Hybrid Implementation of Sobel Edge Detection

By Andre Shapiro

Final program description

- This program uses MPI to deploy between 1 and N processes across an equal amount of nodes.
- Then it uses OpenMP to speed up Sobel edge detection across 4 threads.

The value of your solution

- Simulates a parallel file system by using the /tmp directory.
- Simulates a real-life use case in large movie studios
- MPI allows splitting the work across processes so I/O can be sped up
- OpenMP allows the sobel calculation to be sped up too

numerical methods and algorithms

- The numerical method used in this program is convolution
- to find the edge, we have to use convolute every 3*3 set of pixels by an X and Y filter.
- resulting numbers can be summed up, by squaring both sums, adding them
- taking the square root of them, we can determine the magnitude.
- If the magnitude exceeds a predetermined value, we can assume an edge exists.

parallel programming methods

- MPI in order to split the work between many nodes(deals with I/O bottleneck)
- OpenMP to speed up the edge detection(deals with data transfer bottleneck)

Command to run

- All code located inside of “sobel_mpi_scalable.cpp”
- To Compile
  - Make
- To run Single-threaded
  - Change the THREADS value to 1
  - Set read_file_name folder_name to sg_input_images
  - Make sure /tmp/sg_input_images exists and its populated, make sure /tmp/output_files exists
  - mpirun mpirun -n 1 -ppn 1 -f c2_hosts ./sobel_MPI_scalable
- To Run OpenMP
  - Change the THREADS value to 2 or 4
  - Set read_file_name folder_name to sg_input_images
  - Make sure /tmp/sg_input_images exists and its populated, make sure /tmp/output_files exists through all
  - mpirun mpirun -n 1 -ppn 1 -f c2_hosts ./sobel_MPI_scalable
- To run MPI
  - Change the THREADS value to 1
  - Set read_file_name folder_name to mp_input_images
  - Make sure /tmp/mp_input_images exists and its populated, make sure /tmp/output_files exists through all
  - mpirun mpirun -n 1(number_of_processes) -ppn 1 -f c2_hosts ./sobel_MPI_scalable
- To run hybrid
  - Change the THREADS value to 2 or 4
  - Set read_file_name folder_name to mp_input_images
  - Make sure /tmp/mp_input_images exists and its populated, make sure /tmp/output_files exists through all
  - mpirun mpirun -n 1(number_of_processes) -ppn 1 -f c2_hosts ./sobel_MPI_scalable

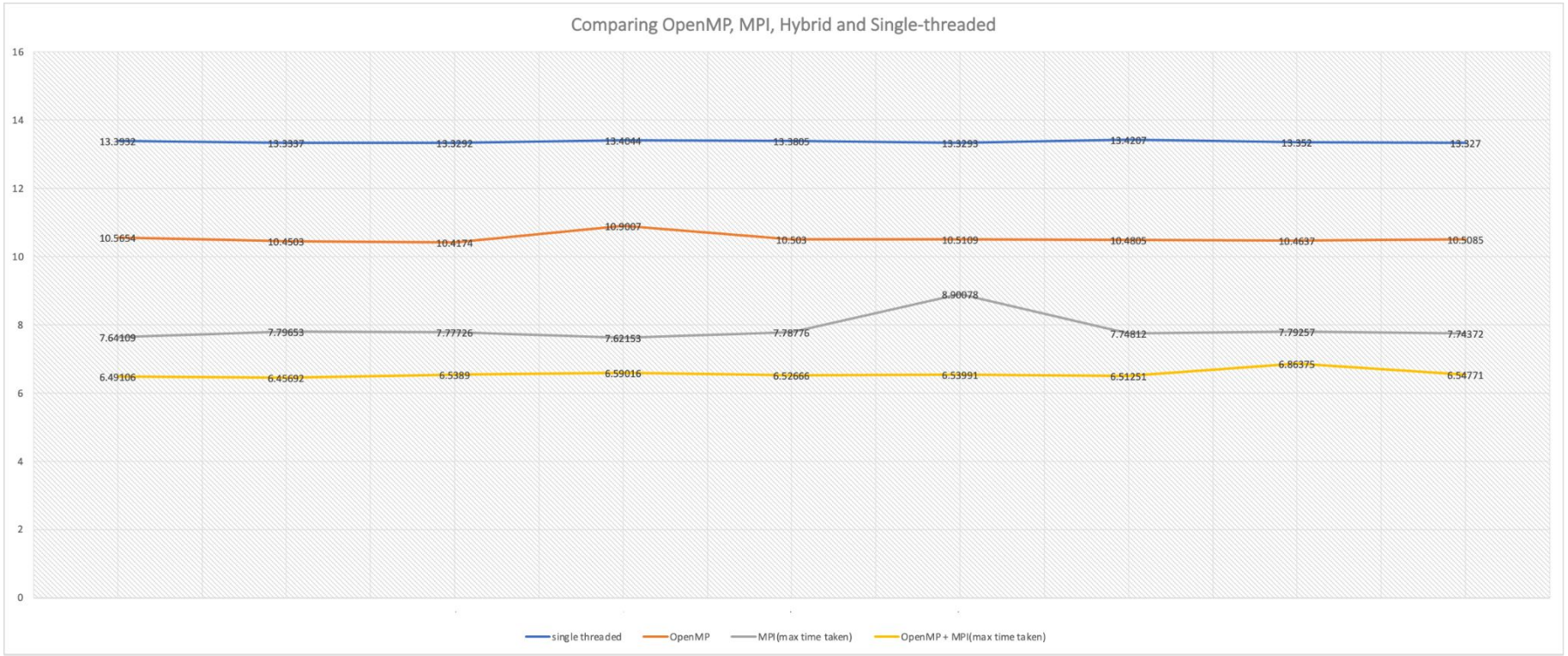
Pseudo code

Read image

- Traverse every 3*3 matrix in the image
  - Use Xfiler and Yfiler to calculate the sum of the 2 resulting matrices
  - Calculate the magnitude by squaring the resulting numbers, adding them, and then taking the square root
- Print the magnitude(can normalize or write past a threshold value)

Print output image using magnitude

Timing differences



Timing

OpenMP: A process doing 20 image processing iterations would take 10.53 with 4 threads

MPI : 5 processes doing 4 images: average of 4.23 seconds, $k frame : A process took an average of 1.08 seconds,

Hybrid: 5 processes doing 4 images with 4 threads : average of 3.5113 seconds
4k frame: A process took an average of 0.87 seconds

Speed Up

Parallel Portion : (calculated time running the function using one thread/total time)

= 0.3656 or %36.56

Number of cores = 4

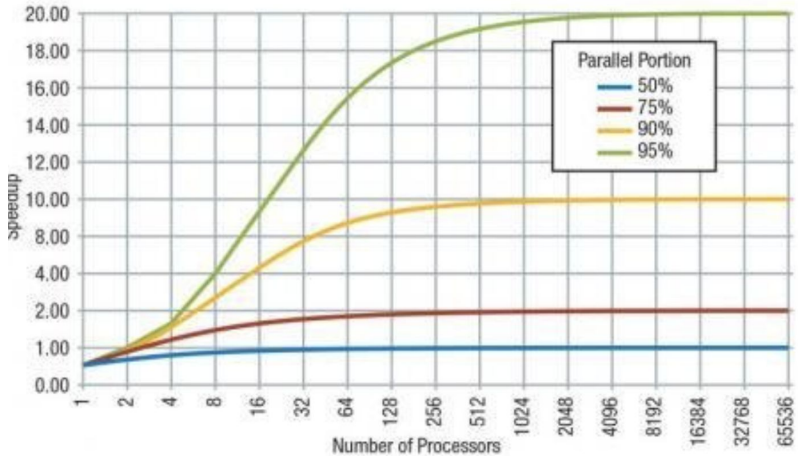
Number of processes = N or 5

parallel OpenMP: (1-0.3656) = 0.6344

parallel MPI: (1- 0.806) = 0.194

Parallel hybrid: = 0.194

speedup



OpenMP = 1/((1-0.3656) + (0.3656/4)) = 1.377

MPI = 1/((1- 0.806) + (0.806/5)) = 2.8

Hybrid = 1/((1-0.806) + (0.806/(4*5))) = 2.8 or 4.26

NOT SURE ABOUT HYBRID

The speedup values seem to match graph for Amdahl's law

Verify Results

Gimp

Me

Gimp

Me

Compromises

- In order to optimize efficiency sobel was modified
  - Usually we keep track of the brightest edge and normalize the x and y edge matrices by that number
  - To use OpenMP I sacrificed that ability, instead we set a threshold and set the brightness to 255 if an edge is detected
- Having a “master” thread was not an option due to the limitations in data transfer
  - At first I attempted to distribute an image across many processes, but sending and receiving data destroyed any speed up that was gained
- We do not have a parallel file system
  - This was simulated using the /tmp directory
  - Meaning there is a lot of set up before work can be performed
- PGM restriction
  - Because I did not attempt to solve this problem, the program can only take in and spit out PGMs, meaning that no colors are ever displayed in edges

Thank you for sitting through this :)