Project 2

MATH 471/571 - Parallel Scientific Computing

Michal A. Kopera

First revision date: April 3rd, 2022 (will give you feedback if you submit by then)

Final submission: April 10th, 2022 (no more redo's after that date)

Please complete the tasks outlined below and submit a PDF write-up along with the code and other files (Makefile, run script) you have used to obtain the results. I will grade your project and provide you with some feedback if you submit by the first revision date. You will then have until the final submission date to make any corrections necessary.

The projects are your individual work. I am happy to clarify some muddy points, but I would like to see your individual attempt to solve the project. If you got help from somebody, or used a resource outside class (i.e. website) please provide appropriate acknowledgement and/or reference. **You can** discuss your ideas on Slack, but **you cannot** share your codes.

Problem description

In this project you will work with two-dimensional solver for diffusion-reaction equations, which occurs often in chemistry, ecology and life sciences. We are going to consider the Barkley model[1], which consists of two interacting equations:

$$\frac{\partial u}{\partial t} = f(u,v) + \nabla^2 u, \tag{1}$$

$$\frac{\partial v}{\partial t} = g(u,v), \tag{2}$$

where $u,\ v$ are concentrations of two chemical species, and $f(u,v)$ and $g(u,v)$ are the production (reaction) terms for species $u,\ v$, respectively. The reaction terms are given by:

[1] Dwight Barkley (2008), Scholarpedia, 3(11):1877, http://www.scholarpedia.org/article/Barkley_model

(3) $$f(u,v) = \frac{1}{\epsilon}u(1-u)\left(u - \frac{v+b}{a}\right),$$

(4) $$g(u,v) = u - v,$$

where $\epsilon,\ a,\ b$ are constant parameters. For more discussion on the physical meaning of the parameters see the footnote on the first page.

We will solve the model in a square 2d domain $(x,y) \in [-L,\ L] \times [-L,\ L]$ and use the no-flux boundary conditions on all boundaries. The no-flux condition means that on the left and right boundaries we have:

$$\frac{\partial u}{\partial x} = 0,$$

and on the top and bottom boundaries we enforce

$$\frac{\partial u}{\partial y} = 0.$$

We implement this boundary condition by setting the value at the ghost point to be the value of the neighboring point in the domain, i.e. $u_{0,j} = u_{1,j}$ for enforcing $\frac{\partial u}{\partial x} = 0$ at the left domain boundary, or $u_{i,N+1} = u_{i,N}$ for enforcing $\frac{\partial u}{\partial y} = 0$ at the top boundary. Once you set the ghosts, you can evaluate equations (7), (8) (see below) for the boundary points, i.e. $u_{1,j}$ or $u_{i,N}$. Remember to apply the no-flux boundary condition at all boundaries.

The initial condition will be:

(5) $$u(x,y) = \begin{cases} 1 & \text{if } y \geq 0 \\ 0 & \text{if } y < 0 \end{cases},$$

(6) $$v(x,y) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}.$$

To approximate the solution of the model given by equations (1-2), we divide the model domain into a grid of $N \times N$ points and use the finite difference approximation for the diffusion term:

(7) $$\nabla^2 u_{i,j} \approx \frac{1}{\Delta x^2}\left(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}\right),$$

where indices $i,\ j$ number points in the $x,\ y$ directions, respectively, and Δx is the grid spacing (we assume $\Delta y = \Delta x$).

We use forward Euler explicit time integration (not a good idea in general, but will do for our project):

$$(8) \qquad u_{i,j}^{n+1} = u_{i,j}^{n} + \Delta t \left(f(u_{i,j}^{n},\, v_{i,j}^{n}) + \frac{u_{i-1,j}^{n} + u_{i+1,j}^{n} + u_{i,j-1}^{n} + u_{i,j+1}^{n} - 4u_{i,j}^{n}}{\Delta x^2} \right),$$

$$(9) \qquad v_{i,j}^{n+1} = v_{i,j}^{n} + \Delta t\ g(u_{i,j}^{n},\, v_{i,j}^{n}),$$

where index n represents the time level at which we take the value of the variable.

Task 1 - Run the serial code

You are provided a serial code for this problem under barkley_serial.c, and there is also a Matlab script which can read and plot the data files produced by the serial model.

Read carefully the code and identify where the computation happens. Identify where the boundary conditions are applied (see the description above).

1. Run the code and plot the results to establish a benchmark for the next steps. Use the parameters $\epsilon = 0.02$, $a = 0.75$, $b = 0.01$, $L = 150$ and $t_{final} = 40$. The code accepts a command line parameter N, which is the number of points in $x,\ y$ directions. Use at least 1000 points. The simulation may take a while (upwards of 1 h), so make sure you run it well in advance, and use a run script with enough resources (i.e. time in the queue).
2. Show plots of u at t_{final}. You should get an image of a spiral. Show the image with the appropriate description in your report.
3. Report on how much time it took to complete the run.

Task 2 - Parallelization

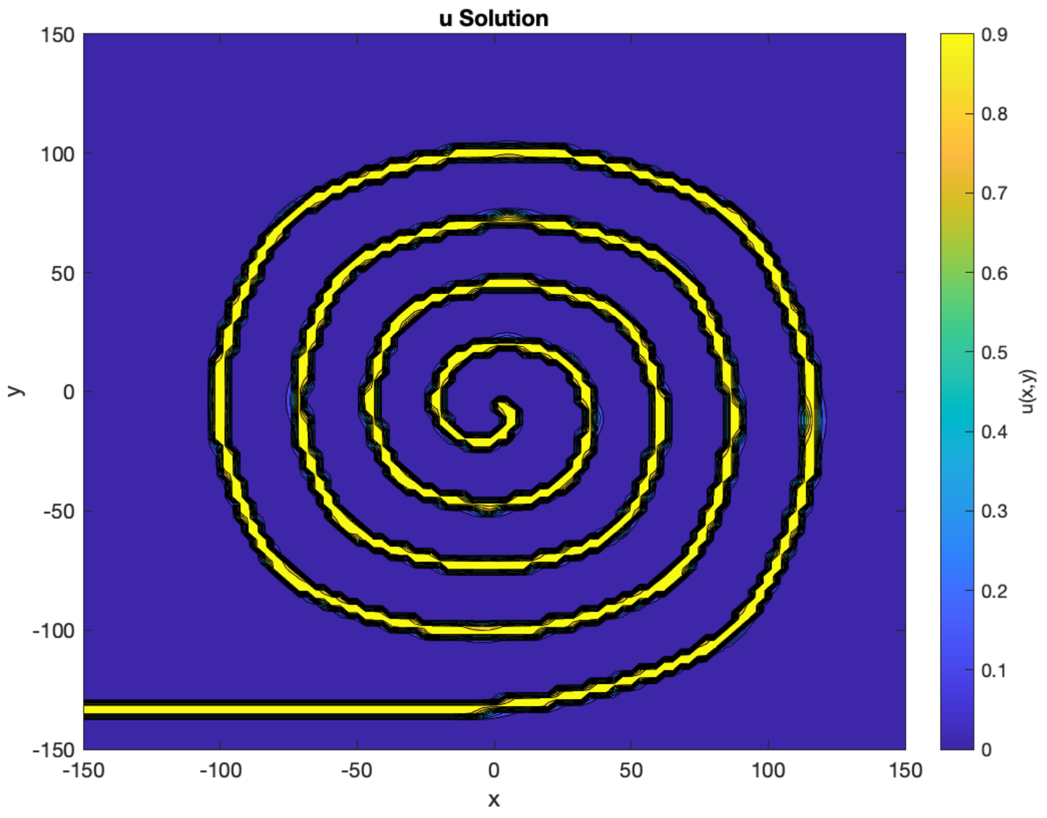
Modify your code from Task 1 to implement a parallelization strategy to complete this task. You can use either MPI_Sendrecv, or MPI_Send and MPI_Recv pairs to complete the communication steps. You can use functions from Week 10 which write data to a file, read command line arguments, etc.

Complete the following steps:

1. Decide on the parallelization strategy. You can decompose the domain in a grid of $q \times q$ processes, like we have done in class. Or, you can decide on a pencil decomposition, where each of the p processes computes $N \times N/p$ chunk of grid points (rather than $N/q \times N/q$), where N is the number of points in x and y

directions. Once you decide how to decompose the grid between processes, calculate how many messages each process will send, and how big (in terms of floating point numbers) the messages will be. **Put those numbers in your report, and indicate your choice of decomposition strategy.**

2. Implement the algorithm using either MPI_Send and MPI_Recv pairs, MPI_Sendrecv, MPI_Sendrecv_replace, or a combination of the above.
3. Show the correctness of your implementation by presenting the solution at final time, using parameters as in Task 1. Report on the configuration used (how many processes, how many points) and the time it took to complete the simulation. You may want to work on a smaller problem (N=100 points, for example) when you debug, but I want you to report the time for the large (at least N=1000) simulation. Mind that the lower resolution result will look slightly different than the high resolution one, but they should be similar enough. For example, here is the result for N=100:



Task 3 - How fast can you run this simulation?

The serial simulation takes quite a bit of time. We would like to decide what is the optimal choice of how many processes to use to run in the optimal time, and with optimal cost.

Complete the following steps:

1. Perform a strong scaling study of the algorithm. Increase the number of processes until you see that the efficiency is dropping and you do not get any faster simulations. This may require a few trial and error. You may want to turn off writing data to a file to complete the scaling. Remember also that the number of processes p should be a square number, and N should be divisible by $\sqrt{p}$.
2. Plot the time to solution as a function of processes used, as well as strong scaling efficiency. In your opinion, what is the best choice of number of processes to run this simulation as fast as possible? Explain your thinking.
3. (optional) To help you reason about how the efficiency plays into choosing the optimal settings, plot the processor-hours used to compute solution with different numbers of processors/ The processor-hours can be computed by multiplying the time it took to complete a simulation by the number of processes used. It is proportional to energy used by the computer, so represents the economic cost of the simulation. How does this plot compare with the efficiency plot?

Mastery

To further improve the strong scaling of the diffusion-reaction problem above we can try the non-blocking communication. We will have discussed this concept in Week 10, but I put a little cheat-sheet in the appendix below.

Complete the following steps:

1. Design an algorithm using non-blocking communication, which overlaps communication with computation. Implement your algorithm for the Barkley model and show correctness by running the same simulation as in Task 1. It is easier to use the MPI_Wait function, but is may be more efficient to use MPI_Test in a smart way, as it will not force all processes to wait for one communication to be done at a time. You can use either of them.
2. Run a strong scaling experiment and compare the results to scalings in Task 3. Do you get an improvement in parallel efficiency? How about the time to solution?

3. **(571 students only)** For each run in your strong scaling experiment, measure the time of the computation part of your code behind which you try to hide the communication, and measure the time from the beginning till the end of the communication steps (excluding the computation that happens after you are done communicating). Compare both times on the plot versus the number of grid points per rank (the size of the array which each process has - $N_{loc} \times N_{loc}$) What is a minimum number of grid points points per process which guarantees a complete hiding of the communication cost?

Appendix

The non-blocking communication uses the following commands:

MPI_Isend - initializes sending of a selected buffer to a selected rank. Issues

MPI_Irecv - initializes the receiving of data from a selected rank to a buffer

Before using the data in the receive buffer, we need to check whether the communication has been completed. Both MPI_Isend and MPI_Irecv use a request variable. We can check whether a request has been completed by using either of the two functions

MPI_Wait - forces the processes to wait until a request is complete. It is a blocking function.

MPI_Test - checks whether a request is completed, and returns a flag, which has a value of 1 if the request is completed, and 0 otherwise. It is a non-blocking function, so allows for a more efficient implementation.

The syntax of all the functions is given below, feel free to look them up online.

```
int MPI_Isend(void *send_buffer, int count, MPI_Datatype
datatype, int destination, int tag, MPI_Comm comm, MPI_Request
*request)

int MPI_Irecv(void *recv_buffer, int count, MPI_Datatype
datatype, int source, int tag, MPI_Comm comm, MPI_Request
*request)

int MPI_Wait(MPI_Request *request, MPI_Status *status)

int MPI_Test(MPI_Request *request, int *ready, MPI_Status
*status)
```