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 learn how to read and write real-world binary file formats, specifically BMP.

1 Background

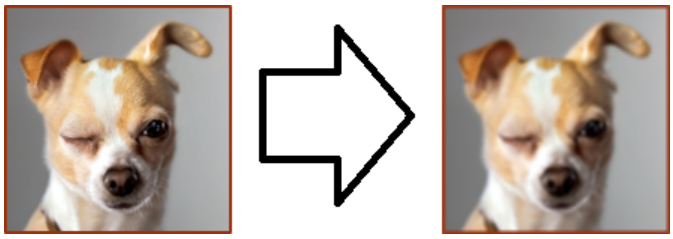


Figure 1: Application of blur algorithm to Wonderbread's portrait. (Image from Dr. Filley's TMC110 slide deck.)

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, specified by the user, and then transformed using a box blur image filter. The resulting file will be saved back to BMP file that can be viewed on the host system. **It is highly highly *highly* suggested that you develop a non-multithreaded version of the filter algorithm and load/save routines before beginning to thread it.**

This document is separated into four sections: Background, Requirements, Box Blur, Loading Binary Files, 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 Box Blur, we discuss the idea of the blur filter. In Loading Binary Files, we will discuss the BMP file format and how to interact with binary files. In Lastly, Submission discusses how your source code should be submitted on BlackBoard.

2 Requirements [40 points]

Your program needs to implement loading a binary BMP file, applying a box blur 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 256 by 256 pixels (see the MAXIMUM_IMAGE_SIZE constant in the base file). Assume that all BMP files are 24-bit uncompressed files[1] (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. This is shown in Figure 3. Regarding threading: 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). Using global memory is fine.

As a base requirement, your program must compile and run under Xubuntu (or another variant of Ubuntu) 18.04. Sample output for this program is shown on the right side of Figure 1.

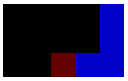
- Specific Requirements:

[1] In case you think this is making the assignment unrealistic, think again: this is the default Windows 10 BMP file format.

- Reads input filename from a command-line argument. (Your program must support be using used as: *./LastNameBoxBlur in.bmp out.bm*p) [1 points]
- Reads output filename from a command-line argument. [1 points]
- Create data structures for storing BMP files (must include DIB header struct, and 24-bit pixel struct). [6 points]
- Supports loading 24-bit BMP files. (No libraries allowed.) [6 points]
- Applies box blur algorithm to each pixel in data. [10 points]
- Supports saving 24-bit BMP files. (No libraries allowed.) [6 points]
- Box blur algorithm is computed in parallel with pthreads. Work must be evenly distributed over a number of threads defined by a constant called THREAD_COUNT, which must support all values great than 2 and less than the image's smallest dimension. [10 points]

3 Box Blur

The filter we will be applying in this assignment is called a box blur, takes a specic 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. 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)

$\rightarrow \frac{(0,0,0)+(0,0,0)+(0,0,0)+(0,0,0)+(100,0,0)+(0,0,200)}{6} = (16,\ 0,\ 32)$

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

4 Loading Binary Files

4.1 The BMP File Format

For reference, use the BMP specification on Wikipedia: https://en.wikipedia.org/wiki/BMP_file_format. In Figure 3, a graphical overview of a BMP file's layout is shown. The layout is literally the meaning of the bits/bytes in the file starting at 0 and going to the end of the file. The first (green) region shows the BMP header information in 14-bits: 2 for the signature, 4 for the file size, 2 for reserved1, 2 for reserved2, and 4 for file offset. *Details on each of these (such as their data format, and contents) can be found on the Wikipedia page.* For example, the area labeled "Signature" should contain the characters BM to confirm that the file is in BMP format. The second (blue region) shows the DIP header information in 40 bytes: 4 for header size, 4 for width, 4 for height, and so on. The last region (yellow) forms a 2D array of pixels. It stores columns from left to right, but rows are inverted to be bottom to top. Note that each row of pixels in this section is padded to be a multiple of four bytes. For example, if a line contains two pixels (24-bits or 3-bytes each), then an addition 2-bytes of blank data will be appended.

Review the following struct called BMP_Header which holds the bmp header information. (You should also consider creating structs to hold the dib header, and pixel data.) Notice that the entries in BMP_Header correspond to the pieces of data listed in the file format. In general, a chunk of 8-bits should be represented as a char, a chunk of 16-bits as a short, and 32-bits as a int. (Optionally: consider using unsigned types.)

```
struct BMP_Header {
        char signature[2];          //ID field
```

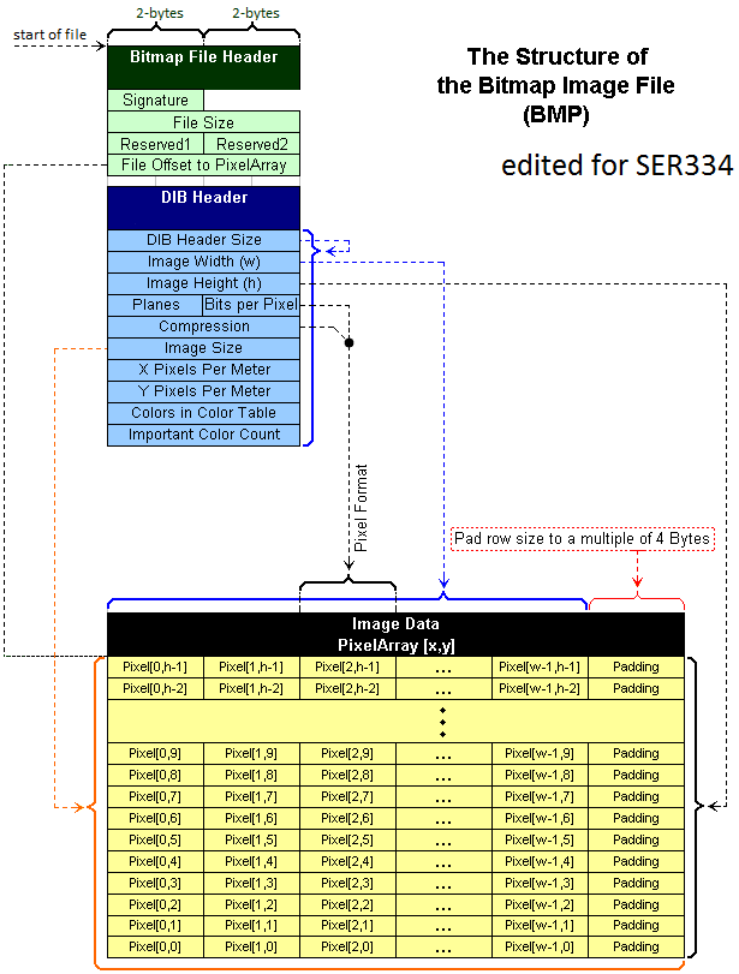


Figure 3: BMP file format structure for 24-bit files without compression. Image modified from https://en.wikipedia.org/wiki/File:BMPfileFormat.png.

```
        int size;                   //Size of the BMP file
        short reserved1;            //Application specific
        short reserved2;            //Application specific
        int offset_pixel_array;     //Offset where the pixel array can be found
};
```

Plan to review "Example 1" on the Wikipedia page to get a feel for the contents of each of these regions. A recreated version of that image is provided as the attached "test2.bmp" file. A good exercise would be to view the file in a hex editor like HxD.

4.2 Loading the BMP header

Already provided for you is a base file that contains code to load the BMP file header (the first 14 bits). Figure 4 shows the output of the base file. The that generates this output is shown below.

```
//sample code to read first 14 bytes of BMP file format
FILE* file = fopen("test2.bmp", "rb");
struct BMP_Header header;
```

Figure 4: Output of base file on test2.bmp.

```
//read bitmap file header (14 bytes)
fread(&header.signature, sizeof(char)*2, 1, file);
fread(&header.size, sizeof(int), 1, file);
fread(&header.reserved1, sizeof(short), 1, file);
fread(&header.reserved2, sizeof(short), 1, file);
fread(&header.offset_pixel_array, sizeof(int), 1, file);

printf("signature: %c%c\n", header.signature[0], header.signature[1]);
printf("size: %d\n", header.size);
printf("reserved1: %d\n", header.reserved1);
printf("reserved2: %d\n", header.reserved2);
printf("offset_pixel_array: %d\n", header.offset_pixel_array);

fclose(file);
```

The key functions here are:

- ```
  FILE *fopen(const char *filename, const char *mode)
  ```

  This creates a new file stream. fopen is used with the "rb" mode to indicate we are "r"eading a file in "b"inary mode. To read a file, use the mode "wb" instead of "rb".

- ```
  size_t fread(void *ptr, size_t size, size_t nitems, FILE *stream)
  ```

  For each call to fread, we give it first a pointer to an element of struct containing the BMP header, then the number of bytes to read, the number of times to read (typically 1), and the file stream to use. Note that order of the calls to fread define the order in which we read data so the order must match the file layout. (There is also a function called fwrite which works in exactly the opposite manner. The first won't be a pointer though.)

One function not used here but which may be useful, is fseek:

- ```
  int fseek ( FILE * stream, long int offset, int origin )
  ```

  The purpose of fseek is to move the "reading head" of the FILE object by some number of bytes. It can used to skip a number of bytes. The first parameter to fseek is the file pointer, followed by a number of bytes to move, and an origin. For origin, SEEK_CUR is a relative repositioning while SEEK_SET is global repositioning.

The basic idea to support loading a BMP file will be to parse it by byte-byte (using fread) into a set of structs. Later, you can use fwrite to write out the contents of those structs to a file stream to save the filtered image.
  ```

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.
- *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. (Possibly useful for extra credit.)
  - Useful functions:
    - *double sqrt(double x)* - Takes the square root of a number.

If you want to include any other files, please check with the instructor or TA before doing so.

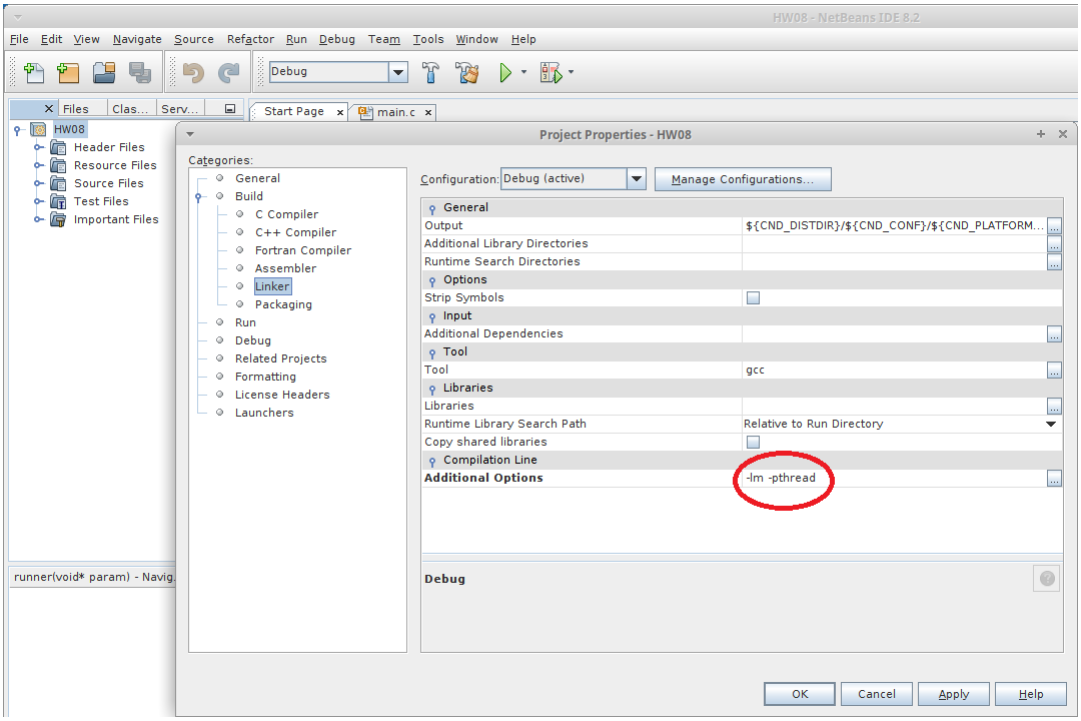
5.1 Linking with External Libraries

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
```

NetBeans is a little more complicated Open your 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 "LastNameBoxBlur.c" (e.g. "AcunaBoxBlur.c").