CS 358
Introduction to Parallel Computing
Spring 2025

Project #02 (v1.3)

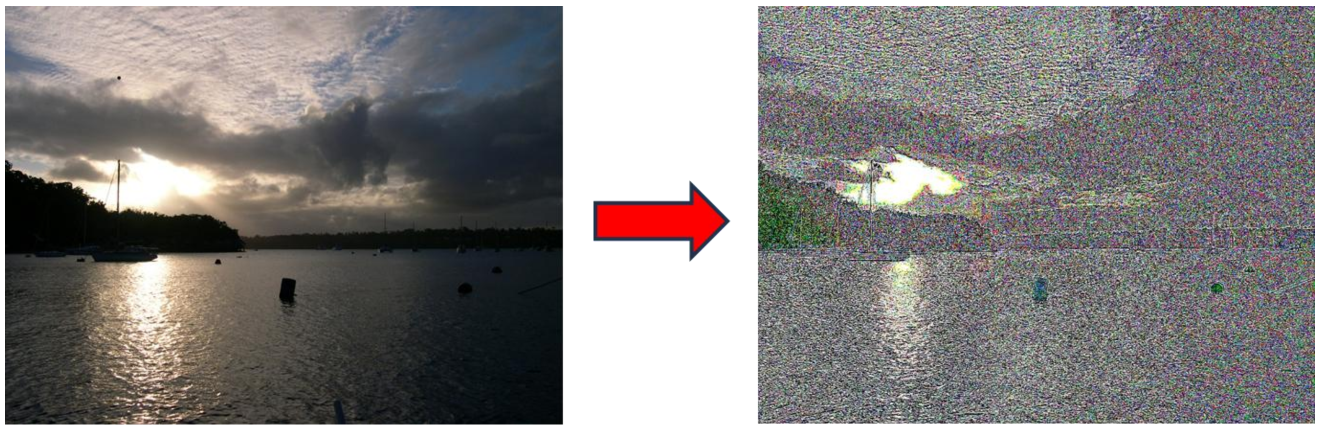
Assignment: Contrast-stretching (base algorithm)
Submission: via Gradescope (unlimited submissions)
Policy: individual work only, late work is accepted
Complete By: Monday May 19th @ 11:59pm CST

Late submissions: see syllabus for late policy… No submissions accepted after Wed 05/21 @ 11:59pm

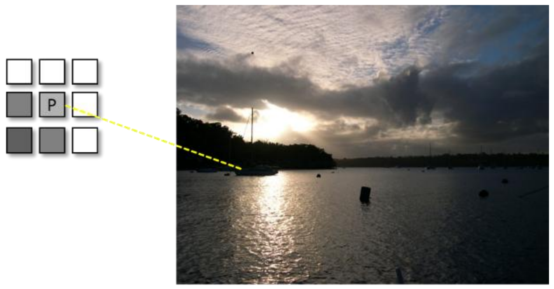
Pre-requisites: Discussion of contrast-stretching in class (PPT / Recording)

Overview

Contrast-stretching is an algorithm that attempts to slowly increase the contrast within an image, with the goal of surfacing objects that are initially hard to detect. Here's the idea, though the image on the right will be hard to see given the copy-paste nature of writing this document:



We discussed the algorithm in class (see links in "pre-requisites" above). The algorithm falls into the class of **nearest-neighbor** algorithms since a pixel is "stretched" based on its neighbors ----> Your assignment is to modify the provided implementation of a basic contrast-stretching algorithm to run on a distributed machine using MPI.



Sunset.bmp

Your implementation must work for N > 0 processes, so it may be true that the # of processes does not divide evenly into the # of rows (or columns) of the image. You must also assume that only one process can read/write the image file; that process (typically myRank 0) is responsible for opening the image file, distributing the image to the other processes, working in parallel with the others, and then collecting the results and writing the stretched image. The other processes have no access to the image file, which is generally true on a distributed system.

Here's the contrast-stretching algorithm in pseudo-code:

```
step = 1;

while (step <= NumSteps)
{
  foreach (non-boundary row r of M)   /* for each pixel, stretch... */
    foreach (non-boundary column c of M)
    {
       M'[r][c] += stretch(M[r-1][c-1], M[r-1][c], M[r-1][c+1],
                           M[r][c-1],   M[r][c],   M[r][c+1],
                           M[r+1][c-1], M[r+1][c], M[r+1][c+1]);
    }

  swap(M, M');
  step++;
}
```

Here in project 02 the focus is getting the basic mechanics of MPI working, i.e. using *MPI_Send* and *MPI_Recv* to do the following:

1. *Distribute the image*
2. *Exchange boundary rows during each step of the algorithm*
3. *Collect the resulting image*

You are not required to optimize your solution, e.g. it's fine to send the entire image to every worker process. You are also not required to use MPI's more advanced collective operations, since the plan is to use those in project 03 (along with implementing a slightly more sophisticated, convergence-based version of the contrast-stretching algorithm). The goal again in project 02 is to gain some experience with MPI, and obtain a working solution in the SPMD model.

However, your program should be "correct" and "safe", where safety was discussed in class (see day 10, PPT / Recording). Recall that replacing *MPI_Send* with *MPI_Ssend* and then running is a quick way to detect many "unsafe" MPI programs. Another way is to use MPI's collective operations.

Downloading the provided files

A number of files are being provided in this dropbox folder. Download, extract, and move the folders into your class docker image (the base image we've been using has MPI installed). If you are not using docker, you need to install OpenMPI on your system. Three folders are being provided:

cs-mpi	**cs-openmp**	**cs-seq**

The "seq" folder contains a working, sequential implementation of the program. To run the program, open a terminal window, run docker, cd into the folder, and do the following to run 10 steps of the contrast-stretching algorithm and save the result in "temp.bmp":

```
make
./cs sunset.bmp temp.bmp 10
diff temp.bmp stretched-10.bmp
```

The last line is a correctness check to confirm the program output matches the expected results. There are two result files: "stretched-10.bmp" and "stretched-75.bmp". Here is the contents of the "seq" folder:

app.h	**main.cpp**	**stretched-10.bmp**
bitmap.cpp	**makefile**	**stretched-75.bmp**
cs.cpp	**matrix.h**	**sunset.bmp**

Focus on "main.cpp" and "cs.cpp"; feel free to experiment with your own bitmap files if you want, the code in "bitmap.cpp" should read/write any valid .bmp file.

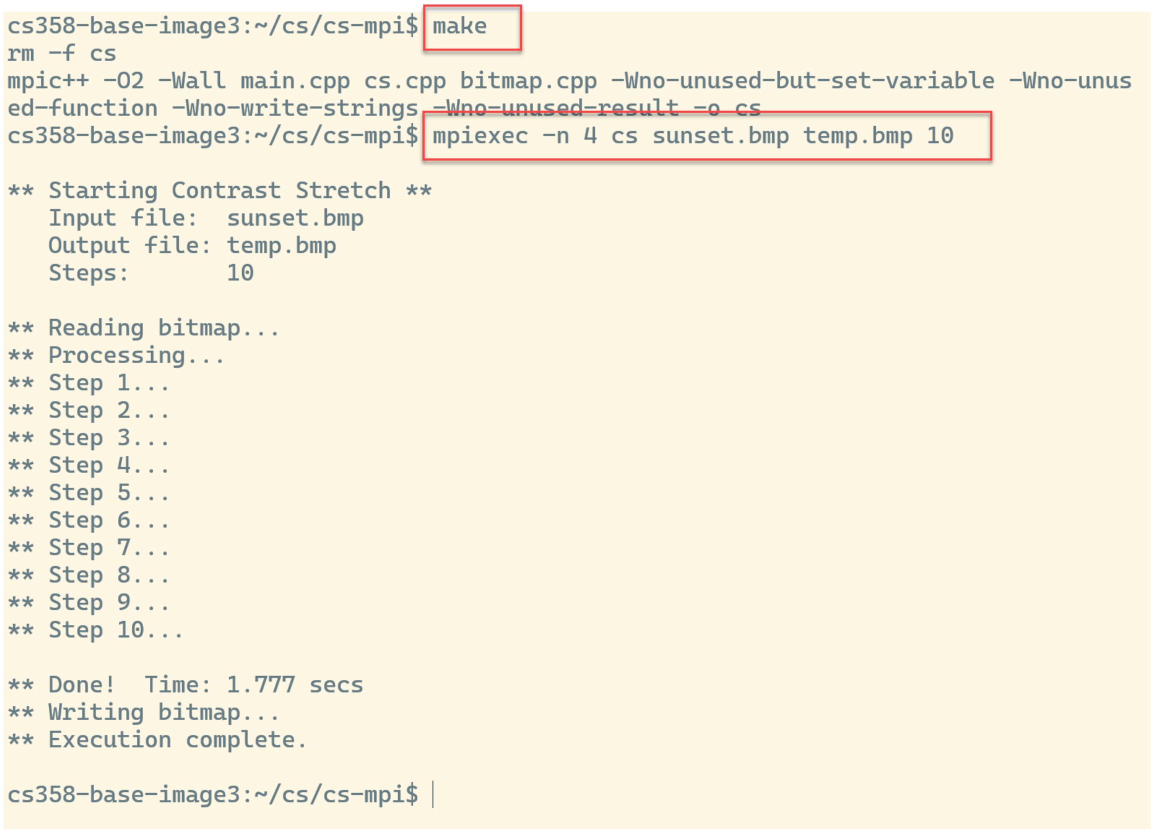
The "openmp" folder contains the same code with 1 OpenMP pragma to parallelize each step of the algorithm. See "cs.cpp". The makefile is also different, providing the command-line option "-fopenmp" to link with the OpenMP library. The OpenMP version sets an upper-bound on our performance goal; in a perfect world, the MPI version will perform as well as the OpenMP version.

The "mpi" folder contains a copy of the sequential code --- your assignment is to modify this version to implement the contrast-stretching algorithm using MPI. The makefile has changed since you compile and run differently using MPI. Use "make" to build using the *mpic++* compiler, and use "make run" to use *mpiexec* to launch and run the program.

<< continued on next page >>

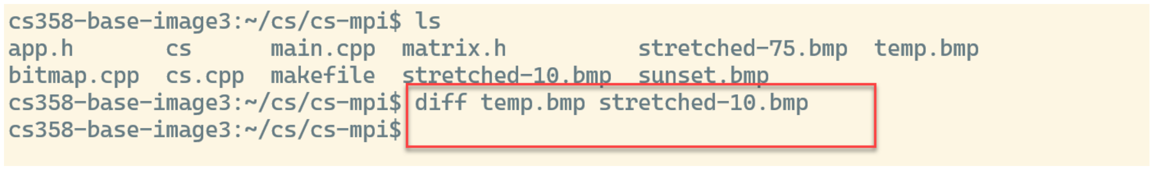
A few implementation details…

Your MPI program should produce the same output as the provided sequential version. Only the main process (myRank == 0) should output so that it appears only once:



```
cs358-base-image3:~/cs/cs-mpi$ make
rm -f cs
mpic++ -O2 -Wall main.cpp cs.cpp bitmap.cpp -Wno-unused-but-set-variable -Wno-unus
ed-function -Wno-write-strings -Wno-unused-result -o cs
cs358-base-image3:~/cs/cs-mpi$ mpiexec -n 4 cs sunset.bmp temp.bmp 10

** Starting Contrast Stretch **
   Input file:  sunset.bmp
   Output file: temp.bmp
   Steps:       10

** Reading bitmap...
** Processing...
** Step 1...
** Step 2...
** Step 3...
** Step 4...
** Step 5...
** Step 6...
** Step 7...
** Step 8...
** Step 9...
** Step 10...

** Done!  Time: 1.777 secs
** Writing bitmap...
** Execution complete.

cs358-base-image3:~/cs/cs-mpi$ 
```

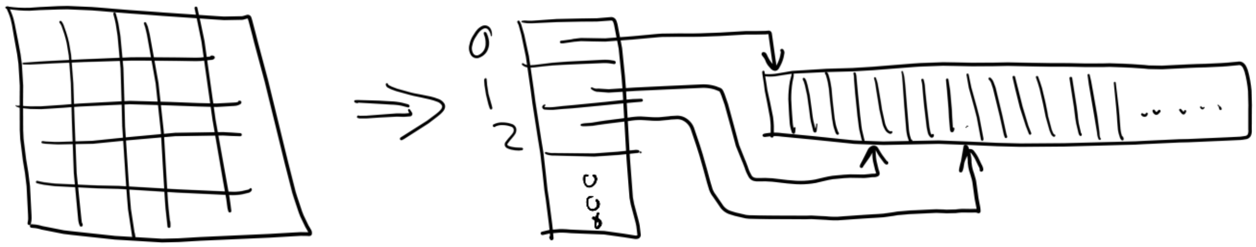
After you run, MAKE SURE YOUR PROGRAM WORKS by comparing your resulting bitmap file to the provided "stretched" files. For example, if you run for 10 steps, use **diff** to compare your output ("temp.bmp" in the screenshot above) to "stretched-10.bmp" and make sure diff reports NO differences:



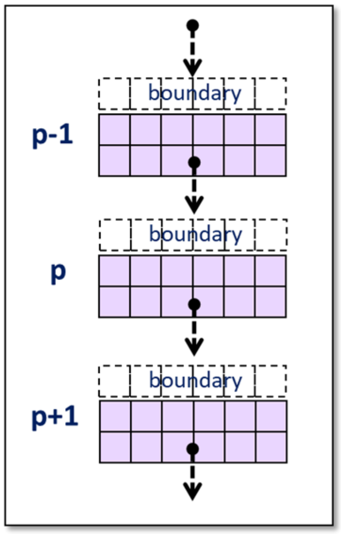
```
cs358-base-image3:~/cs/cs-mpi$ ls
app.h       cs      main.cpp   matrix.h          stretched-75.bmp  temp.bmp
bitmap.cpp  cs.cpp  makefile   stretched-10.bmp  sunset.bmp
cs358-base-image3:~/cs/cs-mpi$ diff temp.bmp stretched-10.bmp
cs358-base-image3:~/cs/cs-mpi$
```

<< continued on next page >>

The image is stored as a matrix, but a quick look at "matrix.h" will reveal that the 2D matrix is actually stored as a 1D array. This will make sending and receiving rows of data much easier:



Divide up the work by row, so that each process gets the same amount of rows (any extra, left-over rows are typically processed by the main process). Since boundary rows have to be communicated to neighboring processes, allocate 2 extra rows in the local matrix for these boundary or "ghost" rows:



Grading and Electronic Submission

Programs will be collected using Gradescope. We will test for correct and safety, but it will be your responsibility once again to confirm the output from Gradescope in terms of speedup. We don't expect linear speedup, but as we increase the # of processes from 1, 2, 4 and 8, your program must run faster. And it must use MPI following the restrictions outlined in this handout; failure to honor the restrictions (e.g. only one process can read/write the image file) will result in a score of 0/100.

Should you comment your program? Yes. How well? Parallel code is hard to understand, so comment what you are doing because the TAs ***are*** going to read your program. And if they find it hard to read / understand, you will lose points. Think 80/20: 80 points for a correct and safe approach, 20 points for readability and understanding.

Academic Conduct Policy

Northwestern publishes a basic guide to academic integrity, which can be found here. In summary, here are NU's eight cardinal rules of academic integrity:

1. *Know your rights*
2. *Acknowledge your sources*
3. *Protect your work*
4. *Avoid suspicion*
5. *Do you own work*
6. *Never falsify a record or permit another person to do so*
7. *Never fabricate data, citations, or experimental results*
8. *Always tell the truth when discussing your work with your instructor*

School policies and more information can be found on NU's academic integrity website. With regards to this class, unless stated otherwise, all work submitted for grading *must* be done individually. While we encourage you to talk and learn from the course staff, peers, and AI systems, this interaction must be high-level with regards to all work submitted for grading. This means you cannot work in teams, you cannot work side-by-side, you cannot submit someone else's work --- nor the work of an AI system --- as your own. In short, you are encouraged to talk to your peers, the staff, and AI systems as a learning exercise, but do not copy and edit code from your peers, the staff, or an AI system. When taking quizzes and exams, no outside source of information is allowed --- no notes, no phones, no laptops.

Examples of what is not allowed? Downloading work from a github repository and submitting it as your own, whether partial or complete. Downloading answers from StackOverflow and submitting them as your own. Emailing your work to another student, or receiving work from another student. Sharing your screen with another student so they can see your work (and potentially submit it as their own). Participating in a screen share and taking screenshots or photos of someone else's work, and then using that as a guide to submit your own work. Copying answers posted to Piazza, making a few simple changes, and then submitting as your own work. Allowing someone else to write / type the answer for you. Using ChatGPT, or CoPilot, or other AI systems, to generate code that you simply copy-paste-edit. Using your phone during a quiz or exam.

Okay, so what is allowed? Talking to the instructor or course staff, and getting insights --- but not direct answers --- in how to solve the problem. Talking to other students about the problem, using diagrams / natural language / pseudo-code to convey ideas. Searching the internet for guidance, and using that guidance to help you form your own solution. Showing your work to an AI system and asking the AI for flaws in your approach or ways to improve your code. If you do receive help / guidance, it's always best to cite your source by name or URL, or by providing the prompt used with an AI system.

What are the penalties for an academic violation? You can expect the minimum penalty to be a 0 on the assignment and a letter drop in your final grade; you will also become ineligible to serve as a Peer Mentor for the department. Matters of academic misconduct are forwarded to the Dean of your college, and recorded internally to identify repeat offenders. The Dean may decide to increase the penalty based on the circumstances and past history; students have been suspended and even expelled for academic misconduct.