TDT4200 PS 2b - MPI Graded and Mandatory

A. C. Elster, B. Gotschall and Z. Svela

Due Friday 10:30pm, Sept 25, 2020

Introduction

This Problem Set counts 10% towards your final grade. Points are given as follows, with 100pts is considered full score, where points are given as follows:

Table 1: GRADING of PS 2b

Task	Points
Task 1	35 - 90 points depending on the solved stage
Task 2	10 points

In this assignment, you will implement a parallel solution to a general convolution algorithm using MPI. This algorithm can be used for image edge detection, sharpening or blurring. Our methodology for dividing the image and distributing it over parallel processes will be based on Fredrik Berg Kjølstad's paper on ghost cells (pdf).

The method is shifting a convolutional kernel over an image which computes the underlying data together with the kernel weights into a new value. In simpler words: a new pixel is generated by multiplying the pixel and its neighbor values with kernel coefficients and summing them up. Some kernels are multiplying the accumulated value at the end with a constant factor (e.g. the gaussian kernel). This mathematical operation allows relatively easy to blur or sharpen images. Special kernels can also amplify edges, which is called edge detection. The mentioned paper includes the following laplacian kernel as an example. Let O be the original image and N the new image. The convolutional operation using a 9-point laplacian kernel can be expressed as follows (see also Figure 7a in Kjølstad's write-up):

$$\begin{aligned} N[i,j] = \quad & -1 * O[i-1, j-1] \quad -4 * O[i, j-1] \quad -1 * O[i+1, j-1] \\ & -4 * O[i-1, j] \quad +20 * O[i, j] \quad -4 * O[i+1, j1] \\ & -1 * O[i-1, j+1] \quad -4 * O[i, j+1] \quad -1 * O[i+1, j+1] \end{aligned} \tag{1}$$

When applying this kernel to a Mandelbrot rendering (Figure 1) we will get an output image on which only edges of the original are marked in Figure 2. By

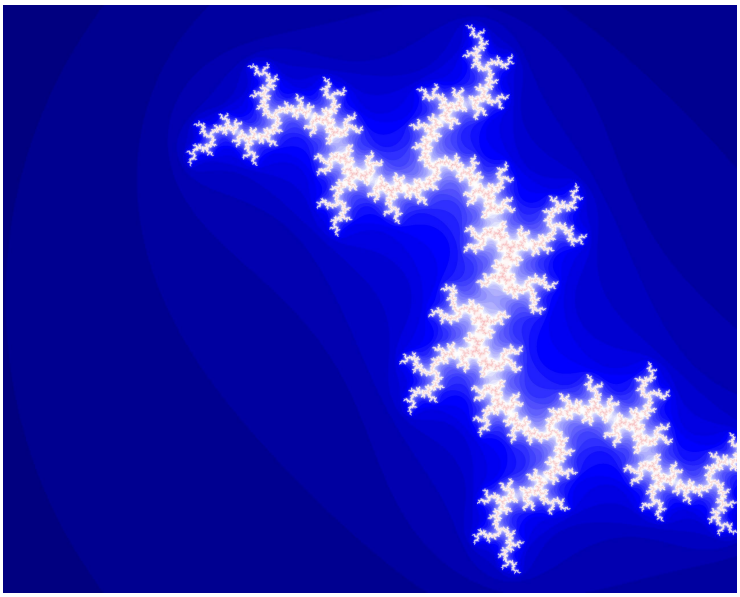

Figure 1: Example Mandelbrot rendering

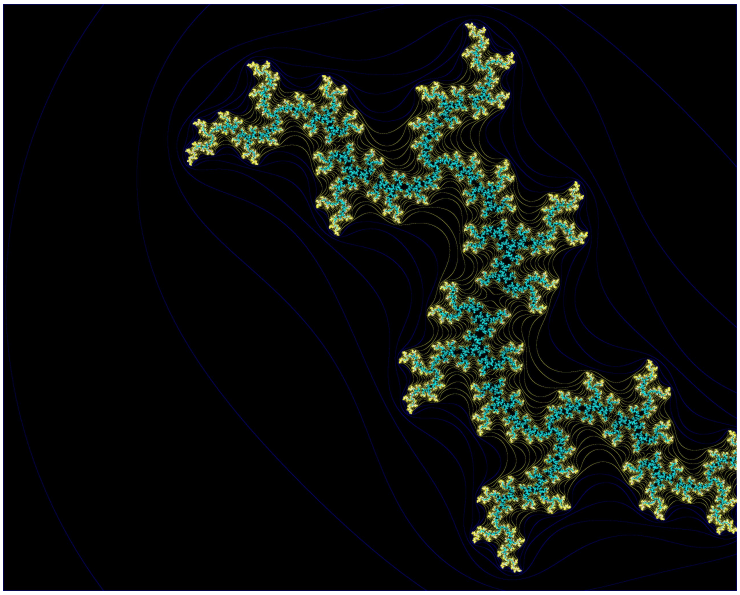

Figure 2: 9-point laplacian kernel applied on the example Mandelbrot rendering from Figure 1

re-applying this kernel on the new image, and repeating the process a number of times, we will produce an image where the most prominent lines are highlighted even stronger.

With this assignment you are handed a code which reads in a bmp image, applies a chosen kernel a number of iterations and saves the result in a new bmp image. However, the code which is provided is completely serial. **Your task is to parallelize the code using MPI.**

As described in the paper and above, the algorithm computes a pixel based on its own value as well as the values of all 8 of its nearest neighbors. This poses a challenge when calculating the next iteration on the local boarders since some of the data needed are stored on other processes.

Preparation

NB! You do not need to document this task

1. Get familiar with the code
2. Search for the TODOs in the code and implement the time measurement for the kernel operation.
3. Try out the provided kernels and different iterations and see what it means for the output image. Also get familiar with the impact on the runtime when using higher dimension kernels or different iterations.

   ```
   ./main -k [0-5] -i [iterations] before.bmp after.bmp
   ```

4. Take a baseline with the Laplacian 1 kernel (-k 2) and some representative iterations beginning at 1 to at least 1024 (-i 1 ... -i 1024)

1 Parallelize with MPI

NB! This task has multiple stages that can be achieved with increasing difficulty and points to obtain. Apart from mentioning which stage you have solved, this task does not need any further documentation (not to be confused with properly commented source code).

You are now going to parallelize the provided code using MPI. Only the master rank is allowed to parse the arguments, read in the input image and write out the output image. It then distributes the image data in chunks of rows to each rank. Every rank applies the convolutions kernel only on the rows it has received and as many iterations as required. The tricky part is that each rank needs actually more rows than it has initially received to apply the kernel. A kernel dimension of 3x3 needs for example one more row at the top and bottom to execute the convolution on its rows. This is solved by the so called

border exchange or halo exchange. That means before each iteration every rank has to receive the bottom rows from the northern rank and the top rows from the southern rank and respectively send its own top and bottom rows to the northern and southern rank. **Do not** gather the whole image back at the master rank every iteration but only at the very end after all iterations are done. This forces you to setup a communication pattern between the single ranks which is more efficient in terms of bandwidth. Let only the master rank do the time measurement and start the time measurement before any MPI communication takes places (to include all overhead that is introduced through parallelisation) and end it after the final image has been gathered back to the master rank. Your solution should ideally work and achieve a speedup with any input image resolution, any number of processes, any iterations and any kernel dimension. However you can obtain points by solving easier stages.

The following stages can be reached. If you are able to solve a higher stage, you do not have to solve or submit any earlier stages. **All stages have to achieve a speedup that scales with the number of processes and it has to work with any number of iterations.**

(i) works with 2 processes, a kernel dimension of 3x3, and the provided input image. [35 points]

(ii) works with any number processes, a kernel dimension of 3x3, and the provided input image [70 points]

(iii) works with any number of processes, a kernel dimension of 3x3, any input image [75 points]

(iv) works with any number of processes, any kernel dimensions, and any input image [90 points]

2 Number crunching

NB! This task needs to be documented.

2.1 Pen and Paper

Take the provided input image and the Laplacian 1 kernel, and think about a parallel computation using 4 processes and 2 iterations.

1. Describe shortly the MPI communication pattern over the whole execution [4 points]
2. Calculate how much data has to be transferred between the processes in total [2 points]
3. Calculate how much bigger the communication overhead is by running 8 processes with the same parameters. [2 points]

2.2 Speedup analysis

NB! You have to solve at least one stage of Task 1 to do the speedup analysis.

1. Calculate the speedup with running at least 2 processes to the baseline taken during preparation. [2 points]

Deliverables

The documentation must be delivered as a PDF file together with the source code packed in a single ZIP file and submitted in time on blackboard. The code must be commented and in a runnable state. Do not use any libraries apart from standard libraries or those provided. Do not change the calling conventions (parameters) of the application.

Provide only **one** version of your code! It should be the latest (most advanced) code from task 1. The following tasks require documentation (if solved):

- Task 2

Do not include binaries or object files in your delivery and don't include any BMP image files.

PS 2a (Theory), will be a mandatory pass/fail assignment on Blackboard.