Parallel Image Filtering

Summary: In this homework, you will be implementing a threaded program that loads an image file, applies a filter to it, and saves it back to disk. You will also practice dealing with threaded algorithms that include both data spliting and data dependency issues.

1 Background

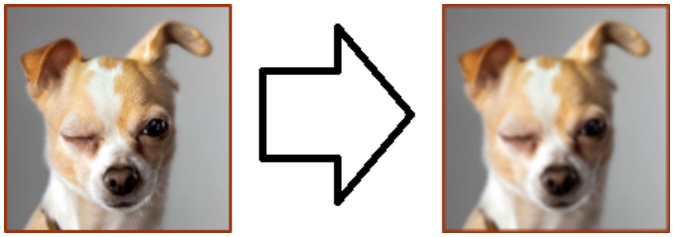


Figure 1: Application of blur algorithm to Wonderbread's resume portrait. (Image from Dr. Filley's TMC110 slide deck.)

In this assignment you will write a program that applies a filter to an image. The image will be loaded from a local BMP file, and then transformed using one of two image filters. The blur filter is shown above as an example. The resulting file will be saved back to a BMP file that can be viewed on the host system. **It is highly suggested that you develop non-multithreaded versions of the filters before beginning to thread them.**

This document is separated into six sections: Background, Requirements, Box Blur Filter, Swiss Cheese Filter, Include Files, and Submission. You have almost finished reading the Background section already. In Requirements, we will discuss what is expected of you in this homework. In the Box Blur Filter and Swiss Cheese Filters sections, we discuss the idea of each of the respective algorithms. In Include Files, we list several files that may provide useful functions for your algorithms, and suggest two ways to handle BMP file IO. Lastly, Submission discusses how your source code should be submitted on Canvas.

2 Requirements [54 points]

Your program needs to implement loading a binary BMP file, applying an image filter to its pixel data (described in the next section), and saving the modified image back into a BMP file. If it helps, you may assume that images will be no larger than 4096 by 4096 pixels (see the MAXIMUM_IMAGE_SIZE constant in the base file). Assume that all BMP files are 24-bit uncompressed files (i.e., compression method is BI_RGB), which have a 14-bit bmp header, a 40-bit dib header, and a variable length pixel array. Regarding threading: it is suggested that you start by hard-coding your program to use four threads to compute the result. Later, extend it to allow an arbitrary number (create and use a constant called THREAD_COUNT).

As a base requirement, your program must compile and run under Xubuntu (or another variant of Ubuntu) 20.04. Your program must also support reading and writing uncompressed 24-bit BMP files. Sample output for for the blur filter is shown on the right side of Figure 1.

Your general approach to parallelizing the algorithms will be to break (decompose) the image into pieces, in parallel compute the filtered result for the pieces, and then combine the images into a final result. **Both**

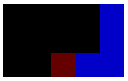
Figure 2: Example decomposition of input. Notice that data will be balanced among threads.

algorithms must break the image into pixel columns as shown below (not rectangles, or color channels). See Figure 2 for an example.

- Specific Requirements:
  - File names and filter selection: Read the input file name, output file name, and image filter from the command line arguments. Assume that users always provides all three, and the files will be valid (they exist and are BMPs). Your program must support being used as: *"./LastNameFilters -i in.bmp -o out.bmp -f b"*. For the filter argument, 'b' stands for blur filter and 'c' stands for cheese filter. [3 Points]
  - Blur filter filter: [25 points total]
    - Apply the box blur algorithm to each pixel in the data. [4 points]
    - Compute the box blur algorithm in parallel with pthreads. It is suggested that you divide work into columns. Work must be evenly distributed over a number of threads defined by a constant called THREAD_COUNT. You may assume that is less than the image's smallest dimension, but we will pick the specific value (could be anything) when we compile/grade your program. [10 points]
    - Each thread must use an independent memory allocation of a minimal size. For example: if you have an image that is 64x64 pixels and it is be run with two threads, you will be creating two threads where each processes a 32x64 region. Your program needs to perform allocations for each of the sub-regions, and pass it to the appropriate thread. Note that sub-regions will not be exactly 32x64 in size - they will be slightly larger. Add exactly the amount of padding to the region each thread gets in order for the filter to be computed correctly. (So if a filter requires information from each of the adjacent pixels, then a sub-region would look more like 33x64 to include an extra column of pixels to compute column 32.) [11 points]
  - Swiss cheese filter filter: [26 points]
    - Apply the Swiss cheese filter to the data. (Hint: a hole can be drawn by computing the distance between a pixel and an origin point. If the pixel is with the hole's radius, set it to black.) [10 points]
    - Compute the Swiss cheese filter in parallel with pthreads. Each thread must only know about holes that it has to render, and only render the portion of that hole within that column. Same details apply as for box blur. See Figure 5 for a visualization. (Note: a single hole may be spread across multiple threads if it lies on a column border. This is a data splitting issue.) [10 points]
    - Each thread must use a independent memory allocation of a minimal size. Same details apply as for box blur (but you won't need padding). [6 points].

3 Box Blur Filter

The first filter we will be applying in this assignment is called a box blur. It takes a specific neighborhood of pixels and processes it (this is a so-called *stencil* operation). A given output pixel is computed as the average of itself and each of its neighbors. This is an example of data dependency. We will use a neighborhood size of 3x3. In the following example, we are looking at a pixel (the 100, 0, 0 one) at the bottom of an image. There are 6 valid pixels in the neighbor, so we add them and divide by 6 to produce the output pixel for that position in the output image.



(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 200)
(0, 0, 0)	(0, 0, 0)	(0 ,0, 0)	(0, 0, 0)	(0, 0, 200)
(0, 0, 0)	(0, 0, 0)	(100, 0, 0)	(0, 0, 200)	(0, 0, 200)

$$\rightarrow \frac{(0,0,0)+(0,0,0)+(0,0,0)+(0,0,0)+(100,0,0)+(0,0,200)}{6} = (16,\ 0,\ 33)$$

Figure 3: Computing blur result for pixel at (2, 2) in an image.

4 Swiss Cheese Filter

This filter is designed to make the input image appear to be a piece of Swiss cheese. The Swiss cheese filter has two steps:

1. Tint the image (at the pixel level) towards being slightly yellow. The exact tint is left as a design choice - you may use any method (or RGB shift) provided that it results in an output image that is noticeably more yellow (or buttery yellow).

2. Randomly draw black circles in the image. See Figure 4 for the base concept of adding holes, and Figure 5 for the overall process described earlier. Holes should be uniformly distributed along the x- and y-axis. Holes must be circles. The radius of the holes should be computed in a such a way that average sized holes are most common, and then smaller or larger holes are less common. The average radius in pixels of a hole should be 8% of the smallest side of the input image (e.g., if an image is 130 pixels, then the average radius is 10). The number of holes should also be 8% of the smallest side (e.g., if an image is 130 pixels, then there will be ~10 holes).

Figure 4: Cheesebread example. Holes only, no tinting. (Exact results will differ.)

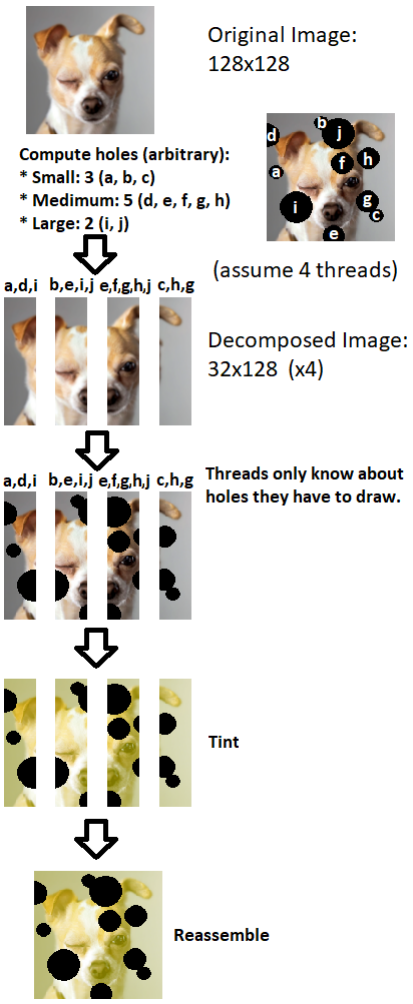


Figure 5: Overall cheesebread algorithm process. (Exact results will differ.)

5 Include Files

To complete this assignment, you may find the following include files useful:

- *stdio.h*: Defines standard IO functions.
- *stdlib.h*: Defines memory allocation functions.
  - *int rand()* - returns a number between 0 and RAND_MAX.
  - *void srand(int seed)* - seeds the random number generator.
- *pthread.h*: Defines functionality for manipulating threads.
  - Useful functions:
    - *int pthread_attr_init(pthread_attr_t *attr)* - Populates a pthreads attribute struct (pthread_attr_t) with default settings.
    - *int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg)* - Creates a new thread with the id of *thread*, attributes specified by *attr*, and some data *arg*.

        * *int pthread_join(pthread_t thread, void **value_ptr)* - Blocks until the specified pthread_t *thread* has terminated.
    - Useful types:
        * *pthread_attr_t* - Contains attributes for a pthread thread.
        * *pthread_t* - Contains a pthread thread id.

- *math.h*: Defines additional functionality for manipulating strings.
    - Useful functions:
        * *double sqrt(double x)* - Takes the square root of a number.

Note that this assignment does not require mutexes or other locks. If you want to include any other files, please check with the instructor or TA before doing so.

5.1 Loading Binary Files

It is not expected that you implement a BMP reader/writing for this assignment, instead you should use an already existing one. For loading the BMP files, you have two options:

1. Reuse your own personal code from the SER334 CP3 programming assignment.
2. You may use the object file that we created, and which contains an already implemented BMP reader/writer. (Think of it as a package in Java.) An object file is a binary representation of a source file, used prior to linking it to other resources (e.g., standard libraries) to create a final executable file. **In order to use the object file, you must be under the VM setup described at beginning of the course. Specifically, (x)Ubuntu 20.04 with 64-bit gcc.** Using the library has three steps:
    (a) Add BmpProcessor.o, PixelProcessor.h, BmpProcessor.h to your source code folder. Note that the read/write functions provided already handle padding and the pixel array will contain only pixels, no padding.
    (b) Include the BmpProcessor.h header in your main file.
    (c) When you compile your project, you will need to set up your linker to include the object file (BmpProcessor.o), otherwise if you call the functions from the For example, if you're using GCC and your main file is called LastNameFilters.c, you will run the command "gcc LastNameFilters.c BmpProcessor.o -o LastNameFilters" to compiler your .c file, link it with the .o, and produce the LastNameFilters final executable. LastNameFilters can then be run using commands such as "*./LastNameFilters -i wonderbread.bmp -o blurrybread.bmp -f b*".

5.2 Linking with External Libraries (pthreads)

By default, including a header file into your project only adds the code associated with that file to your project. For files like stdio.h or stdlib.h, which are self contained, that is enough. However, for other include files like pthread.h or math.h, we need to make sure that our object files (produced from our .c) is also linked with the library object file that supports that functionality. In GCC, we will do this with the "-lm" argument for math.h and "-pthread" argument for pthread.h.

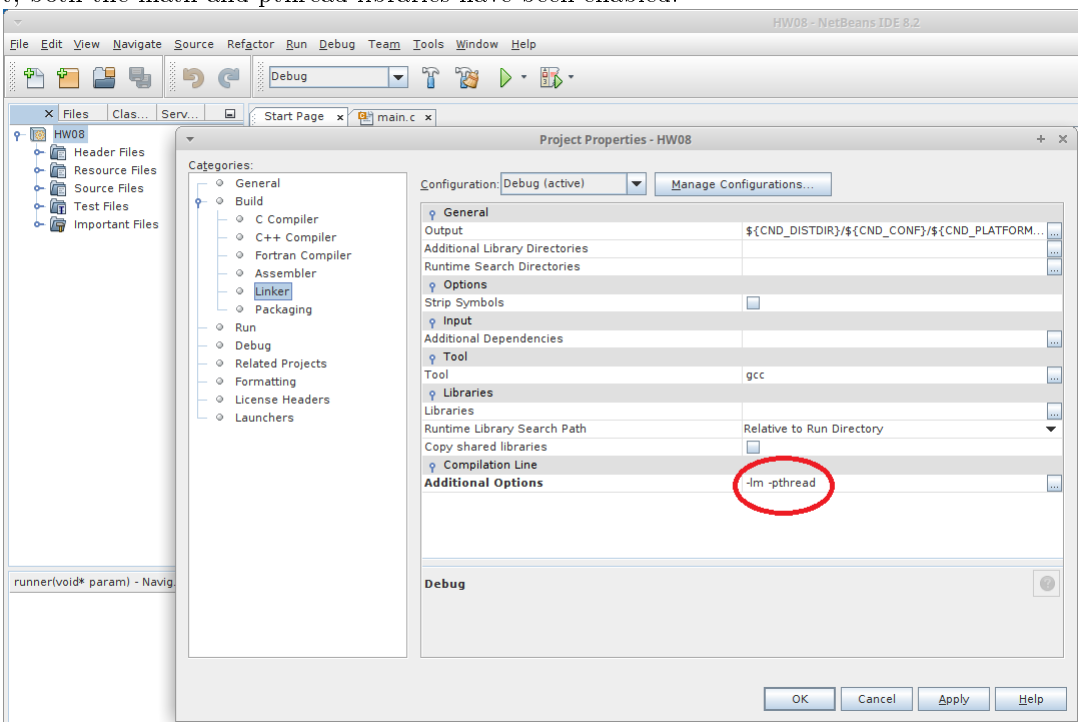
If you are using the command-line to compile, or a makefile, then these arguments should be appended to the end of your GCC command. For example:

```
gcc -c somefile.c -lm -pthread
```

When using an IDE the steps will be different depending on which IDE you are using. **CLion:** will automatically include the math library but requires adding

```
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -pthread")
```

to the CMakeLists file. **NetBeans:** The first step is to open the project in Netbeans. Right click the title of the project and select Properties. This will open up a new window called Project Properties. Select the Linker page from the left categories menu. At the bottom is a Additional Options line, which you can edit or open with the ellipsis button. There you can add any arguments that you need. In the sample screen shot, both the math and pthread libraries have been enabled.



6 Submission

The submission for this assignment has one part: a source code submission. The file should be attached to the homework submission link on Canvas.

Writeup: For this assignment, no write up is required.

Source Code: Please name your main class as "LastNameFilters.c" (e.g. "AcunaBoxFilters.c").