TDT4200-Problem Set 2 - Programming Part 1 MPI - Image Convolution and Morphing (Graded)

Tor Andre Haugdahl, Zawadi Svela and TDT4200 Staff

September 15, 2021

Deadlines:

- **Theory:** TBD (Pass/Fail)
- **Programming - Part 1 & 2: Sept 29, 2021** by 22:00 in BB

This assignment must be done individually and w.o. help from others other than TDT4200 staff. We encourage that clarification questions be posted on the Forum in BB so all can benefit. All sources used found on the internet or otherwise must be referenced

Grading: This assignment is graded and counts towards your final grade in this course. Programming Part 1 and 2 together counts about 15% of your final grade and must be passed to take the next PS and the final in the course. Code documentation and code clarity will count 10% of the grade on this assignment, i.e 1.5% of your final grade. E.g document each function and what is does, variable names, etc

MPI programming - Imaging – Part 1

In this exercise you are going to parallelize two programs using MPI. The first one (Part 1) is a image convolution program while the second (Part 2) is a implementation of the Beier-Neely image morphing algorithm.

MPI-Imaging Part 1: Parallel image convolution using MPI

In this program, a convolutional kernel is shifted across an image in order to produce a new image. A kernel is an equation that determines the value of a pixel in the new image based on pixels in the old one. A typical kernel will look like this:

$$N[i,j] = \begin{array}{lll} -1 * O[i-1,j-1] & -4 * O[i,j-1] & -1 * O[i+1,j-1] \\ -4 * O[i-1,j] & +20 * O[i,j] & -4 * O[i+1,j] \\ -1 * O[i-1,j+1] & -4 * O[i,j+1] & -1 * O[i+1,j+1] \end{array} \quad (1)$$

In words, for each color channel, the value in a new pixel $N[i,j]$ is a sum of the corresponding old pixel $O[i,j]$ and its neighbors. If we are applying the above kernel, and for the sake of example only look at a single color channel (say, red), then we'd multiply the "here"-pixel's red-value by 20, and then subtract 4 times the red-value of the orthogonal neighboring pixels and subtract 1 times the red-value of the diagonally neighboring pixels. The sum would be the new pixel's red-value. This process is then repeated across all pixels and channels, and often across multiple iterations.

These kernels are often provided in matrix form, the one corresponding to the kernel above would look like this:

$$\begin{array}{ccc} -1 & -4 & -1 \\ -4 & 20 & -4 \\ -1 & -4 & -1 \end{array} \quad (2)$$

The hand-out code in this assignment is a partially implemented parallel MPI program:

- Rank 0 reads the image and distributes it using MPI_Scatter.
- Each rank performs the convolution on its own partition for some number of iterations
- The result is gathered at rank 0 again and written to file.

The challenge is that because of the shape of the kernels, each process needs the value of pixels outside it's own partition. This is the problem you will have to solve by implementing what is called *border exchange*.

Tasks

Note: All necessary changes should be doable only by adding code to the main() function. You can assume the number of processes are a power of 2.

1. Compile and run the code using at least 2 processes. Look at the resulting image. You should see a solid line on the border between the partitions.

   You are provided a Makefile. This means the program can be compiled by simply running the command "make" in the root folder of the handout code. This will produce an executable called "main".

2. Fix the border exchange. Think about:
   - Which pixels need to be sent to which processes?

- What order of communication is best to ensure quick and deadlock-free communication?
- Do the local image partitions have room to store received pixels? And do you need to make changes to scatter/gather if you change this?

3. Using MPI_Wtime, measure how much time the program spends from start to finish using a single process (mpirun -np 1). Test with 2, 4 and 8 processes and document the *speedup* you get compared the single-process case.