

## High Performance Computing

### Project - Homework #7

**Due: Monday Nov 20 2017 before 11:59 PM (Midnight)**

**Email-based help Cutoff: 5:00 PM on Fri, Nov 17 2017**

Total Maximum Points: 45

#### Submission Instructions

This homework assignment must be turned-in electronically via Canvas. Submit the following artifacts from this project via Canvas:

1. Your style checked C++ source file(s) (.cpp and .h files) that you developed for this project.
2. The PBS script file you developed for running tests on red hawk.
3. Your report document (a PDF file) named with the convention `MUId_HW6_Report.pdf` that meet the requirements of this homework.

No credit will be given for submitting code that does not meet base case requirements.

#### **Objective**

The objective of this homework is to continue to gain experience with OpenMP to develop parallel applications by developing a reasonably straightforward image searching program and verifying its performance on Red Hawk cluster.

### **Grading Rubric:**



This is an advanced course and consequently the expectations in this course are higher. Accordingly, the program submitted for this homework must pass necessary tests in order to qualify for earning a full score.

**NOTE: Program that do not compile, have methods/functions longer than 25 lines, or do not meet base case will be assigned zero score.**

Scoring for this assignment will be determined as follows assuming your program compiles (and is not skeleton code):

- **NOTE: Base case requirement:** In order to earn any credit for this project, on the bare minimum the program should operate correctly in serial / single thread mode.

- **Program 35 points:** allocated for overall implementation of the required data parallel program described in Part #1 of this document. This assumes that base case requirements have been met.
- **10 points:** allocated for performance analysis report. **You are required to collect performance data on Red Hawk cluster. These 10 points for the report are earned only if your parallelization is functionally correct.**
- **-1 Points:** for each warning generated by g++ when compiling your C++ program with -Wall (all warnings) flag.
- **-1 Points:** for each style violation reported by the CSE department's C++ style checker cpplint.py.

**NOTE:** Points will be deducted for violating stylistic qualities of the program such as: program follows formatting requirements (spacing, indentation, suitable variable names with appropriate upper/lowercase letters, etc). The program includes suitable comments at appropriate points in each method to elucidate flow of thought/logic in each method. Program strives to appropriately reuse as much code as possible.

## Background

This homework is a relatively straightforward (aka brute force) image searching approach, which requires the development of an OpenMP-parallelized image searching program. The images being used in this homework are Portable Network Graphics (PNG) files (see: [http://en.wikipedia.org/wiki/Portable\\_Network\\_Graphics](http://en.wikipedia.org/wiki/Portable_Network_Graphics)) that stores each pixel in an image. In other words, the image is a matrix of pixels and each pixel consists of four 8-bit values corresponding to the Red-Green-Blue-Alpha values. There are several different ways to view PNG files on Linux. The common command that I use is:

```
$ eog star_mask.png
```

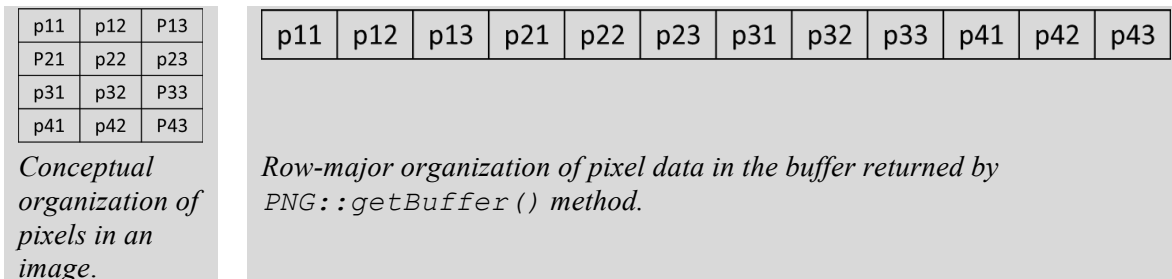
## Starter Code

You are supplied with PNG reading/writing class and several sample test images as part of this homework. Copy the necessary files from the following shared folder on Red Hawk to your NetBeans project directory:

```
/shared/raodm/csex43/homeworks/homework7
```

In this homework, you are supplied with a PNG class (review the supplied PNG.h file) that facilitates reading and writing of PNG files that contain images in RGBA (Red-

Green-Blue-Alpha) format. Note that the supplied PNG class requires the images to be in RGBA format otherwise it will not read such files (and will generate an exception). The pixels in an image may be obtained via call to the `PNG::getBuffer()` method. This method returns a “flat” buffer in which the pixels are stored in a row-major organization as illustrated in the figure below:



Each pixel consists of four 8-bit values stored in the order Red-Green-Blue-Alpha and they can be accessed as suggested below:

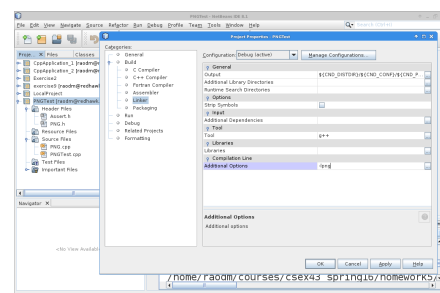
```
PNG img;
img.load("Mammogram.png");
// Maybe some more code here...
const std::vector<unsigned char>& buffer = img.getBuffer();
// Use a getPixelIndex method to convert row, col to index in buffer
const int index = getPixelIndex(10, 10, img.getWidth());
std::cout << "red   = " << buffer[index]
           << "green = " << buffer[index + 1]
           << "blue  = " << buffer[index + 2]
           << "Alpha = " << buffer[index + 3] << std::endl;
```

### Compiling:

The supplied `PNG.cpp` utilizes the system image processing libraries. Consequently, when compiling your program ensure you link with the library as shown below:

```
$ g++ -g -Wall -O3 -stdc++14 -fopenmp raodm_ImageSearch.cpp PNG.cpp -o ImageSearch -lpng
```



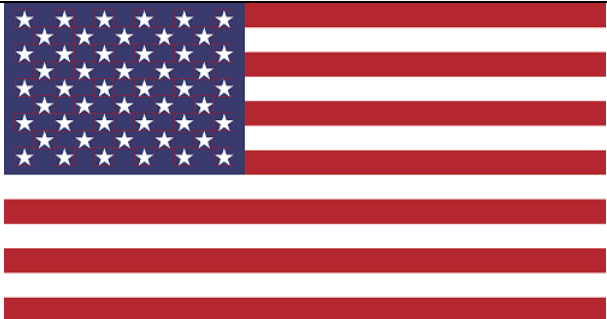
For NetBeans, add `-lpng` to Project Properties→Linker→Additional Options as shown in the adjacent screenshot (zoom into the adjacent screenshot as needed to see full details).



## Part #1: Parallelized Image Search

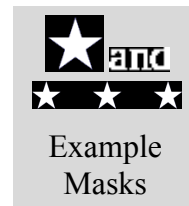
### Description:

This part of the homework requires development of a suitably parallelized OpenMP-based image search program that can identify and mark (with a red-box) occurrences of a given sub-image in a larger image as illustrated in the figure below:


<p><b>The Larger image:</b> This is the image to search in (first command-line argument to the program)</p>	
<p><b>The sub-image:</b> This is the sub-image, aka <i>mask</i>, to search for (second command-line argument to program) in the given larger image.</p>	
<p><b>Resulting image:</b> This is an image generated by the program in which sub-images in a black-box (output image name is supplied as third command-line argument to program).  <b>Note: You need to zoom-in to see the red-boxes around each one of the 50 stars.</b></p>	

The image search program must be designed to handle searching for matches using an image “mask”. An image mask is a black-and-white image that essentially represents a pattern to search for. Examples of image masks are shown in the adjacent figure. The pixels in an image mask are interpreted as follows:

- Pixels that are black essentially define the “background” for the information we are searching.
- Pixels that are not-black (or shades of grey) is the information or meaningful pattern to be identified.



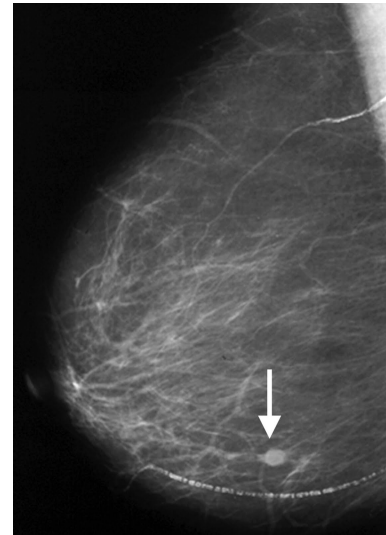
The objective of the search is to essentially distinguish the meaningful information immaterial of the (background and foreground) colors – the key distinguishing factor is sufficient contrast between the background and foreground (with black-and-white being the extreme contrast) as illustrated by the following examples:

				
Mask	Successful Match	Successful Match	Successful Match	Unsuccessful Match

Searching using masks is a powerful strategy not only for color agnostic processing but is also a handy strategy for processing images and information where the contrast in regions of the images is important. For example, this strategy can be used to identify high risk areas with potential tumors from mammograms (see adjacent image), high-contrast CT scans and MRI images.

***Tips: Searching using a Mask***

With a mask the key distinguishing factor is the contrast between background (identified by black pixels in the mask) versus the information (or foreground identified by white pixels in the mask). Consequently, the image search can proceed in the following manner:



1. Logically slide/move the mask over the image, row-by-row and column-by-column (from top-left to bottom-right of search image; somewhat akin to blocks in block matrix multiplication) to delineate the region of the image to search and check for potential match as suggested below.
2. Given a region of the image to search, first compute the average background-averaging the color of the pixels corresponding to the black pixels in the mask. For example, given the series of star images above, in the first successful case, all pixels corresponding to the light-blue color would be averaged (since the corresponding pixels in the mask are black).
3. Once the average background color has been determined, next recheck each pixel to verify if it matches (or mismatches) the mask in the following manner:
  - a. If the corresponding pixel in the mask is black, then the pixel should be “*same shade*” of the average background.
  - b. If the corresponding pixel in the mask is white, then the pixel should not be the “*same shade*” as the average background.

§ In this context, two colors are considered to be the “*same shade*” as determined by pixel color tolerance value (sixth optional command-line argument to the program).

4. The net-matching pixels are computed by subtracting the number of mismatching pixels (see previous step) from the number of matching pixels (see previous step), that is: `netMatch = matchPixCount - mismatchPixCount`. If the net number of matching pixels is more than the required percentage pixel (`perPixMatch`) match (fifth optional command-line argument to the program) then a match is found – that is, area matches if `netMatch > mask.getWidth() * mask.getHeight() * perPixMatch / 100`.

5. Once a region of the image has been matched, it must not be included as part of any other matches. This is best accomplished using a list that tracks the regions that has been matched. It is up to you on how you design and manage this list of successful matches.
6. Once all the regions have been searched, sort the list of matches (based on row and then on column) and print the list of matched regions. The matching regions are printed in the form: row, col, row + mask.height, col + mask.width (see sample output for details)
7. For each matched region, draw a red box around the region. You can draw a red box in a given image using the following method:

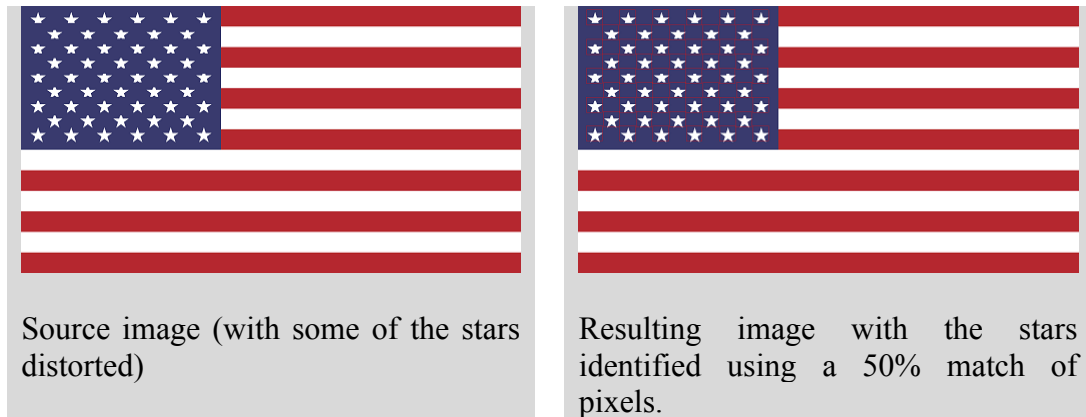
```
void drawBox(PNG& png, int row, int col, int width, int height) {  
    // Draw horizontal lines  
    for(int i = 0; (i < width); i++) {  
        png.setRed(row, col + i);  
        png.setRed(row + height, col + i);  
    }  
    // Draw vertical lines  
    for(int i = 0; (i < height); i++) {  
        png.setRed(row + i, col);  
        png.setRed(row + i, col + width);  
    }  
}
```

8. Write the resulting PNG (with red boxes in it) to a given output file using helper methods in the supplied PNG class

### ***Search Options:***

The search for sub-images must permit the following additional options:

1. Percentage pixel match: A section of the search image must be considered as a match to the given sub-image if sufficient number of pixels matches. This parameter specifies the percentage number of pixels that must match (based on second option below) in order to decide that a region of the image is a match to the given sub-image. This parameter is a value in the range 1 to 100 (corresponding to 0% to 100%). For example, if this parameter is 50, then a 50% pixel match is deemed sufficient to identify a section of the search image as a match to the search image. Accordingly, with a 50% match the image search program should be able to identify stars (using the search sub-image shown earlier) and identify 50 stars on the following image (which has some partial stars showing (possibly because the flag was waving or the camera had some distortion in it):



This is an optional command-line argument to the program. This value may be specified as the fifth command-line argument to the program. If sufficient number of arguments is not supplied, then the default value for this option is 75 (corresponding to 75% match).

2. **Pixel color tolerance:** The three color values for each pixel (namely RGB) does not need to match exactly and can match approximately with a given tolerance. This option is an integer the range 1 to 255 that indicates the acceptable tolerance (or difference) between pixel color values in the search image versus the sub-image. For example, assume that a pixel has values **<230, 0, 255>** and the tolerance is set to 10 then the following pixels would be a match (as none of the individual RGB values differ by more than 10): **<240, 0, 255>**, **<220, 9, 250>**, **<230, 9, 246>** while the following pixels will not be a match: **<200, 0, 255>**, **<230, 20, 255>**, **<210, 0, 240>**. In other words, close shades of the same color are acceptable but not shades of other colors.

This is an optional command-line argument to the program. This value may be specified as the sixth command-line argument to the program. If sufficient number of arguments is not supplied, then the default value for this option is 32.

### ***Command-line Arguments to program:***

The following 6 command-line arguments (the last three are optional) must be accepted and suitably processed by the program:

1. The first required command-line argument is the path to the large PNG file to be searched-in.
2. The second required command-line argument is the path to the PNG file that contains the sub-image to be searched for.
3. The third required command-line argument is the name of the output PNG file to which the resulting image is to be written.

4. The fourth optional command-line argument is a Boolean string ("**true**" or "**false**") to indicate if the search sub-image is a mask. In this homework you may assume this parameter is always **true**. (and safely ignore it)
5. The fifth optional command-line argument is the desired percentage pixel match (as described earlier) to determine a match between the given sub-image and the search image. **If this parameter is not specified then use 75 as the default value.**
6. The sixth optional command-line argument is the pixel color tolerance (as described earlier). **If this argument is not specified then use 32 as the default value.**

### *Sample Outputs:*

Expected outputs from multiple independent runs of the completed program are shown below (resulting images with suitable red-boxes are not shown but are necessary for full score in this homework). The matching regions are printed in the form: row, col, row + mask.height, col + mask.width. **Note: The output is exactly the same immaterial of the number of threads used.**

```
$ ./ImageSearch TestImage.png and_mask.png result.png true 75 16
sub-image matched at: 73, 630, 85, 660
sub-image matched at: 120, 310, 132, 340
sub-image matched at: 202, 677, 214, 707
sub-image matched at: 226, 864, 238, 894
sub-image matched at: 274, 67, 286, 97
Number of matches: 5
```

```
$ ./ImageSearch TestImage.png i_mask.png result.png true
sub-image matched at: 45, 243, 60, 248
sub-image matched at: 45, 249, 60, 254
sub-image matched at: 45, 377, 60, 382
sub-image matched at: 45, 383, 60, 388
sub-image matched at: 45, 528, 60, 533
sub-image matched at: 45, 545, 60, 550
sub-image matched at: 45, 750, 60, 755
sub-image matched at: 45, 837, 60, 842
sub-image matched at: 45, 843, 60, 848
sub-image matched at: 69, 108, 84, 113
sub-image matched at: 69, 152, 84, 157
sub-image matched at: 69, 177, 84, 182
sub-image matched at: 69, 202, 84, 207
sub-image matched at: 69, 237, 84, 242
sub-image matched at: 69, 299, 84, 304
sub-image matched at: 69, 358, 84, 363
sub-image matched at: 69, 395, 84, 400
sub-image matched at: 69, 471, 84, 476
sub-image matched at: 69, 521, 84, 526
sub-image matched at: 69, 655, 84, 660
sub-image matched at: 69, 712, 84, 717
sub-image matched at: 69, 730, 84, 735
sub-image matched at: 69, 776, 84, 781
sub-image matched at: 69, 787, 84, 792
sub-image matched at: 92, 66, 107, 71
sub-image matched at: 92, 91, 107, 96
sub-image matched at: 92, 330, 107, 335
sub-image matched at: 92, 361, 107, 366
sub-image matched at: 92, 388, 107, 393
sub-image matched at: 92, 519, 107, 524
sub-image matched at: 92, 603, 107, 608
sub-image matched at: 92, 625, 107, 630
sub-image matched at: 92, 735, 107, 740
sub-image matched at: 92, 746, 107, 751
```



sub-image matched at: 92, 838, 107, 843  
sub-image matched at: 92, 889, 107, 894  
sub-image matched at: 116, 204, 131, 209  
sub-image matched at: 116, 251, 131, 256  
sub-image matched at: 116, 274, 131, 279  
sub-image matched at: 116, 290, 131, 295  
sub-image matched at: 116, 335, 131, 340  
sub-image matched at: 116, 435, 131, 440  
sub-image matched at: 116, 519, 131, 524  
sub-image matched at: 116, 678, 131, 683  
sub-image matched at: 116, 701, 131, 706  
sub-image matched at: 116, 718, 131, 723  
sub-image matched at: 116, 857, 131, 862  
sub-image matched at: 140, 175, 155, 180  
sub-image matched at: 140, 194, 155, 199  
sub-image matched at: 140, 272, 155, 277  
sub-image matched at: 140, 283, 155, 288  
sub-image matched at: 140, 344, 155, 349  
sub-image matched at: 140, 472, 155, 477  
sub-image matched at: 140, 524, 155, 529  
sub-image matched at: 140, 530, 155, 535  
sub-image matched at: 140, 546, 155, 551  
sub-image matched at: 140, 632, 155, 637  
sub-image matched at: 140, 673, 155, 678  
sub-image matched at: 140, 684, 155, 689  
sub-image matched at: 140, 733, 155, 738  
sub-image matched at: 164, 39, 179, 44  
sub-image matched at: 164, 165, 179, 170  
sub-image matched at: 164, 190, 179, 195  
sub-image matched at: 164, 274, 179, 279  
sub-image matched at: 164, 291, 179, 296  
sub-image matched at: 198, 42, 213, 47  
sub-image matched at: 198, 196, 213, 201  
sub-image matched at: 198, 254, 213, 259  
sub-image matched at: 198, 279, 213, 284  
sub-image matched at: 198, 645, 213, 650  
sub-image matched at: 198, 702, 213, 707  
sub-image matched at: 198, 764, 213, 769  
sub-image matched at: 198, 775, 213, 780  
sub-image matched at: 222, 88, 237, 93  
sub-image matched at: 222, 171, 237, 176  
sub-image matched at: 222, 208, 237, 213  
sub-image matched at: 222, 236, 237, 241  
sub-image matched at: 222, 289, 237, 294  
sub-image matched at: 222, 319, 237, 324  
sub-image matched at: 222, 332, 237, 337  
sub-image matched at: 222, 418, 237, 423  
sub-image matched at: 222, 424, 237, 429  
sub-image matched at: 222, 536, 237, 541  
sub-image matched at: 222, 565, 237, 570  
sub-image matched at: 222, 606, 237, 611  
sub-image matched at: 222, 688, 237, 693  
sub-image matched at: 222, 710, 237, 715  
sub-image matched at: 222, 764, 237, 769  
sub-image matched at: 222, 770, 237, 775  
sub-image matched at: 222, 810, 237, 815  
sub-image matched at: 246, 58, 261, 63  
sub-image matched at: 246, 133, 261, 138  
sub-image matched at: 246, 178, 261, 183  
sub-image matched at: 246, 219, 261, 224  
sub-image matched at: 246, 322, 261, 327  
sub-image matched at: 246, 421, 261, 426  
sub-image matched at: 246, 475, 261, 480  
sub-image matched at: 246, 519, 261, 524  
sub-image matched at: 246, 545, 261, 550  
sub-image matched at: 246, 610, 261, 615  
sub-image matched at: 246, 621, 261, 626  
sub-image matched at: 246, 657, 261, 662  
sub-image matched at: 246, 854, 261, 859  
sub-image matched at: 270, 92, 285, 97  
sub-image matched at: 270, 127, 285, 132  
sub-image matched at: 270, 189, 285, 194  
sub-image matched at: 270, 311, 285, 316  
sub-image matched at: 270, 342, 285, 347  
sub-image matched at: 270, 348, 285, 353  
sub-image matched at: 270, 388, 285, 393  
sub-image matched at: 270, 531, 285, 536  
sub-image matched at: 270, 562, 285, 567  
sub-image matched at: 270, 630, 285, 635  
sub-image matched at: 270, 656, 285, 661  
sub-image matched at: 270, 716, 285, 721

```
sub-image matched at: 270, 810, 285, 815
sub-image matched at: 294, 131, 309, 136
sub-image matched at: 294, 181, 309, 186
sub-image matched at: 294, 207, 309, 212
sub-image matched at: 294, 230, 309, 235
sub-image matched at: 294, 246, 309, 251
sub-image matched at: 294, 308, 309, 313
sub-image matched at: 294, 330, 309, 335
sub-image matched at: 294, 400, 309, 405
sub-image matched at: 294, 507, 309, 512
sub-image matched at: 294, 560, 309, 565
sub-image matched at: 294, 566, 309, 571
sub-image matched at: 294, 572, 309, 577
sub-image matched at: 294, 645, 309, 650
sub-image matched at: 294, 651, 309, 656
sub-image matched at: 294, 706, 309, 711
sub-image matched at: 294, 742, 309, 747
sub-image matched at: 294, 748, 309, 753
sub-image matched at: 294, 760, 309, 765
sub-image matched at: 368, 573, 383, 578
sub-image matched at: 372, 127, 387, 132
sub-image matched at: 372, 304, 387, 309
sub-image matched at: 372, 556, 387, 561
sub-image matched at: 372, 562, 387, 567
sub-image matched at: 372, 641, 387, 646
sub-image matched at: 396, 123, 411, 128
sub-image matched at: 396, 338, 411, 343
sub-image matched at: 396, 384, 411, 389
sub-image matched at: 396, 806, 411, 811
sub-image matched at: 420, 129, 435, 134
sub-image matched at: 420, 471, 435, 476
sub-image matched at: 440, 414, 455, 419
sub-image matched at: 444, 232, 459, 237
sub-image matched at: 444, 328, 459, 333
sub-image matched at: 444, 602, 459, 607
sub-image matched at: 444, 706, 459, 711
sub-image matched at: 444, 760, 459, 765
sub-image matched at: 444, 806, 459, 811
sub-image matched at: 468, 38, 483, 43
sub-image matched at: 526, 340, 541, 345
sub-image matched at: 526, 542, 541, 547
sub-image matched at: 550, 853, 565, 858
sub-image matched at: 570, 834, 585, 839
sub-image matched at: 574, 599, 589, 604
sub-image matched at: 574, 885, 589, 890
sub-image matched at: 593, 830, 608, 835
sub-image matched at: 597, 198, 612, 203
sub-image matched at: 597, 354, 612, 359
sub-image matched at: 621, 245, 636, 250
sub-image matched at: 621, 379, 636, 384
sub-image matched at: 621, 524, 636, 529
sub-image matched at: 621, 541, 636, 546
Number of matches: 167
```

```
$ ./ImageSearch Flag_of_the_US.png star_mask.png result.png true 50 32
sub-image matched at: 20, 35, 80, 98
sub-image matched at: 20, 168, 80, 231
sub-image matched at: 20, 300, 80, 363
sub-image matched at: 20, 433, 80, 496
sub-image matched at: 20, 566, 80, 629
sub-image matched at: 20, 698, 80, 761
sub-image matched at: 77, 101, 137, 164
sub-image matched at: 77, 234, 137, 297
sub-image matched at: 77, 367, 137, 430
sub-image matched at: 77, 499, 137, 562
sub-image matched at: 77, 632, 137, 695
sub-image matched at: 133, 36, 193, 99
sub-image matched at: 133, 169, 193, 232
sub-image matched at: 133, 301, 193, 364
sub-image matched at: 133, 434, 193, 497
sub-image matched at: 133, 567, 193, 630
sub-image matched at: 133, 699, 193, 762
sub-image matched at: 190, 102, 250, 165
sub-image matched at: 190, 235, 250, 298
sub-image matched at: 190, 368, 250, 431
sub-image matched at: 190, 500, 250, 563
sub-image matched at: 190, 633, 250, 696
sub-image matched at: 247, 36, 307, 99
sub-image matched at: 247, 168, 307, 231
```

```
sub-image matched at: 247, 301, 307, 364
sub-image matched at: 247, 434, 307, 497
sub-image matched at: 247, 566, 307, 629
sub-image matched at: 247, 699, 307, 762
sub-image matched at: 304, 102, 364, 165
sub-image matched at: 304, 235, 364, 298
sub-image matched at: 304, 367, 364, 430
sub-image matched at: 304, 500, 364, 563
sub-image matched at: 304, 632, 364, 695
sub-image matched at: 361, 35, 421, 98
sub-image matched at: 361, 168, 421, 231
sub-image matched at: 361, 301, 421, 364
sub-image matched at: 361, 433, 421, 496
sub-image matched at: 361, 566, 421, 629
sub-image matched at: 361, 699, 421, 762
sub-image matched at: 418, 102, 478, 165
sub-image matched at: 418, 234, 478, 297
sub-image matched at: 418, 367, 478, 430
sub-image matched at: 418, 499, 478, 562
sub-image matched at: 418, 632, 478, 695
sub-image matched at: 475, 35, 535, 98
sub-image matched at: 475, 168, 535, 231
sub-image matched at: 475, 300, 535, 363
sub-image matched at: 475, 433, 535, 496
sub-image matched at: 475, 566, 535, 629
sub-image matched at: 475, 698, 535, 761
Number of matches: 50
```

```
$ ./ImageSearch MiamiMarcumCenter.png WindowPane_mask.png result.png true 50 64
sub-image matched at: 567, 818, 601, 859
sub-image matched at: 567, 1021, 601, 1062
sub-image matched at: 568, 619, 602, 660
sub-image matched at: 568, 1226, 602, 1267
sub-image matched at: 578, 1791, 612, 1832
sub-image matched at: 582, 1996, 616, 2037
sub-image matched at: 590, 2198, 624, 2239
sub-image matched at: 605, 817, 639, 858
sub-image matched at: 605, 1020, 639, 1061
sub-image matched at: 606, 618, 640, 659
sub-image matched at: 607, 1225, 641, 1266
sub-image matched at: 616, 1791, 650, 1832
sub-image matched at: 620, 1995, 654, 2036
sub-image matched at: 627, 2197, 661, 2238
sub-image matched at: 816, 1209, 850, 1250
Number of matches: 15
```

### ***Parallelization with OpenMP***

Parallelization of the program with OpenMP should be straightforward once the serial version of the program has been developed. Appropriately parallelize suitable loop (or loops) as you see fit in the program so as to extract good performance. Pay attention to how you are managing the list of matching regions from different threads. Of course the project expects you to independently design, implement, and validate suitable parallelization approach for this problem. How you parallelize the problem is completely up to you.



Needless to add, the list of matches generated by your program should be exactly the same immaterial of the number of threads used to run your program. Meeting this criteria is important for full credit in this project.

### ***Performance Analysis***

Complete the short performance analysis report provided with this homework. Once you have tested and verified correct operation of the parallel program, develop a report containing a comprehensive performance analysis (using at least 5 sample runs with 95% confidence intervals) of your program using 1, 4, and 8 threads using three different pairs of images (namely: MiamiMarcumCenter.png with WindoPane\_mask.png, TestImage.png with and\_mask.png, Mammogram.png with Cancer\_mask.png) and record timings in the report document.

### ***Troubleshooting***

In order to help you troubleshoot your logic, the supplied ImageSearch\_sequential program can print details on checks performed at a given row and column. You can run the executable on Red Hawk cluster as shown below:

```
$ ./ImageSearch_sequential Flag_of_the_US.png star_mask.png result.png
true 50 32 20 35
Number of black mask pixels: 1051
Total of red, blue, green values: 238111, 245777, 237957
Average Background color: (226,226,233)
Matching pixel count at row: 20, col: 35 = 2954
Matching pixels needed: 1890
```

### ***Turn-in***

Submit the following artifacts from this project via Canvas:

1. Your style checked C++ source file(s) (.cpp and .h files) that you developed for this project.
2. The PBS script file you developed for running tests on red hawk.
3. Your report document (a PDF file) named with the convention MUIId\_HW7\_Report.pdf that meet the requirements of this homework.

No credit will be given for submitting code that does not meet base case requirements. Verify that your program meets all the requirements as stated in the grading rubric. Ensure your C++ source files are named with the stipulated naming convention. **Do not submit zip/7zip/tar/gzip files. Upload each file independently.**