BOISE STATE UNIVERSITY

(BSU, USA)

Name: Brian KYANJO
Course: Parallel Scientific Computing

Project Number: 2
Date: March 19, 2021

Problem description

I have worked on two-dimensional solver for diffusion-reaction equations, which occurs often in chemistry, ecology and life sciences. I have considered the Barkley model1, 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.

Task 1 - Write a serial code for the Barkley model

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

   The code has been modified and Barkley model has been solved. The modified code named **barkley.c** is attached with this report.

2. Parameters $\epsilon = 0.02$, $a = 0.75$, $b = 0.01$, $L = 20$, and $tfinal = 40$ have been used to run the model, and a plots of u at tfinal = 40 has been obtained as shown in the figure 1 below;

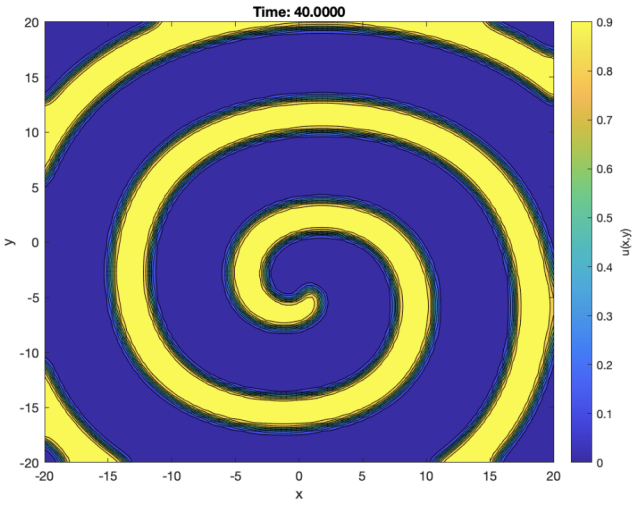


Figure 1: Task 1

The spiral in figure 1, looks like the spiral the question paper.

3. Optional - Parameters $\epsilon = 0.02$, $a = 0.75$, $b = 0.02$, $L = 150$, and $tfinal = 40$ have been used to run the model with $N = 100$, and a plots of u at tfinal = 40 has been obtained as shown in the figure 2 below;

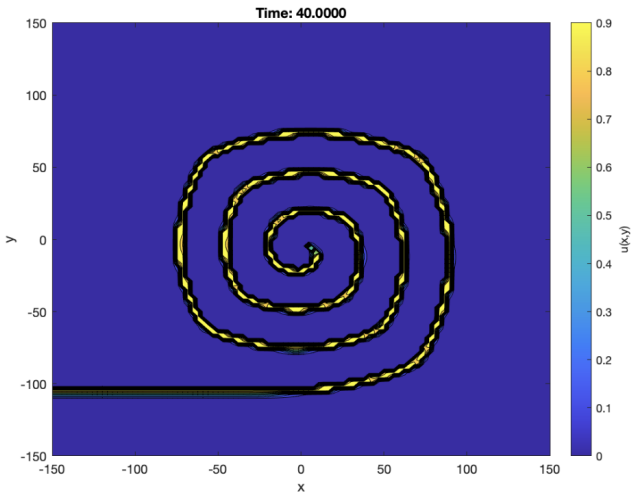


Figure 2: Task 1 (optional)

The spiral in figure 2, becomes choatic, due to change of parameters.

Task 2 - Parallelization using 2D decomposition

1. Below is the numbered list of the algorithm

   (a) Initialize MPI and then compute row and column index in the rank grid and grid spacing.

   (b) Allocate buffers and arrays.

   (c) Intialize x,y and array u.

   (d) Start time loop.

   (e) Initialize Boundary conditions (BC) in fictious cells.

   (f) Go over interior points and then enforce BC from the fictious cells

   (g) Start communication by first packing data to send buffers from ghosts.

   (h) Move data to left(101), right(102), botton(103) and top(104) respectively for all ranks using MPI_Sendrecv, MPI_Send, and MPI_Recv.

   (i) Then unpack data from recv buffers to ghosts and then finialize communication.

   (j) Update solution and end the time loop

   (k) Write resul ts and stop.

2. The program (**barkley_mpi.c**) attached with this report implemenents a MPI_Sendrecv, with a combination of MPI_Send and MPI_Recv.

3. The correctness of the code is shown in the figure 3 below by using the parameters in Task1 and obtaining the same plot as shown below.

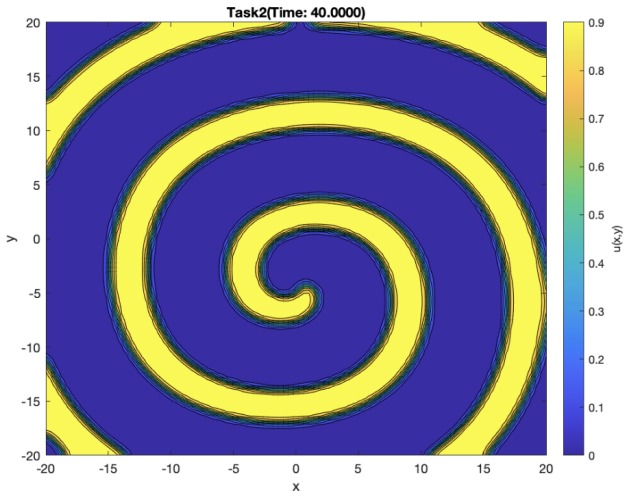


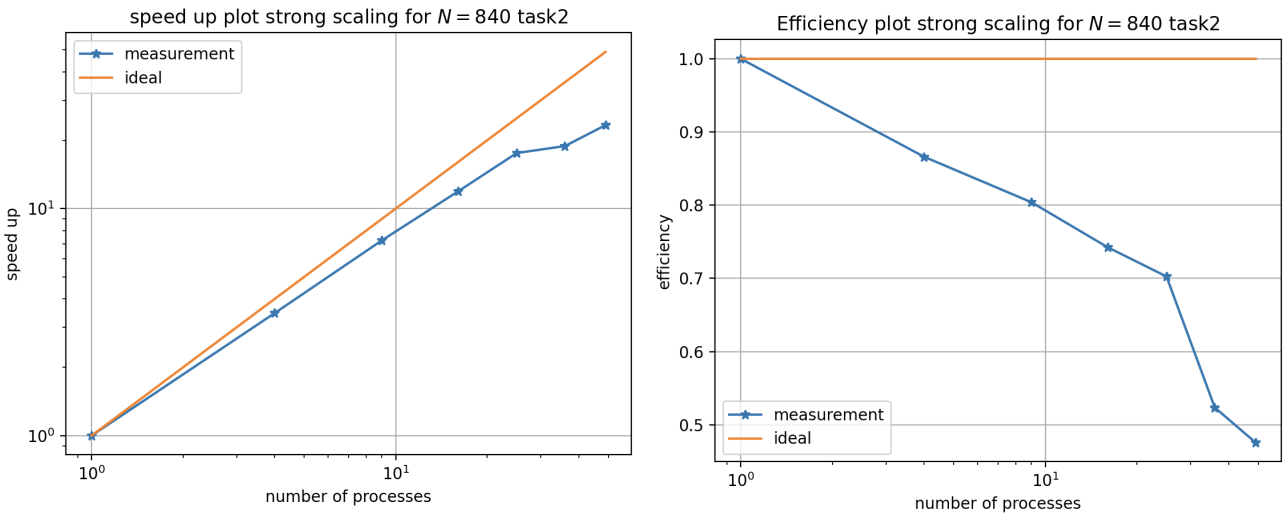
Figure 3

4. A strong scaling has been performed with choosing a problem size of $N = 840$, $dt = 0.025$ and $tfinal = 10$, as shown in the figure below. Due to cfl issues, i was forced to choose $N = 840$ and number of processors: $1, 4, 9, 16, 25, 36, 49$ which are square numbers. Beyond this the system becomes unstable, since the Numerical method we are implementing is unstable, so choosing an appropriate dt and N, for large process count is quite challenging. I have tried many options but i ended up stopping on 49 processors for now. And at that the efficiency is already dropping below 50%, which implies that 49 processors are enough for me to run the tasks.

   The figures: 4a and 4b, represent speed up and efficiency plots for Task 2 strong scaling.

   Figure 4a depict linear scaling for number of processors:$1, 4, 9, 16$, and 25 after which scaling drops. This means that the program was perfectly scaled up to 25 processors then scaling drops.

   Figure 4b depicts that parallel efficiency decrease with increase in number of processors with a fixed problem size, This is because according to Amdahl's law as the number of processors increases, with a fixed problem size, for the first resourses, this is due to load imbalance and overhead, and minimised concurency, hence as the processes increases to large count, parallel efficiency decreases up to below 50



(a) (b)

Figure 4: (a) and (b) respectively show speedup and Efficiency plots for strong scaling Task2.

Task 3 - 1D decomposition

1. The 1D decomposition (**barkley_task3.c**) has been implemented in the Barkley model and depicted correctness of the results after running the same simulation as in Task 1 and plotting the solution at final time. The plot similar to the taht in Task 1 after uing the same parameters is shown in figure 5 below;

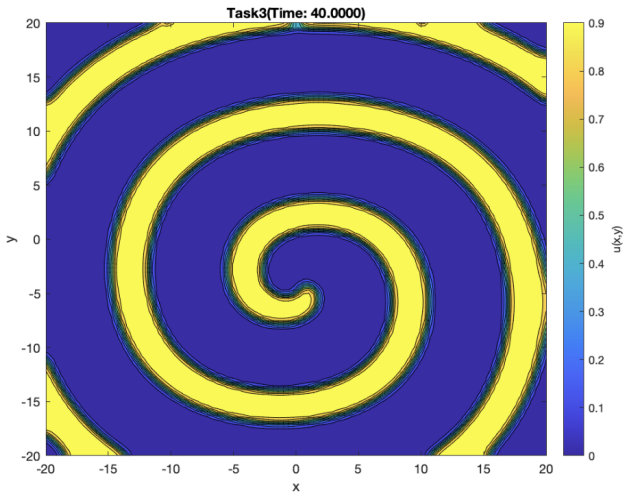


Figure 5: Task 3

2. The strong scaling experiment has been performed using the same problem size as in Task 2. And the strong scaling efficiency for 1D and 2D decomposition are ploted on the same plot as shown in figure 6 .

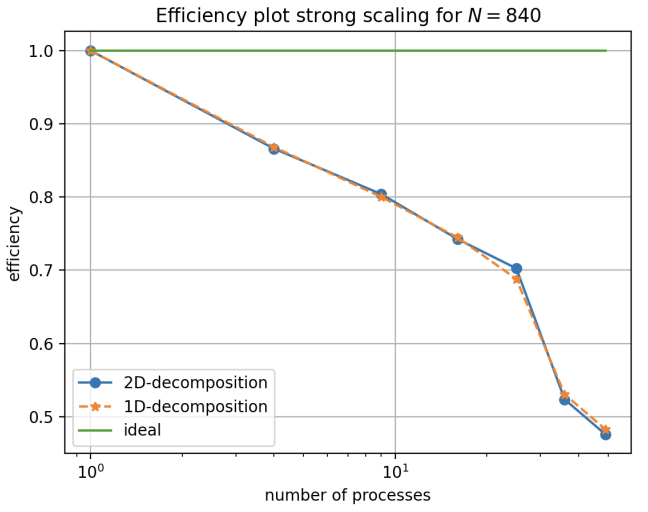


Figure 6: Efficiency plot for both 2D and 1D decomposition.

3. Based on the result (6) in part 2 above, I would choose 1D decomposition, since it exhibits overall good efficiency even for large processor counts compared to 2D decomposition.

Mastery

1. The program (**barkley_mastery.c**) using non-blocking communication has been designed. And below is a numbered list of the algorithm.

   (a) Start by declaring variables for tracking requests and the intialize MPI.

   (b) Compute row and column index in the rank grid and the grid spacing.

   (c) Allocate arrays and buffers.

   (d) Initialize x, y and array u.

   (e) Start the time loop

   (f) Initialize Boundary conditions(BC) in ghost cells and then intialize communication by packing data to send buffers.

   (g) Compute neighbour indices.

   (h) Initialize receives and sends.

   (i) Go over interior points and then enforce BC by copying values from ghosts.

   (j) Check if communication is complete, if complete, then complete computation at the rank edges.

   (k) Complete computation.

   (l) Update the solution and close the time loop.

   (m) Write out results.

2. The algorithm for the Barkley model has been implemented and it exibited correctness after running the same simulation as in Task 1. The plot obtained is shown in figure 7 below;

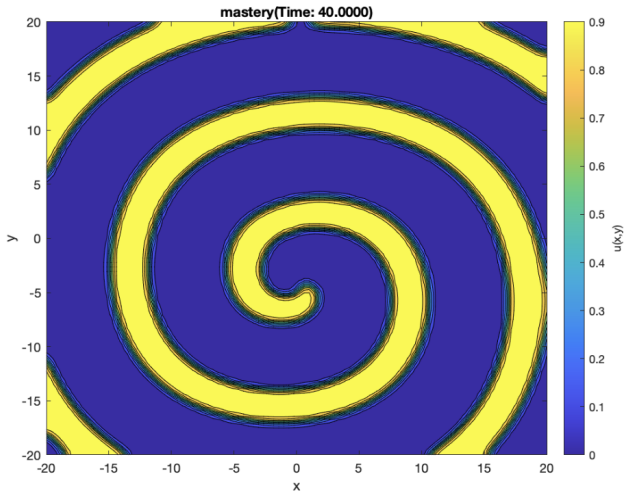


Figure 7: Mastery

3. The strong scaling experiment has been performed and compared to the results of scalings in Tasks 2 and 3, by ploting all of them on the same plot, as shown in figure 8.

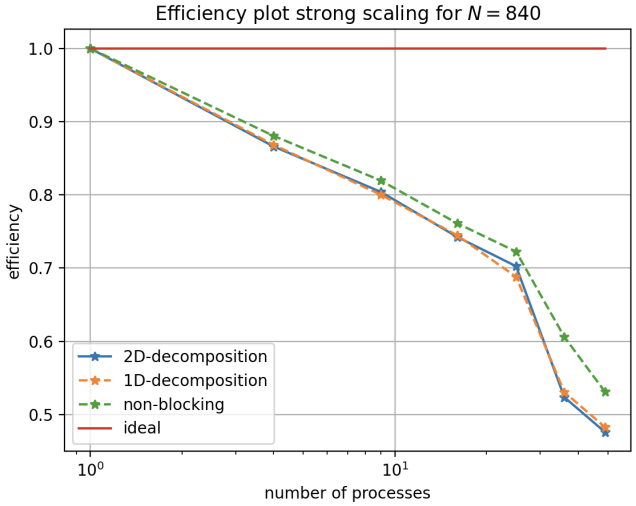


Figure 8: Efficiency plot: 2D, 1D, and non blockig

As shown on figure 8, non-blocking communication depicts an overall good efficiency through out the run, and maintains it, even for large processor counts, hence an improvement in parallel efficiency. With we can conclude that nonblocking communication perfectly scales the program compared to 2D and 1D.

4. For each run in the strong scaling experiment, the time of the computation part of the code behind which you try to hide the communication and, the time from the beginning till the end of the communication steps (excluding the computation that happens after you are done communicating) has been measured.

   Both times have been compared by ploting them together on the same plot against N_loc^2 as shown in figure 9.

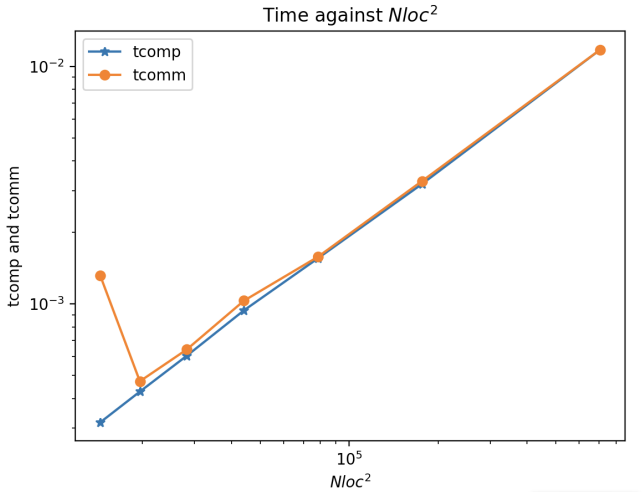


Figure 9: Computation and Communication time

Figure 9, shows that working on the full grid size without dsicretising it costs more on communication than computation time, as the problem is further discretised, with more processor counts, communication time scales with computation time.

The minimum number of grid points points per process which guarantees a complete hiding of the communication cost is 28224 grid points.

There is a relation between the relative length of the computation and communication steps and the strong scaling efficiency. Since strong scaling efficiency η is gievn by equation (3).

$$\eta = \frac{\text{serial time}(T_1)}{\text{parallel time } (T_p) * \text{number of processors(nproc)}} \tag{3}$$

So according to equation (3) above, the length of computation and communication steps affects parallel time, and since η is inversly propotional to T_p, therefore a change in T_p greatly affects η. So computing at the same time communicating which is the non blocking communication, helps alot in obtaining good strong efficiency.