SOBEL EDGE DETECTION

Alexander Robles, Khadichabonu Valieva

CSC 410

University of Southern Mississippi

School of Computing Sciences and Computer Engineering

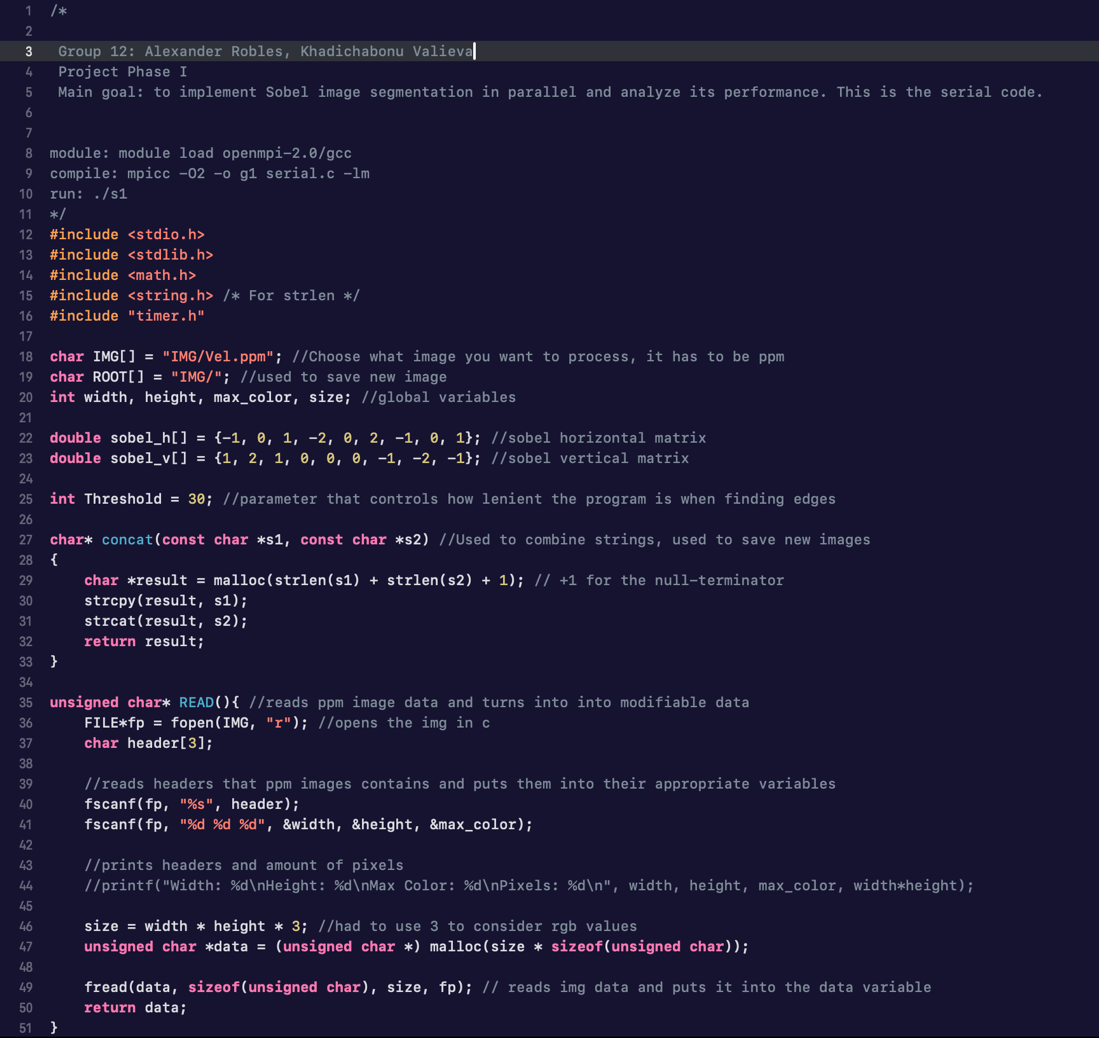
**SOBEL EDGE DETECTION**

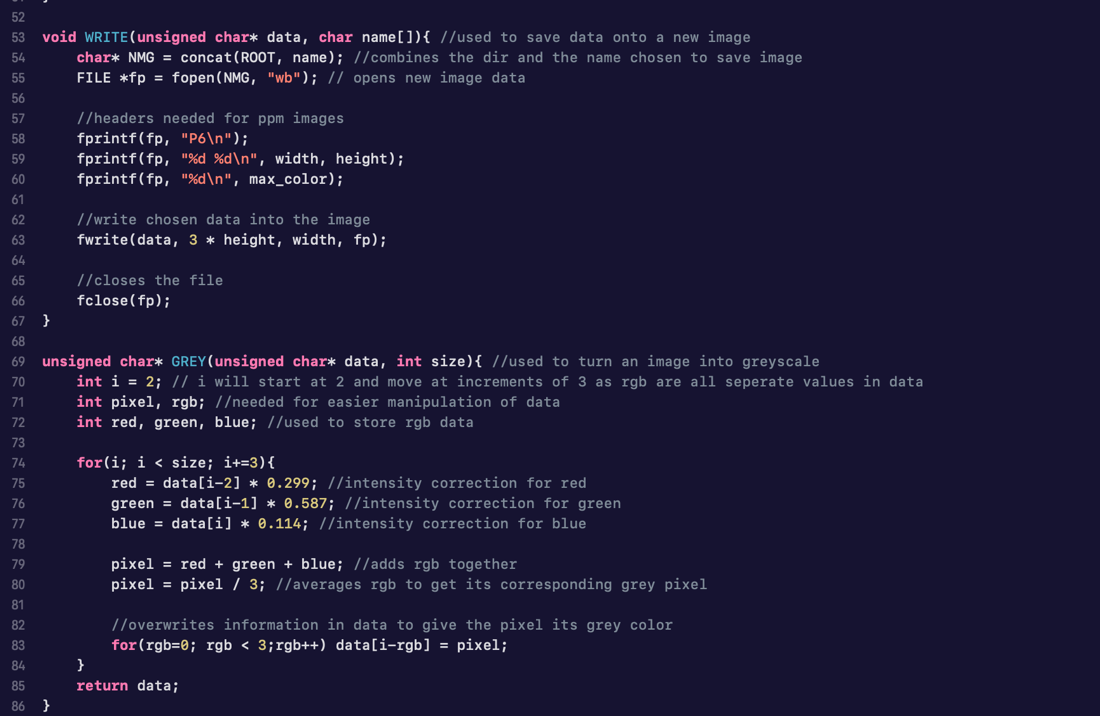
**Problem Description, Methods, and Techniques**

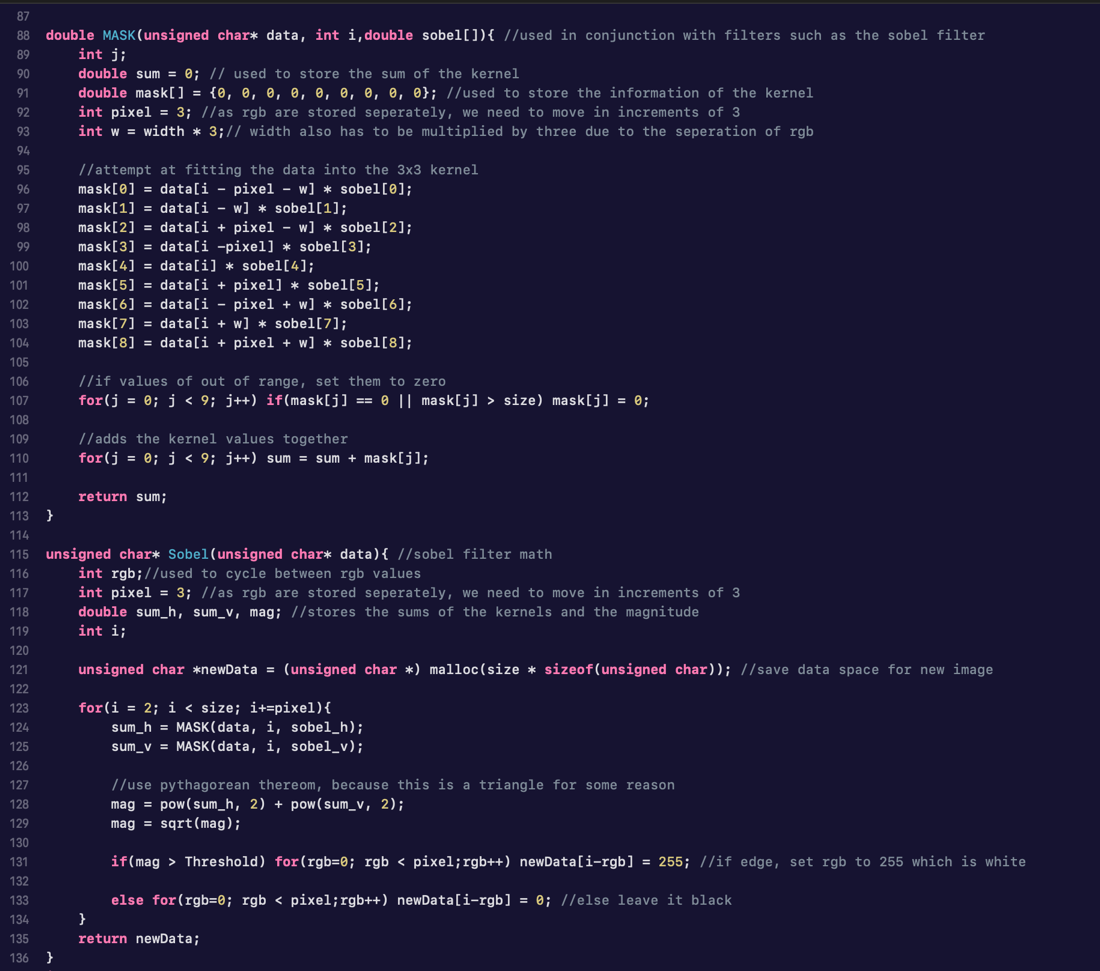
“The project aims to implement Sobel image segmentation in parallel and analyze its performance. The Sobel operator will be employed to identify edges within the image, which will serve as the basis for segmentation.”

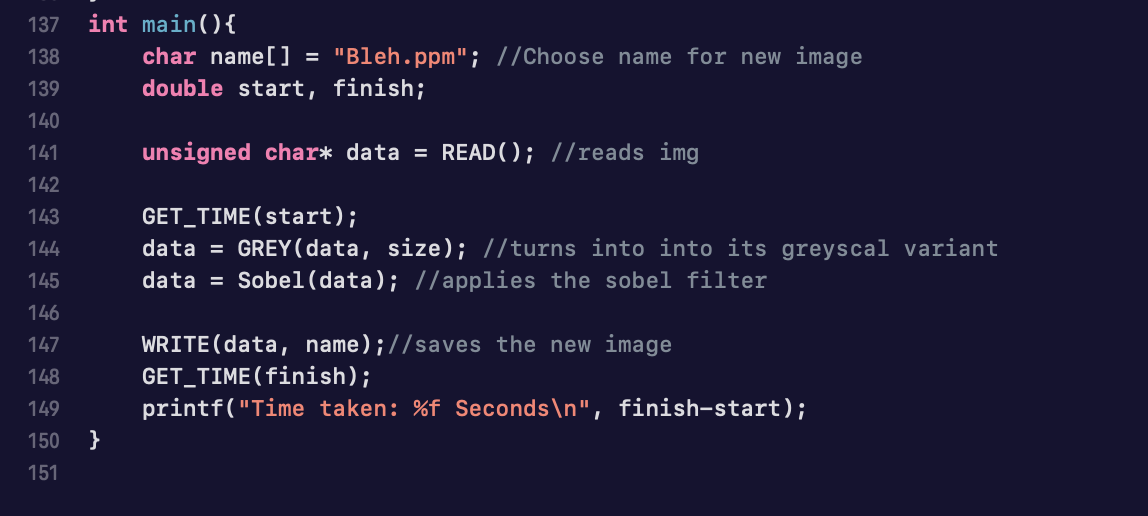
For this project we used convolution filters, which is a 3x3 matrix repeated over every pixel in the image using the Sobel matrixes used to compare surrounding pixel RGB values to tell if the current pixel is an edge or not. We also have included the timer.

**Serial implementation using C and its results.**

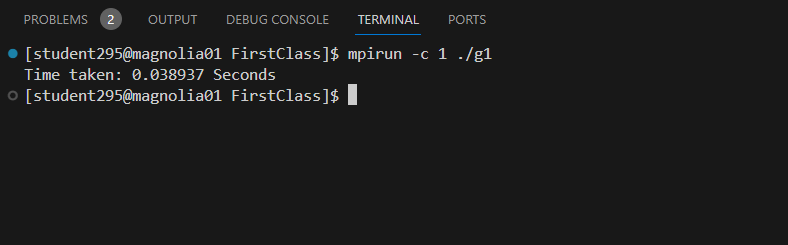






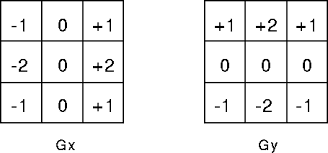


Result:



**Design of parallel algorithm**

Sobel operator is a discrete differential operator, it computes the gradient approximation of image intensity function for image edge detection. We used 3x3 kernels/masks to calculate the vertical and horizontal Sobel matrices:

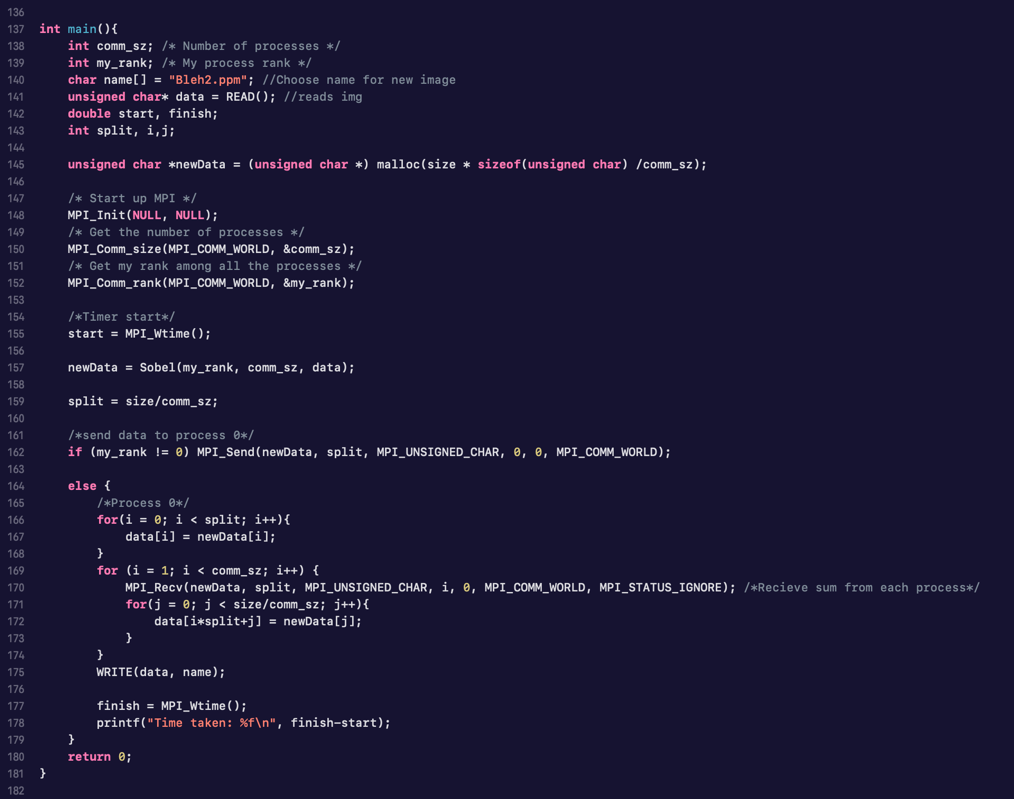


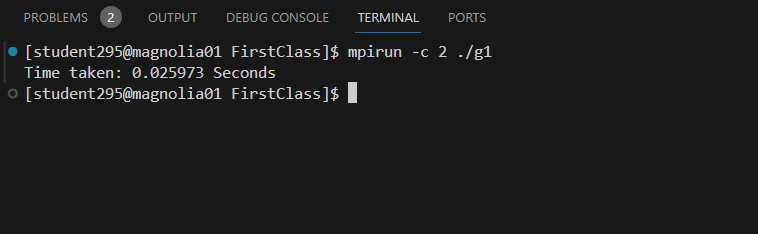
The process included reading an image, converting the actual RGB image to the grayscale image, then we use MASK function that applies a Sobel filter to a pixel of the image, as well as Sobel function that calculates the Sobel edge detection for a portion of the image based on the rank of the process and comm\_sz. Finally, in the main function, we initialize MPI, use basic MPI commands, read the image, execute Sobel function in parallel based on the rank and size, and finally, splits the image among processes and sends it to process 0 where it gathers all data and rebuilds the final image.

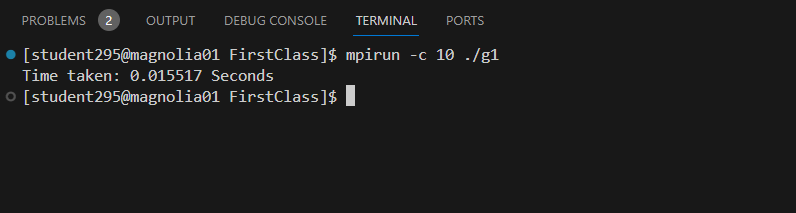
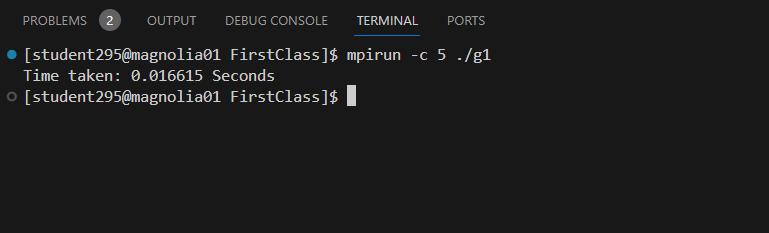
-------------------- ----------------------- ------------------- ---------------- --------------- -------------------

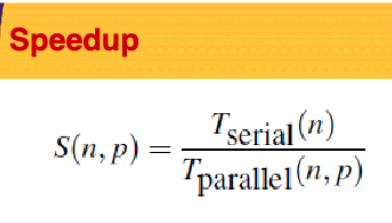
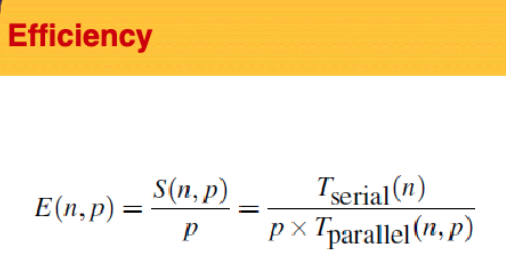
**PHASE 1: Implementation of the parallel algorithm using MPI**

**[-almost all code lines before, so we are including screenshot of only the main function-]**

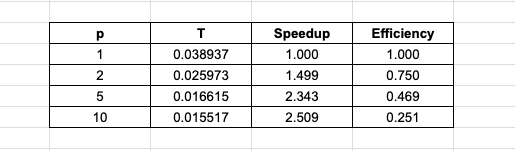


**Compilation and execution of your program on the Magnolia system. Include commands, screenshots showing your account, compilation, execution, and performance metrics (runtime and speedup)**



For evaluation, we calculated the speedup and efficiency according to the following formulas:  
 

And got the following results:



--------- ------------ ------------ ----------- ------------- ------------- ------------ ---------- ---------- --------

**PHASE 2: Implementation of the parallel algorithm using OpenMP**

Two functions that were changed for project phase 2 are Sobel and main() functions, so we are including screenshots for only those two: **A computer screen with text and images

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

**Compilation and execution of your program on the Magnolia system. Include commands, screenshots showing your account, compilation, execution, and performance metrics**

For potential comparison and analysis purposes, we decided to run on the same number of threads: 2, 5, and 10.

Serial:

A screenshot of a computer

Description automatically generated

Parallel:

A computer screen shot of a black screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

For evaluation, we calculated the speedup and efficiency where we got the following results:

A table with numbers and text

Description automatically generated

--------- ------------ ------------ ----------- ------------- ------------- ------------ ---------- ---------- --------

**PHASE 3: Implementation of the parallel algorithm using CUDA**

For CUDA implementation, we have made significant changes in the main () function as well as added a new function \_\_global\_\_ void sobelFilter() with parameters, which implements CUDA programming:

**A computer screen with text and images

Description automatically generated**

**A computer screen shot of a program code

Description automatically generated**

**Compilation and execution of your program on the Magnolia system. Include commands, screenshots showing your account, compilation, execution, and performance metrics**

For potential comparison and analysis purposes, we decided to run on the same number of threads: 2, 5, and 10.

Serial:

**A screen shot of a computer

Description automatically generated**

Parallel:

**A screen shot of a computer program

Description automatically generated**

For evaluation, we calculated the same attributes, which are speedup and efficiency where we got the following results:

**A table with numbers and a number

Description automatically generated**

**Comparison of output images obtained from the serial and parallel programs.**

**Performance analysis and discussion of the parallel implementation, including tables and figures.**

**A table with numbers and letters

Description automatically generated** Setting up three result tables for parallel runs using MPI OpenMP, and CUDA, we can see that the serial as well as parallel results for OpenMP were much better compared to the other two. Although, focusing on CUDA implementation specifically, we can see that with the increase in the number processes, the time slightly decreased. These results are solely based on our code and CUDA implementation, so the conclusions may be totally different, if we changed the code and improved the efficiency of using CUDA code.

**Conclusion**

In this phase of the project, we were able to learn how to implement CUDA for parallel computing on the Soble image segmentation. We showed the performance analysis in the form of the table that includes time it took for different number of processes to run with parallel, their speedup as well as efficiency compared to the serial code. We do acknowledge that the code design for CUDA could be improved to get much better results, however, considering our values for speedup and efficiency and comparing it to the results of our MPI and OpenMP code snippets for the same number of threads, we can say that our overall outcome for phase III was successful.