SHPC4001 - Assignment 10

Ayden S. McCann

May 2021

Code Listing at the end

1

1.a

The two bugs found in the given program were:

1. the array to which the received array was to be stored was not initialised for rank 1
2. rank 1 was trying to receive from itself rather than rank 0.

1.b

The given code for part 1b and 1c was modified by first partitioning A (renamed a_global) into local arrays (a_local) with dimensions (n,size*n). The global array was then scattered from root 0 into the local arrays, where the computation:

$$a\_local+ = rank \tag{1}$$

Was performed. Using the supplied print statement format the desired outputs are achieved.

1.c

Using the starting point given in the assignment the send and receive statements were completed, most notably to include ranks in the middle (where both upper and lower arrays are required to be both sent and received). Where applicable the upper and lower values were added to their respective local arrays, printing the values gives the desired output (albeit sometimes out of order).

1.d

Following the steps outlined in the assignment the averages of all a_local elements were taken and stored in an array called averages, the sum of which were broadcast back into each rank.

1.e

The a_local arrays were gathered to rank 0, where they were reshaped into a (9,9) matrix (the shape of the original a_global array).

2

Using the same line of thought as Question 1, the Jacobi iteration was completed using the following steps:

1. Define the dimensions (num_points) and divide this evenly by the number of ranks (n = axis 0 size of local arrays)
2. Define initial conditions in rank 0 and initialise m (equivalent of a_global in part 1)
3. Jacobi Function
   (a) Takes input of m
   (b) Creates local arrays, local update arrays and local error arrays
   (c) m scattered into local and update arrays to recover initial conditions
   (d) Creates empty arrays for upper and lowers for a given rank
   (e) While loop controlled by error (epsilon in the slides)
      i. Define m_local as m_local_update
      ii. Calculate and send upper and lowers
      iii. Define m_local_update as 4 nearest m_local neighbours. (This step contains a lot of if statements to ensure upper and lower values are called when appropriate)
      iv. Calculate errors per row
      v. Find the max error per rank
      vi. Gather these ranks
      vii. Find the max of these errors and broadcast it to all ranks as the parameter that controls the while loop
   (f) create empty array for gathered local arrays (size,n,num_points)
   (g) reshape and broadcast this array into m

(h) return m, the count and the error

4. Call and time Jacobi function

5. Save plot (.png), the time and outputs from function into a .txt

This was run on my own laptop with 3 ranks, an accepted error of 0.001 and a 60x60 initial matrix. The plot is shown as Figure 1, comparison with part_2_expected_output.png confirms they would only differ by the point at which iterations ceased. The output of the .txt file is as follows:

****
number of processes = 3
finished after 2485 iterations
error = 0.000999218583129409
function run time = 5.0462679999999995

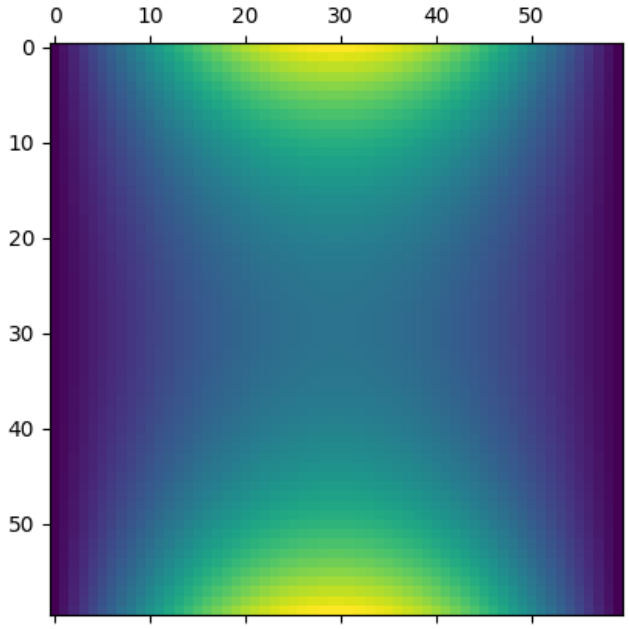

Figure 1: The result of the code described in Section 2 being run with the following parameters: 3 ranks, an accepted error of 0.001 and a 60x60 initial matrix.

3

This code was then run on the QUISA workstation for a 256x256 matrix and an accepted error of 0.001. It was run for 16,8,4 and 2 processes. The resulting Figure (each iteration was identical) is shown in Figure 2, the performance of the code with respect to number of processes is shown in Figure 3 and the outputs from the .txt file are below:

```
**********
number of processes = 16
finished after 27129 iterations
error = 0.0009999681299176812
function run time = 229.5242794980295
**********
number of processes = 8
finished after 27129 iterations
error = 0.0009999681299176812
function run time = 280.9082504779799
**********
number of processes = 4
finished after 27129 iterations
error = 0.0009999681299176812
function run time = 527.7014292930253
**********
number of processes = 2
finished after 27129 iterations
error = 0.0009999681299176812
function run time = 862.7828196940245
```

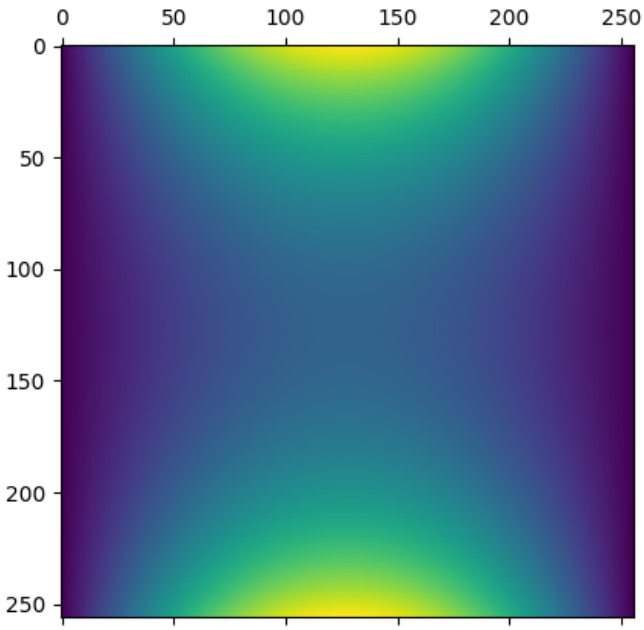

Figure 2: The result of the code described in Section 3 being run with the following parameters: an accepted error of 0.001 and a 256x256 initial matrix. Note this particular iteration used 2 processes but experimentation (and the .txt file readout) showed the number of processes did not change the results.

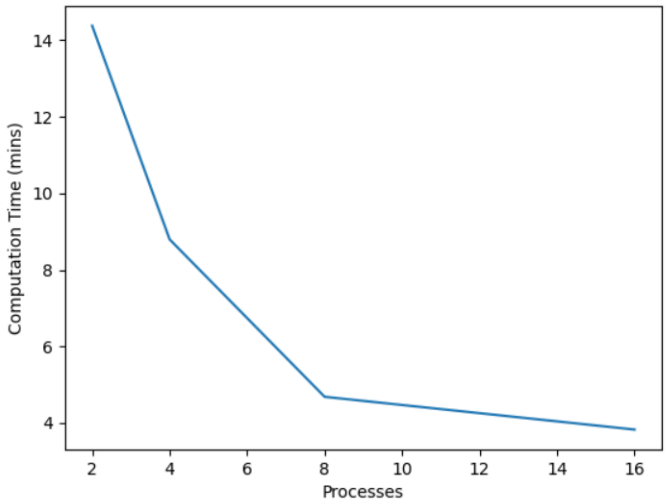

Figure 3: The time required to complete the code detailed in Section 3 with an accepted error of 0.001 and a 256x256 initial matrix.

4 Code Listing

Note: 1a.py through 1e.py only contain code up to that point. eg 1e.py contains all Question 1 material (with earlier print statements commented out).

Question 1a:
1a.py

Question 1b:
1b.py

Question 1c:
1c.py

Question 1d:
1d.py

Question 1e:
1e.py

Question 2:
2.py

Question 3: slurm (extensionless)