Due Date: Friday March 16, 11:59pm

Application:

When you go to the hospital and get an MRI [magnetic resonance imaging] scan, it takes a stack of images as cross-sectional slices of the scanned area. Individually, these images are of very poor quality. They are blurry, noisy, and polluted by artifacts creating from the scan process. Because of this, image processing programs use the wealth of information present in all these images collectively to improve the quality of any individual imaged region. This is done by leveraging the fact that these images are all pictures of the same thing, just taken of different parts of this same object. Thus to improve the quality of any given pixel, all near by pixels can be examined for clues of our targetted pixel's true value.

One of these image processing operations is deblurring, which takes a fuzzy image and produces a clearer image from it. This is the application that has been given to you to parallelize.

The distributed project consists of 5 source files and 2 data files.

The 2 data files contain 3D images of random static that can be used as test platforms. They are not actual scan images, but are useful to pick up algorithmic problems that may otherwise only surface probabilistically., as a result of either race conditions between threads or strange arithmetic corner-cases.

The 5 source files are main.c, containing a driver program that measures your solution's performance, seqDeblur[.h/.c] and ompDeblur[.h/.c]. seqDeblur.c implements a deblurring image pass sequentially. ompDeblur.c currently also implements a deblurring image pass sequentially, and is identical to the seqDeblur case, but you are aiming to correct that. Function prototypes are found in both .h files. The driver will measure the execution time of your OMP_Deblur, and compare it against the sequential SEQ_Deblur.

To initialize your function, OMP_Initialize will be called. To clean up your function, OMP_Finish will be called. Neither Initialize or Finish will count toward your execution time. The only function that will be timed will be OMP_Deblur.

Edit and submit only ompDeblur.c. Grading will be based on the speed up achieved over the stock version. The expected minimum performance improvement, using whatever optimization method that you would like, will be 10-times. This means that your optimized version is expected to run in at most 1/10th the time as the unmodified version. The number of threads is limited to 8. You may NOT change the number of allowable threads in ompDeblur.c

In order to run the program on your own platform, or on a lnxsrv.seas.ucla.edu machine:

1. Unpack the zipped file downloaded from Courseweb

tar xzf openmp_project.tar.gz
This will create a directory openmp_project

2. Compile the program by typing the following commands:

cd openmp_project/
make

3. Run the program by typing:

./deblurTest

4. Each time you modify ompDeblur.c, you need to re-compile the program by typing "make"

While this is appropriate for debugging purposes, this will not be graded. The performance of your optimized code also depends on the load (what else is running) on the system. You can use programs like top to see the load of the system. After top starts press 1 to see the load on each core of the machine.

There are 4 different servers to test your code on:
lnxsrv01
lnxsrv02
lnxsrv03
lnxsrv04
You can also log into lnxsrv and have seasnet perform load balancing for you.

These are shared by all students in the engineering departments, including your fellow CS 33. When multiple students are running their programs the performance of your program may be drastically lower. To avoid problems testing problems START EARLY!

Grading: Grading will be done on a isolated machine similar to lnxsrvXX (01, 02, 03, 04 – they are all similarly spec'd machines) to prevent any other processes from interfering with the timing results. You will receive points proportional to the speed up:

> 10X speed up – 100/100
9X speed up – 90/100
8X speed up – 80/100
7X speed up – 70/100
6X speed up – 60/100
5X speed up – 50/100
4X speed up – 40/100
3X speed up – 30/100
2X speed up – 20/100
< 2X speed up – 0/100

NOTE: If your solution does not pass "CompareResults", you will receive a 0 regardless of your speed up