Project 2

MATH 471/571 - Parallel Scientific Computing

Michal A. Kopera

First revision date: March 17th, 2021 (will give you feedback if you submit by then)

Final submission: March 24st, 2021 (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. You submit your solution by putting it in a folder named Project 2 in your Google Drive personal folder.

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. Please refer to the Syllabus regarding how the project points and mastery points affect your final grade. You can earn one project point for each complete task, and one mastery point for completing the mastery part. There is no partial credit, so you have to complete all the steps to earn a point.

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:

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

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

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

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:

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

$$g(u, v) = u - v, \tag{4}$$

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:

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

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

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:

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

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):

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

$$v_{i,j}^{n+1} = v_{i,j}^{n} + \frac{\Delta t}{\Delta x^2} g(v_{i,j}^{n}), \tag{9}$$

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

Task 1 - Write a serial code for the Barkley model

This time I do not provide you with a serial code as a starting point. You can use the 2d code we used in Week 9 classes and modify the equations to represent the Barkley model. One significant difference will be the fact that in the diffusion equation we used Jacobi iteration to find a steady state solution, while in the Barkley model we are back to time-dependent equations. All it means, that for the Barkley model you will repeat the computation sequence for a prescribed number of time-steps until you reach final time, while for the diffusion equation we iterated until convergence.

Complete the following steps:

1. Modify the code from class to solve the Barkley model.

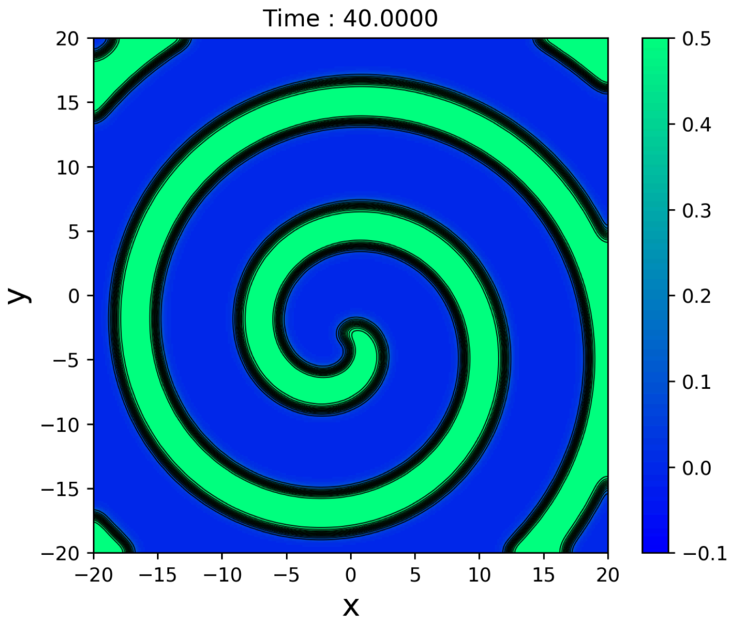

Fig 1. Spiral wave solution for variable u at time $t_{final} = 40$

2. Use the parameters $\epsilon = 0.02$, $a = 0.75$, $b = 0.01$, $L = 20$ and $t_{final} = 20$ and run the model. Show plots of u at t_{final}. You should get a pattern which looks like the spiral in Fig. 1. below. You can use the matlab script from Week 9 to plot your results, but please change the surface plot to a contour plot.
3. (optional) You may want to explore this problem further by checking what is going to happen for other choices of parameters. Try $\epsilon = 0.02$, $a = 0.75$, $b = 0.02$, $L = 150$ and check both u and v solutions at selected times.

Task 2 - Parallelization using 2D decomposition

In class we worked on the parallelization of the Poisson equation using a 2D decomposition of the computational domain. Modify your code from Task 1 to implement a similar 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 9 which write data to a file, read command line arguments, etc.

Complete the following steps:

1. Design the flowchart of the parallel algorithm (it can be in graphical form, or alternatively a description of the algorithm using numbered list) which outlines the computation and communication steps.
2. Implement the algorithm using either MPI_Send and MPI_Recv pairs, MPI_Sendrecv, or combination of both.
3. Show the correctness of your implementation by presenting the solution at final time, using parameters as in Task 1.
4. Perform a strong scaling study of the algorithm by choosing large enough problem size to see some decent scalability, but small enough to see the departure from good efficiency for large processor count. Feel free to use as many as 128 processes, if it makes sense (i.e. your efficiency does not drop before then). This may require a few trial and error. Please explain your choices of problem size. Remember that you choose N, but the actual problem size is NxN, so N=1000 results in 1000000 points being used. 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 $\sqrt{p}$. Plot the strong scaling efficiency and write your conclusions from the study.

Task 3 - 1D decomposition

We have choices in how we decompose the data between processes in our algorithms. In this task you will explore 1D decomposition and compare it to the results of Task 2. In the 1D decomposition, each process gets a “pencil” of data (either horizontal or vertical) of size $N \times \frac{N}{p}$. The benefit of this approach is that now we only worry for the problem size N to be divisible by p, and don’t care whether p is a square number. The communication patters appears to be simpler, as well, since we send the data only left-right (or top-bottom, if you chose to decompose into horizontal “pencils”). The question remains how this choice affects the strong scaling efficiency.

Complete the following steps:

1. Implement the 1D decomposition in the Barkley model and show the correctness of your results by running the same simulation as in Task 1 and plotting the solution at final time.
2. Perform the strong scaling experiment using the same problem size as in Task 2. Plot the strong scaling efficiency for 1D and 2D decomposition on one plot.
3. Based on the result in point 2 above, which of the decompositions would you rather choose?

Mastery

In Week 10 we will have discussed the concept of non-blocking communication. This mastery task is aiming for you to explore how the non-blocking communication affects the strong scaling of the Barkley model.

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)
```

Complete the following steps:

1. Design an algorithm using non-blocking communication, which overlaps communication with computation. Show the algorithm as a graphical block diagram, or as a numbered list. Indicate at which points you initialize the communication, and where you check for completion.
2. 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. Use MPI_Test if you want an additional challenge.
3. Run a strong scaling experiment and compare the results to scalings in Tasks 2 and 3. Do you get an improvement in parallel efficiency?
4. 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? Is there a relation between the relative length of the computation and communication steps and the strong scaling efficiency?