

   The deadline is strict. (We do not enforce the deadline on Moodle, but submissions after the deadline may not be considered.) On Inginious, submit your files `SubMatrix.hpp`, `SubMatrix.cpp`, `poisson.hpp`, `poisson.cpp` containing your implementation.

   You are allowed to make as many submissions as you need, only the last submission will be taken into account. You are advised to verify that your submission passes the tests in Inginious early before the deadline. On Inginious, the files `test.cpp`, `main.cpp` available on Moodle are compiled together with the classes submitted and results of the tests included on `test.cpp` are checked.
4. **Language:** Reports in French are accepted without penalty. However, English is strongly encouraged. The quality of the English will not impact the grade, provided that the text is intelligible.

References

[1] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. Vol. 1. MIT press, 1999.