CS 358
Introduction to Parallel Computing
Spring 2025

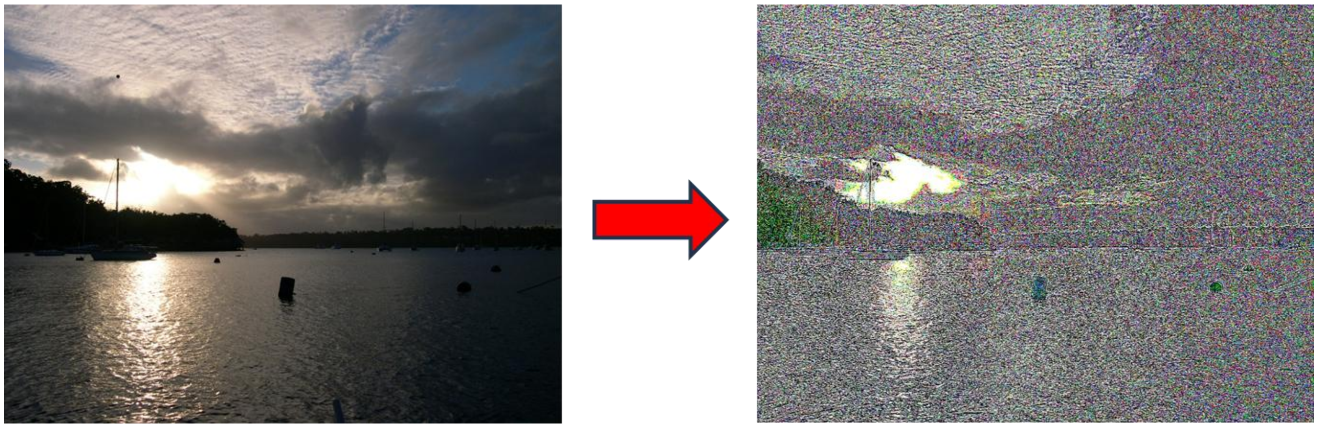
Project #03 (v1.0)

Assignment: Contrast-stretching w/ Convergence Testing
Submission: via Gradescope, Canvas and Quest
Policy: individual work only, late work is accepted
Complete By: Friday June 6th @ 11:59pm CST
Late submissions: see syllabus for late policy… No submissions accepted after Sun 06/08 @ 11:59pm

Pre-requisites: Project 02

Overview

In project 02 we implemented an MPI-based **contrast-stretching** algorithm to run on distributed-memory machines:



Here in project 03 we're going to make the following extensions:

1. *Use MPI's collective operations **Bcast**, **Scatter**, **Gather**, and **SendRecv** to build a more efficient version.*
2. *Add **convergence testing** to the algorithm to stop early if the image has converged.*
3. *Run and test on Northwestern's **Quest** HPC cluster.*

Our solution to project 02 will be provided if you would prefer to build upon our solution here in project 03.

Step 01: Quest account?

Can you login to quest.northwestern.edu? Check now so that if you don't have an account, there's time to create an account before the due date. Open a terminal window and try to connect using ssh, your netid, and your Northwestern password:

```
ssh YOUR-NETID@quest.northwestern.edu
```

If you can login, all is well, you can exit and close the terminal window. If you CANNOT login, post privately to Piazza and we'll ask the quest support team to investigate.

Step 02: which solution to build upon?

Decide which solution you want to build upon…

If you plan to build upon YOUR solution, in "main.cpp" you'll need to #include <unistd.h>, and then at the start of main( ) add a call to gethostname( ) and cout process # and machine's hostname. Here's the code to add to main():

```
char host[128];

gethostname(host, sizeof(host)/sizeof(host[0]));  // machine we are running on:

cout << "process " << myRank
     << " starting on node '" << host << "'..."
     << endl;

cout.flush();
```

When running on quest, this code will identify the compute node each process is running on.

If you plan to build upon OUR solution, a .zip file is available in this dropbox folder. Download, extract, and move the folder into your class docker image (the base image we've been using has MPI installed; if you are not using docker, make sure OpenMPI is installed on your system). A folder named "cs2" is being provided with two sub-folders:

mpi **seq**

The "seq" folder contains a working, sequential implementation of the program (with support for convergence testing, to be discussed in later step) --- ignore this folder for now. The "mpi" folder contains a solution to project 02. Use **make** to build, and **make run** to launch 4 processes that run for 10 steps. To confirm the solution is working correctly, use diff:

```
diff temp.tmp stretched-10.bmp
```

Running diff should yield no output --- i.e. no differences.

Step 03: a more efficient MPI program…

As needed, rewrite the project 02 solution to be more efficient. In particular, you MUST do the following to receive credit for this project:

1. *Use **MPI_Bcast** to send out image parameters.*
2. *Use **MPI_Scatter** to distribute image.*
3. *Use **MPI_Sendrecv** to exchange ghost rows during contrast stretching.*
4. *Use **MPI_Gather** to recieve image*

We talked about these in class, with example calls, so review the lecture notes. You CANNOT use any other MPI functions in the steps above, the idea here in project 03 is to build an efficient solution for contrast stretching with MPI. Failure to meet the above 4 requirements is a 0/100 on the project --- especially since our provided solution is in a format already amenable to the required changes. For example: you CANNOT use Bcast or Send/Recv to distribute the image, you must use MPI_Scatter so that each process gets an equal-sized chunk. No more sending the entire image around.

After you have made the above changes, test, test, test. Be sure to test the following scenarios, using diff to confirm the output matches the expected output:

1. *test with different numbers of processes: -n 1, -n 2, -n 4*
2. *test with odd # of steps (13)*
3. *test with left-over rows (-n 7)*

When you're confident the program is working, confirm linear or near-linear speedup. If not, something is wrong…

Step 04: convergence testing

Extend the algorithm by adding support for convergence testing. It turns out that in some cases, the image stops changing --- at some point each step of the algorithm produces the same image over and over again. In other words, the image cannot be "stretched" any further.

It's a simple change to the algorithm. As the image is stretched, you count how many pixels actually change in value. If the # of differences is 0, the image has converged, and the algorithm stops. Here's the SEQUENTIAL version of the algorithm with convergence testing:

```
int  step = 1;
bool converged = false;

while (step <= steps && !converged) {

  int diffs = 0;
```

```
    // perform one step of algorithm, counting # of pixels changes
    for (each row) {
      for (each col) {
        diffs += stretch_one_pixel(image2, image, row, basecol);
      }
    }

    cout << "** Step " << step << "..." << endl;
    cout << "   Diff " << diffs << endl;

    converged = (diffs == 0);  // have we converged?
  }
```

Modify your contrast stretching algorithm as shown above, including both cout statements (which should only be output by the main process). You'll also need to modify the function **stretch_one_pixel( )** in "cs.cpp" as follows:

1. *Change the return type from void to int*
2. *Declare a local variable named "changes" and initialize to 0*
3. *After the update of each pixel value (Green, Blue, and Red), check to see if the value has changed, and if so, update "changes":*

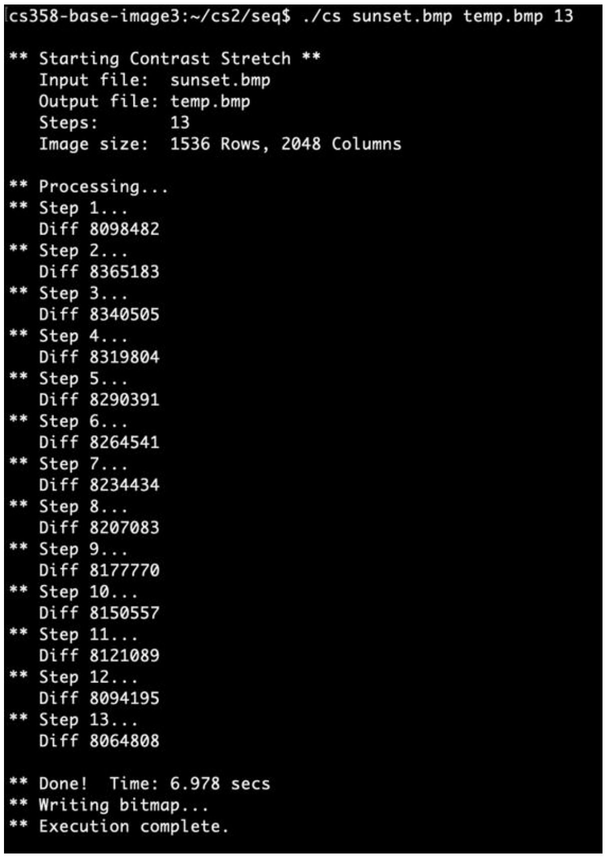
```
   // did value change?
   if (image2[baserow][basecol] != image[baserow][basecol])
     changes++;
```

4. *Return "changes" from the function --- the function is now returning either 0, 1, 2, or 3*

Of course, the interesting aspect of implementing this algorithm on a distriburted memory machine is that "diffs" is not actually global --- each process will have its own local "diffs" variable. <u>A single, global diffs value must be computed before a decision can be made as to whether the algorithm has converged or not</u>. In other words, the algorithm stops IF AND ONLY IF all processes report 0 differences. Otherwise all processes keep running.

There are many ways to compute (and then share) a common diffs value. You are required to take an efficient approach using MPI's collective operations. In particular, (1) use **MPI_Reduce** to compute a global diffs value on the main process, and then (2) have the main process use **MPI_Bcast** to distribute this value out to all the workers so a convergence decision can be made by all.

For testing purposes, here's a screenshot of the sequential version running with convergence testing. Your MPI-based



```
cs358-base-image3:~/cs2/seq$ ./cs sunset.bmp temp.bmp 13

** Starting Contrast Stretch **
   Input file:  sunset.bmp
   Output file: temp.bmp
   Steps:       13
   Image size:  1536 Rows, 2048 Columns

** Processing...
** Step 1...
   Diff 8098482
** Step 2...
   Diff 8365183
** Step 3...
   Diff 8340505
** Step 4...
   Diff 8319804
** Step 5...
   Diff 8290391
** Step 6...
   Diff 8264541
** Step 7...
   Diff 8234434
** Step 8...
   Diff 8207083
** Step 9...
   Diff 8177770
** Step 10...
   Diff 8150557
** Step 11...
   Diff 8121089
** Step 12...
   Diff 8094195
** Step 13...
   Diff 8064808

** Done!  Time: 6.978 secs
** Writing bitmap...
** Execution complete.
```

output should match these values. The sequential version is provided along with our solution to project 02; you already have a copy if you downloaded our solution, otherwise you can download the sequential version from dropbox.

Step 05: execution on quest

At this point you should have a working MPI-based version of contrast stretching with convergence testing. Well done! The last two steps are to (a) run on Quest, and (b) submit to Gradescope. Let's focus on Quest first. Follow this steps to run and submit on quest.northwestern.edu:

1. *Login to quest, create a "cs" directory, cd cs, and then create "mpi" sub-directory*
2. *Upload your program and data files to your ~/cs/mpi directory on quest using sftp*
3. *Change into ~/cs/mpi directory, copy job scripts:* ***cp /home/jeh0060/scripts/cs/* .***
4. *Submit and run batch scripts using sbatch filename.bash: 1, 2, 4, 8 and 16 nodes*
5. *Use squeue -u <netid> to monitor jobs...*
6. *View the resulting .txt files, confirm correct results and linear / near-linear speedups*
7. *more cs-output-16.txt, take a photo WITH YOUR ID VISIBLE, and upload to Canvas under the assignment "Project 03 – Quest" --- make sure the photo includes your ID. No ID, no grade*
8. *Run this program to submit your files as proof of quest execution: /home/jeh0060/tools/collect **

Step 06: Grading and Electronic Submission

Before submitting to Gradescope, note that you should have already submitted two things: (1) your source code and output files on quest (step 8), and (2) a photo of your 16-node quest output with ID, uploaded to Canvas (step 7).

Final submission and testing will be done via Gradescope, so please submit all source code files to Gradescope under "Project 03". Gradescope will confirm correctness, but it will be your responsibility to confirm the output in terms of linear or near-linear speedup. The TAs will then manually confirm speedup, and then review your program for adherence to the restrictions outlined earlier. Your goal is an autograder score of 10/10; you have unlimited submissions.

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.