CSCI 455: Project 1 (Lab #5–6) — Image Transformation

Darwin Jacob Groskleg

Winter 2020

Contents

Program Output

Console

```
Transcript written on tmp/Lab5-6-HANDIN.log.
[ darwingroskleg@starbuck ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab5-6-Project_1_Image_Transformation make test-serial
mpicc -std=c99 -Wall -Wextra -g -D_GLIBCXX_DEBUG -O0   -c -o transform-serial.o ./src/transform-serial.c
mpicc   transform-serial.o ppm.o filter_ppm.o  -lm -o transform-serial
./transform-serial
infile name: ./im1.ppm
Neighborhood pixel radius |x y|: 5 5
Average intensity is: 108.889805
 darwingroskleg@starbuck ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab5-6-Project_1_Image_Transformation
```

Figure 1: Output from serial implementation found in `src/transform-serial.c` for the first image.

```
 darwingroskleg@starbuck ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab5-6-Project_1_Image_Transformation make
mpicc -std=c99 -Wall -Wextra -g -D_GLIBCXX_DEBUG -O0   -c -o transform-serial.o ./src/transform-serial.c
mpicc -std=c99 -Wall -Wextra -g -D_GLIBCXX_DEBUG -O0   -c -o ppm.o ./src/ppm.c
mpicc -std=c99 -Wall -Wextra -g -D_GLIBCXX_DEBUG -O0   -c -o filter_ppm.o ./src/filter_ppm.c
mpicc -std=c99 -Wall -Wextra -g -D_GLIBCXX_DEBUG -O0   -c -o transform-parall.o ./src/transform-parall.c
mpicc -std=c99 -Wall -Wextra -g -D_GLIBCXX_DEBUG -O0   -c -o test_filter.o ./src/test_filter.c
mpicc   transform-serial.o ppm.o filter_ppm.o  -lm -o transform-serial
mpicc   transform-parall.o ppm.o filter_ppm.o  -lm -o transform-parall
mpicc   test_filter.o ppm.o filter_ppm.o  -lm -o test_filter
 darwingroskleg@starbuck ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab5-6-Project_1_Image_Transformation make test-parall-all
Platform: Darwin (4 cpu cores recognized)
PMIX_MCA_gds=hash mpirun --host localhost --mca btl_vader_backing_directory /tmp --mca btl ^tcp --oversubscribe  -np 4 ./transform-parall  im1.ppm
[0]: infile name: im1.ppm
[0]: file read, 515788 pixels, 676px wide, 763px in height
[0]: Averaging Filter 4 node parallel computation time: 8.8748e-08 seconds
[0]: Threshold Filter 4 node parallel computation time: 5.341e-09 seconds
[0]: Overall average intensity is: 108.889805
Platform: Darwin (4 cpu cores recognized)
PMIX_MCA_gds=hash mpirun --host localhost --mca btl_vader_backing_directory /tmp --mca btl ^tcp --oversubscribe  -np 4 ./transform-parall  im2.ppm
[0]: infile name: im2.ppm
[0]: file read, 486160 pixels, 590px wide, 824px in height
[0]: Averaging Filter 4 node parallel computation time: 8.8676e-08 seconds
[0]: Threshold Filter 4 node parallel computation time: 3.389e-09 seconds
[0]: Overall average intensity is: 43.199266
Platform: Darwin (4 cpu cores recognized)
PMIX_MCA_gds=hash mpirun --host localhost --mca btl_vader_backing_directory /tmp --mca btl ^tcp --oversubscribe  -np 4 ./transform-parall  im3.ppm
[0]: infile name: im3.ppm
[0]: file read, 419175 pixels, 575px wide, 729px in height
[0]: Averaging Filter 4 node parallel computation time: 7.4068e-08 seconds
[0]: Threshold Filter 4 node parallel computation time: 4.194e-09 seconds
[0]: Overall average intensity is: 50.299069
```

Figure 2: Console output from running parallel implementation found in `src/transform-parall.c` for all 3 image files.

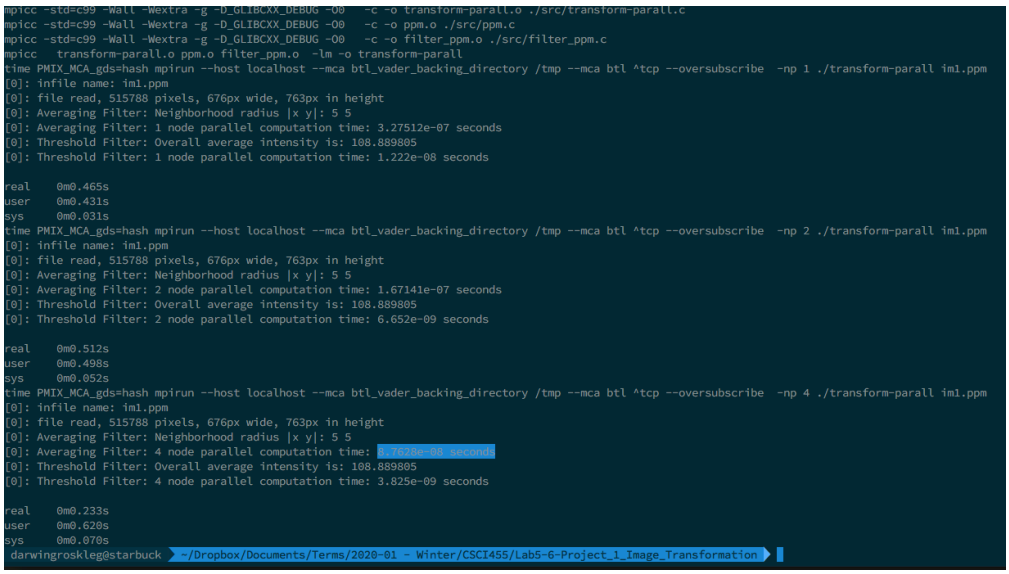

Figure 3: Console output from node scaling performance comparison (see Q3).

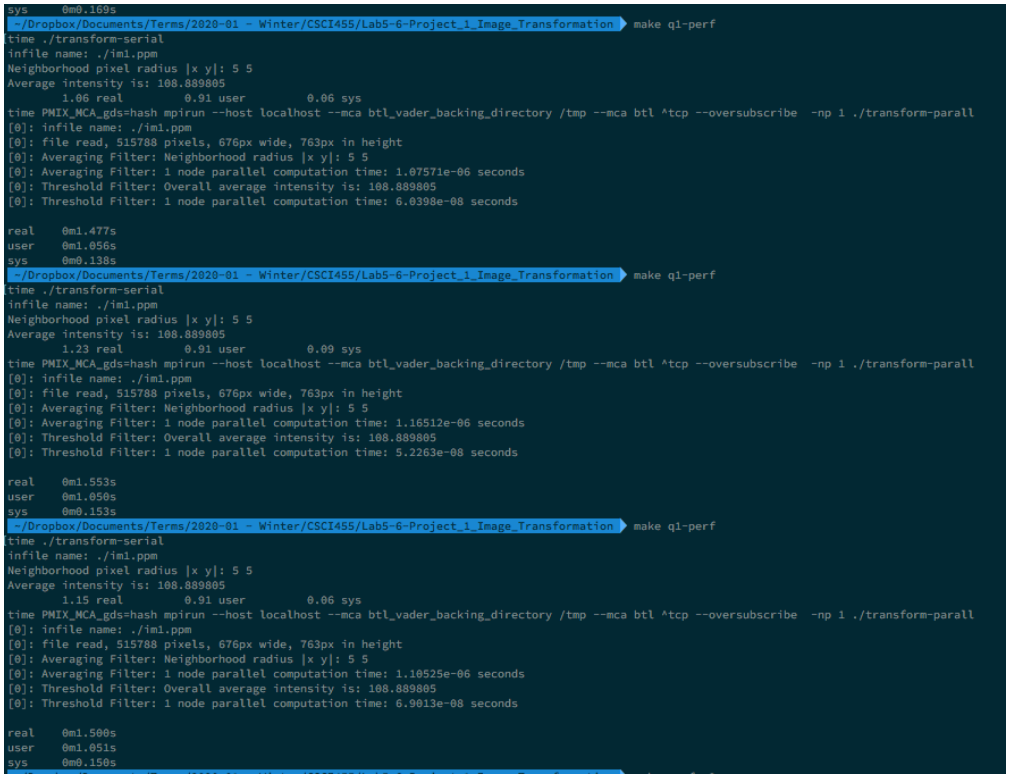

Figure 4: Console output from single threaded performance comparison (see Q1).

Filtered Images

Images shown before and after filtering using parallel implementation. Ordered original, fuzzy then threshold from the left. Averaging filter used a radius of (x=5, y=5).

Figure 5: From im1.ppm of *Small Cowper Madonna* by Raphael, 1505. Overall intensity of 108.889805.

Figure 6: From im2.ppm of *Doña Teresa Sureda* by Francisco Goya, 1805. Overall intensity of 43.199266.

Figure 7: From im3.ppm of *Sir Brian Tuke* by Hans Holbein the Younger, c. 1532-1534. Overall intensity of 50.299069.

Questions

Q1. How does the single threaded perform in the serial vs parallel code?

Using the `time` command running both parallel (1 node) and serial on `im1.ppm` we get the following averages for **real** execution time in seconds:

- Serial: 1.15 seconds (1.06, 1.23, 1.15)
- Parallel: 1.51 seconds (1.477, 1.553, 1.500)

Four runs of the code in a row were made, throwing away the first where the data may not have been in the disk cache (mitigates outliers). The resulting performance shows the serial code, without the extra unused machinery of MPI to setup, is faster than the single node execution of the parallel code.

Q2. Which method of communication was implemented and why?

There are 3 areas of communication: loading the initial image file data, running the fuzzy transformation and running the threshold transformation. First, opening the image file is done by the root process which then broadcasts the image data to all other processes. The fuzzy transformation only needs to communicate once, that is a Gather from all processes, their respective subset of the image data, to the root node. Finally, the threshold transformation communicates using All-Reduce in the calculation of the overall image intensity and then again with a Gather in the same way as the fuzzy transformation did.

Q3. How does the parallel code scale with number of processors?

Program performance indexed by node count, tracked using `time`:

- 1: real 0.465s

- 2: real 0.512s
- 4: real 0.233s

Below are the performance times of the individual transform algorithms as measured in the program using MPI's given performance functions `WTick()` and `WTime()`.

The "fuzzy transformation" scaled as follows:

- 1: 3.27512 e-07 seconds
- 2: 1.67141 e-07 seconds
- 4: 0.87628 e-07 seconds

The "threshold transformation" scaled as follows:

- 1: 1.222 e-08 seconds
- 2: 0.6652 e-08 seconds
- 4: 0.3825 e-08 seconds

Further Improvements

Improvements that were not required but I believe would yield better performance or speedup.

- Parallel I/O: use the MPI file I/O calls to read and write to the PPM image files. This would allow the expensive overhead of broadcasting (communicating) all the file data from the root process to the others. Especially true for the Averaging Filter. Most importantly this improves overall program performance.
- Pipe-lining with Cartesian Grid for communicating. Avoid calculating sums twice with the averaging filter.

Code

See `src/transform-serial.c` for serial implementation (modified from prof. L.T. Yang). See `src/transform-parall.c` for the MPI/parallel implementation. All other sources files are used in both implementations. The `Makefile` is included to clarify how performance tests may have been conducted but personal/private makefiles libraries are not included in this document.

src/transform-parall.c

```c
/* transform-parall.c
 * ------------------
 * Authors: Darwin Jacob Groskleg
 */
#include <assert.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#include <mpi.h>

#include "ppm.h"
#include "filter_ppm.h"

enum TaskRanks {
    ConsoleIORank = 0
};

void make_ppm_pixel_dt(MPI_Datatype* MPI_PIXEL);
char* append_to_basename(const char* filename, const char* appendix);

int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int cluster_size;
    MPI_Comm_size(MPI_COMM_WORLD, &cluster_size);

    const char* default_image_filename = "./im1.ppm";
    char* image_filename;
    if (argc > 1) {
        image_filename = (char *) malloc( sizeof (char) * strlen(argv[1]));
        strcpy(image_filename, argv[1]);
    } else {
        size_t default_len = strlen(default_image_filename);
        image_filename = (char *) malloc( sizeof (char) * default_len);
        strcpy(image_filename, default_image_filename);
    }
    if (rank == ConsoleIORank)
        printf("[%d]: infile name: %s\n", rank, image_filename);

    /* Create Pixel datatype for MPI */
    MPI_Datatype MPI_PIXEL;
    make_ppm_pixel_dt(&MPI_PIXEL);

    double t0,tf,t_delta_seconds;
    double seconds_per_tick = MPI_Wtick();

/*
 * Read and Share Initial PPM Image Data
 *
 */
    Pixel* in_image = (Pixel *) malloc(PPM_MAX_PIXELS * sizeof (Pixel));
    assert(in_image != 0);

```

```
56      int width, height, max;
57      int err = 0;
58      if (rank == ConsoleIORank) {
59          err = read_ppm(image_filename, &width, &height, &max, (char *)in_image);
60          if (err != 0) MPI_Abort(MPI_COMM_WORLD, err);
61          printf("[%d]: file read, %d pixels, %dpx wide, %dpx in height\n",
62                  rank, width*height, width, height);
63      }
64      MPI_Bcast(&width,  1, MPI_INT, ConsoleIORank, MPI_COMM_WORLD);
65      MPI_Bcast(&height, 1, MPI_INT, ConsoleIORank, MPI_COMM_WORLD);
66      MPI_Bcast(&max,    1, MPI_INT, ConsoleIORank, MPI_COMM_WORLD);
67      MPI_Bcast(in_image, width*height, MPI_PIXEL, ConsoleIORank, MPI_COMM_WORLD);
68
69      const Image img  = { .pixels=in_image, width, height, max };
70      int subset_size  = img_partition_pixels(img.width*img.height, cluster_size);
71      int subset_start = img_partition_head(subset_size, rank);
72
73
74  /*
75   * The "fuzzy" transformation
76   *
77   */
78      Pixel* out_image_fuzzy = (Pixel *) malloc(PPM_MAX_PIXELS * sizeof (Pixel));
79      assert(out_image_fuzzy != 0);
80
81      t0 = MPI_Wtime();
82      int xradius = 5, yradius = 5;
83      averaging_filter_subset(img, subset_start, subset_size, xradius, yradius,
84              out_image_fuzzy);
85      MPI_Gather(out_image_fuzzy+subset_start, subset_size, MPI_PIXEL,
86                 out_image_fuzzy,             subset_size, MPI_PIXEL,
87                 ConsoleIORank, MPI_COMM_WORLD);
88      tf = MPI_Wtime();
89
90      if (rank == ConsoleIORank) {
91          char* fuzzy_fname = append_to_basename(image_filename,
92                                                "-parall-fuzzy.ppm");
93          err = write_ppm(fuzzy_fname, width, height, (char *) out_image_fuzzy);
94          if (err != 0) MPI_Abort(MPI_COMM_WORLD, err);
95
96          printf("[%d]: Averaging Filter: Neighborhood radius |x y|: %d %d\n",
97                  rank,                                xradius, yradius);
98
99          t_delta_seconds = (tf - t0) * seconds_per_tick;
100         printf("[%d]: Averaging Filter: %d node parallel computation time: ",
101                 rank,             cluster_size);
102         printf("%g seconds\n", t_delta_seconds);
103     }
104
105
106 /*
107  * The threshold transformation
108  *
109  */
110     GPixel* out_image_thresh = (GPixel *) malloc(PPM_MAX_PIXELS * sizeof (GPixel));
111     assert(out_image_thresh != 0);
```

```
112
113     t0 = MPI_Wtime();
114     double local_intensity=sum_subset_intensity(img, subset_start, subset_size);
115     double global_threshold;
116     MPI_Allreduce(&local_intensity, &global_threshold, 1, MPI_DOUBLE, MPI_SUM,
117             MPI_COMM_WORLD);
118     global_threshold /= img.width * img.height;
119     threshold_filter_subset(img, subset_start, subset_size, global_threshold,
120             out_image_thresh);
121
122     MPI_Gather(out_image_thresh + subset_start, subset_size, MPI_UNSIGNED_CHAR,
123                out_image_thresh,                subset_size, MPI_UNSIGNED_CHAR,
124                ConsoleIORank, MPI_COMM_WORLD);
125     tf = MPI_Wtime();
126
127     if (rank == ConsoleIORank) {
128         char* threshold_fname = append_to_basename(image_filename,
129                                                    "-parall-threshold.pgm");
130         err = write_pgm(threshold_fname, width, height,
131                         (char *) out_image_thresh);
132         if (err != 0) exit(err);
133
134         printf("[%d]: Threshold Filter: Overall average intensity is: %f\n",
135                 rank, global_threshold);
136         t_delta_seconds = (tf - t0) * seconds_per_tick;
137         printf("[%d]: Threshold Filter: %d node parallel computation time: ",
138                 rank, cluster_size);
139         printf("%g seconds\n", t_delta_seconds);
140         fflush(stdout);
141     }
142
143     MPI_Finalize();
144     return EXIT_SUCCESS;
145 }
146
147 void make_ppm_pixel_dt(MPI_Datatype* MPI_PIXEL) {
148     int lengths[3] = {1,1,1};
149     const MPI_Aint displacements[3] = {
150         0,
151         sizeof (unsigned char),
152         2 * sizeof (unsigned char)
153     };
154     MPI_Datatype types[3] = {
155         MPI_UNSIGNED_CHAR, MPI_UNSIGNED_CHAR, MPI_UNSIGNED_CHAR
156     };
157     MPI_Type_create_struct(3, lengths, displacements, types, MPI_PIXEL);
158     MPI_Type_commit(MPI_PIXEL);
159 }
160
161 /* example:
162  *
163  *  append_to_basename("img.ppm", "-fuzzy.pgm")
164  *      ->"img-fuzzy.pgm"
165  */
166 char* append_to_basename(const char* filename, const char* appendix) {
167     size_t newlen = strlen(filename) + strlen(appendix);
```

```
168     char* newfile = (char *) malloc(sizeof (char) * newlen);
169     char* ft_extension = strrchr(filename, '.');
170     size_t basename_len = strlen(filename) - strlen(ft_extension);
171     strncpy(newfile, filename, basename_len); // without the file extension
172     strcat(newfile, appendix);
173     //strcat(newfile, ft_extension);
174     //printf("outfile name: %s\n", newfile);
175 
176     return newfile;
177 }
```

src/filter_ppm.h

```c
/* filter_ppm.h
 * ------------
 * Authors: Darwin Jacob Groskleg
 *
 * Purpose: extract filtering methods from both programs to make them more
 *          testable by comparison.
 *          Extension of ppm image struct.
 *
 * Filters: take image data & image metadata (width, height),
 *          mutating the output img pointer data.
 */
#ifndef FILTER_PPM_H_INCLUDED
#define FILTER_PPM_H_INCLUDED

#include "ppm.h"
#include <math.h>

void averaging_filter(const Image img, int x_radius, int y_radius,
        Pixel* extant_img);
void averaging_filter_subset(const Image img, int start_pos, int count,
        int x_radius, int y_radius,
        Pixel* extant_img);

double threshold_filter(const Image img, GPixel* extant_img);
double sum_subset_intensity(const Image img, int start_pos, int count);
void threshold_filter_subset(const Image img, int start_pos, int count,
        double threshold, GPixel* extant_img);


/* Helpers for Subset Filtering
 *  they're only helpers so go ahead and show the implementation for now.
 */

/* how many pixels in a partition? */
inline int img_partition_pixels(int pixel_area, int partitions) {
    return ceil(pixel_area / (float) partitions);
}

/* partition_size == pixels in a partition */
inline int img_partition_head(int partition_size, int kth_subset) {
    return partition_size * kth_subset;
}


#endif // FILTER_PPM_H_INCLUDED
```

src/filter_ppm.c

```c
/* filter_ppm.c
 * ------------
 * Authors: Darwin Jacob Groskleg
 *
 * Purpose: should be agnostic to using MPI.
 */
#include "filter_ppm.h"

#include <assert.h>
#include <stddef.h>


#define min(a,b) \
   ({ __typeof__ (a) _a = (a); \
      __typeof__ (b) _b = (b); \
     _a < _b ? _a : _b; })

/* Provide "external definitions" of the helper functions,
 *  required for proper linking when using C99 inline keyword.
 */
extern inline int img_partition_pixels(int pixel_area, int partitions);
extern inline int img_partition_head(int partition_size, int kth_subset);


/* Averaging / Fuzzy Filter
 *
 * Simplest method for parallel:
 * - (broadcast) send all data to all procs
 * - partition the resulting pixel stream by process number
 * - collect / gather by root, which writes to file
 *
 *
 * Implementation Suggestions:
 *
 * The value for a pixel (x; y) in the output image is the sum of all the pixels
 * in a rectangle in the input image centered around (x; y) divided by the
 * number of pixels in the rectangle.
 *
 * In other words, the pixel in the output image is an average of the
 * rectangle-shaped neighborhood of corresponding pixel in the input image.
 *
 * Your implementation should work with different sizes of the rectangle.
 * Therefore, it is recommended that the function implementing this
 * transformation take parameters specifying the x-radius and y-radius of the
 * rectangle (see Figure 1). It is allowed to use integer limits for the
 * rectangle to avoid the problem of interpolation.
 */
void averaging_filter_subset(const Image img, int start_pos, int count,
        int x_radius, int y_radius,
        Pixel* extant_img)
{
    assert(extant_img != 0);    // memory allocated?

    // radius for averaging
    int xr = x_radius;
```

```
56      int yr = y_radius;
57      assert(xr > 0 && yr > 0);
58
59      int sentinel = min(start_pos + count, img.width * img.height);
60      // k indexing = k'th element is a_ij
61      for (int x,y,k=start_pos; k < sentinel; k++) {
62          // row index = i
63          y = (int) k / img.width;    // integer division
64          // column index = j
65          x = k % img.width;
66
67          int r, g, b, n;
68          n = 0; r = 0; g = 0; b = 0;
69          for (int xas = x - xr; xas < x + xr; xas++) {
70              for (int yas = y - yr; yas < y + yr; yas++) {
71                  Pixel *pix;
72
73                  if (   (xas < 0) || (xas >= img.width)
74                      || (yas < 0) || (yas >= img.height))
75                  {
76                      pix = NULL;
77                  } else {
78                      pix = img.pixels + yas*img.width + xas;
79                  }
80
81                  if (pix != NULL) {
82                      r += pix->red;
83                      g += pix->green;
84                      b += pix->blue;
85                      n++;
86                  }
87              }
88          }
89
90          if (n > 0) {
91              extant_img[k].red   = r/n;
92              extant_img[k].green = g/n;
93              extant_img[k].blue  = b/n;
94          }
95      }
96  }
97
98  /* Averaging Filter
99   *
100  * Purpose: makes the image "fuzzy".
101  */
102 void averaging_filter(const Image img, int x_radius, int y_radius,
103                       Pixel* extant_img)
104 {
105     assert(extant_img != 0);
106
107     // radius for averaging
108     int xr = x_radius;
109     int yr = y_radius;
110     assert(xr > 0 && yr > 0);
111
```

```
112     for (int y = 0; y < img.height; y++) {
113         for (int x = 0; x < img.width; x++) {
114             int r, g, b, n;
115             n = 0; r = 0; g = 0; b = 0;
116             for (int xas = x - xr; xas < x + xr; xas++) {
117                 for (int yas = y - yr; yas < y + yr; yas++) {
118                     Pixel * pix;
119                     if (   (xas < 0) || (xas >= img.width)
120                         || (yas < 0) || (yas >= img.height))
121                     {
122                         pix = NULL;
123                     } else {
124                         pix = img.pixels + yas*img.width + xas;
125                     }
126 
127                     if (pix != NULL) {
128                         r += pix->red;
129                         g += pix->green;
130                         b += pix->blue;
131                         n++;
132                     }
133                 }
134             }
135 
136             if (n > 0) {
137                 extant_img->red = r/n;
138                 extant_img->green = g/n;
139                 extant_img->blue = b/n;
140             }
141             extant_img++;
142         }
143     }
144 }
145 
146 double average_intensity(const Image img) {
147     Pixel* pix;
148     double intensity = 0;
149     for (int y=0; y<img.height; y++) {
150         for (int x=0; x<img.width; x++) {
151             pix = img.pixels + y*img.width + x;
152             intensity += ( (1.0 * (double) pix->red
153                            + 1.0 * (double) pix->green
154                            + 1.0 * (double) pix->blue
155                           )/ 3.0);
156         }
157     }
158 
159     intensity /= img.width * img.height;
160     return intensity;
161 }
162 
163 /* Threshold Filter
164  *
165  * Purpose: creates a binary-pixel image.
166  *
167  * The filter computes the average intensity of the whole input image and use
```

```
 * this value to threshold the image. The result is an image containing only
 * black and white pixels. The white for those pixels in the input image that
 * are lighter than the threshold and the black for those pixels in the input
 * image that are darker than the threshold.
 */
double threshold_filter(const Image img, GPixel* extant_img) {
    assert(extant_img != 0);

    double intensity = average_intensity(img);

    Pixel* pix;
    for (int y=0; y<img.height; y++) {
        for (int x=0; x<img.width; x++) {
            pix = img.pixels + y*img.width + x;
            double pix_intensity = 0;
            pix_intensity = ( 1.0 * (double) pix->red
                            + 1.0 * (double) pix->green
                            + 1.0 * (double) pix->blue
                            ) / 3;
            if ( pix_intensity > intensity) {
                extant_img[y*img.width + x] = 255;
            } else {
                extant_img[y*img.width + x] = 0;
            }
        }
    }
    return intensity;
}

/* for use in a threshold filter
 */
double sum_subset_intensity(const Image img, int start_pos, int count) {
    Pixel* pix;
    double intensity = 0;
    int sentinel = min(start_pos + count, img.width * img.height);
    // k indexing = k'th element is a_ij
    for (int x,y,k=start_pos; k < sentinel; k++) {
        // row index = i
        y = (int) k / img.width;    // integer division
        // column index = j
        x = k % img.width;
        pix = img.pixels + y*img.width + x;
        intensity +=  ((1.0 * (double) pix->red
                      + 1.0 * (double) pix->green
                      + 1.0 * (double) pix->blue
                      )/ 3.0);
    }

    return intensity;
}

/* Threshold Filter
 *
 * Simplest method:
 * - scatter initial pixel stream, partitioned, to procs
 * - calculate avg in each
```

```
 * - Reduce and scatter the result, the average intensity
 * - each proc applies threshold to values
 * - collect / gather by root, which writes to file
 */
void threshold_filter_subset(const Image img, int start_pos, int count,
        double threshold,
        GPixel* extant_img)
{
    assert(extant_img != 0);
    assert(threshold >= 0 && threshold <= 255);

    int sentinel = min(start_pos + count, img.width * img.height);
    // k indexing = k'th element is a_ij
    for (int x,y,k=start_pos; k < sentinel; k++) {
        // row index = i
        y = (int) k / img.width;    // integer division
        // column index = j
        x = k % img.width;

        Pixel *pix = img.pixels + y*img.width + x;
        double pix_intensity = 0;
        pix_intensity = ( 1.0 * (double) pix->red
                        + 1.0 * (double) pix->green
                        + 1.0 * (double) pix->blue
                        ) / 3;
        if ( pix_intensity > threshold) {
            extant_img[y*img.width + x] = 255;
        } else {
            extant_img[y*img.width + x] = 0;
        }
    }
}
```

src/ppm.h

```c
/* ppm.h
 * -----
 * Authors: Darwin Jacob Groskleg, Dr Laurence T. Yang
 * Purpose: interface writing and reading Portable Pix Map (ppm) image format
 *          files, the datatypes to hold it in.
 */
#ifndef PPM_H_INCLUDED
#define PPM_H_INCLUDED

#define PPM_MAX_PIXELS (1000*1000)


/* image.h : MPI agnostic */
typedef struct Pixel {
    unsigned char red, green, blue;
} Pixel;

/* Gray Pixel: 0 (black) – 255 (white) */
typedef unsigned char GPixel;

typedef struct Image {
    /* PPM Body */
    Pixel* pixels;
    /* PPM Header */
    int width;
    int height;
    int max_color;
} Image;


/* ppm.h : Single Threaded Only! */
int  read_ppm(const char * fname, int* xpix, int* ypix, int* max, char* data);
int write_ppm(const char * fname, int  xpix, int  ypix,           char* data);
int write_pgm(const char * fname, int  xpix, int  ypix,           char* data);

#endif // PPM_H_INCLUDED
```

src/ppm.c

```c
/* ppm.c
 * -----
 * Authors: Darwin Jacob Groskleg, Dr Laurence T. Yang
 *
 * Purpose: implementation for writing and reading Portable Pix Map (ppm)
 *          image format files.
 */
#include "ppm.h"

#include <errno.h>
#include <assert.h>
#include <stdio.h>
#include <string.h>

/*
 * Input arguments:
 *      *fname : filename to open
 *
 * Output arguments:
 *      *xpix :
 *      *ypix :
 *      *max  :
 *      data[] :
 */
int read_ppm(const char* fname, int* xpix, int* ypix, int* max, char* data) {
    char ftype[40];
    char ctype[40] = "P6";  // binary data, more efficient than P3
    char line[80];

    FILE * fp;
    errno = 0;

    if (fname == NULL) fname = "\0";
    fp = fopen(fname, "r");
    if (fp == NULL) {
        fprintf(stderr,
                "read_ppm failed to open %s: %s\n", fname, strerror(errno));
        return 1;
    }

    fgets(line, 80, fp);        // first 80 char or until newline
    sscanf(line, "%s", ftype);  // assign string from `line` to ftype[]

    while ( fgets(line, 80, fp) && (line[0] == '#') );  // skip comments

    // grab the pixel width/height numbers from `line`
    sscanf(line, "%d%d", xpix, ypix);

    // grab the next number from the stream,
    // the maximum value on the colour range
    fscanf(fp, "%d\n", max);

/*
    fprintf(stderr, "typ= %s xmax= %d  ymax= %d intensity max= %d\n",
         ftype, *xpix, *ypix, *max);
```

```
56      fflush(stderr);
57  */
58
59      assert(*xpix * *ypix <= PPM_MAX_PIXELS);
60
61      if (strncmp(ftype, ctype, 2) == 0) {
62          size_t pixel_channels = *xpix * *ypix * 3;  // pixels x 3
63          size_t position = fread(data, sizeof (char), pixel_channels, fp);
64          if (position != pixel_channels) {
65              perror("Read failed");
66              fclose(fp);
67              return 2;
68          }
69      } else {
70          fprintf(stderr, "Wrong file format: %s\n", ftype);
71      }
72
73      if (fclose(fp) == EOF) {
74          perror("Close failed");
75          return 3;
76      }
77
78      return 0;
79  }
80
81
82  int write_ppm(const char * fname, int xpix, int ypix, char * data) {
83      FILE * fp;
84      errno = 0;
85
86      if (fname == NULL) fname = "\0";
87      fp = fopen(fname, "w");
88      if (fp == NULL) {
89          fprintf(stderr,
90                  "write_ppm failed to open %s: %s\n", fname, strerror(errno));
91          return 1;
92      }
93
94      fprintf(fp, "P6\n");
95      fprintf(fp, "%d %d 255\n", xpix, ypix);
96      size_t pixel_channels = xpix * ypix * 3;  // pixels x 3
97      if (fwrite(data, sizeof (char), pixel_channels, fp) != pixel_channels) {
98          perror("Write failed");
99          fclose(fp);
100         return 2;
101     }
102     if (fclose(fp) == EOF) {
103         perror("Close failed");
104         return 3;
105     }
106     return 0;
107 }
108
109
110 int write_pgm(const char * fname, int xpix, int ypix, char * data) {
111     FILE * fp;
```

```
112     errno = 0;
113
114     if (fname == NULL) fname = "\0";
115     fp = fopen(fname, "w");
116     if (fp == NULL) {
117         fprintf(stderr, "write_ppm failed to open %s: %s\n", fname,
118                 strerror(errno));
119       return 1;
120     }
121
122     fprintf(fp, "P5\n");
123     fprintf(fp, "%d %d 255\n", xpix, ypix);
124     size_t pixels = xpix * ypix;
125     if (fwrite(data, sizeof (char), pixels, fp) != pixels) {
126         perror("Write failed");
127         fclose(fp);
128         return 2;
129     }
130     if (fclose(fp) == EOF) {
131         perror("Close failed");
132         return 3;
133     }
134     return 0;
135 }
```

src/transform-serial.c

```c
/* transform-serial.c
 * ------------------
 * Derived from: serial.c (L.T. Yang)
 * Authors: Darwin Jacob Groskleg, Dr Laurence T. Yang
 *
 * Usage:
 *      transform-serial [input.ppm]
 * Output:
 *  - file:     input-fuzzy.ppm
 *  - file:     input-threshold.pgm
 *  - stdout:   average intensity calculated for the fuzzy filter.
 */
#include <stdio.h>
#include <errno.h>
#include <assert.h>
#include <stdlib.h>
#include <stddef.h>
#include <string.h>

#include "ppm.h"
#include "filter_ppm.h"

char* append_to_basename(const char* filename, const char* appendix);

int main(int argc, char *argv[]) {
    const char* default_image_filename = "./im1.ppm";
    char* image_filename;
    if (argc > 1) {
        image_filename = (char *) malloc( sizeof (char) * strlen(argv[1]));
        strcpy(image_filename, argv[1]);
    } else {
        size_t default_len = strlen(default_image_filename);
        image_filename = (char *) malloc( sizeof (char) * default_len);
        strcpy(image_filename, default_image_filename);
    }
    printf("infile name: %s\n", image_filename);


    Pixel* in_image = (Pixel *) malloc(PPM_MAX_PIXELS * sizeof (Pixel));
    assert(in_image != 0);

    int err = 0;

    int width, height, max;
    err = read_ppm(image_filename, &width, &height, &max, (char *) in_image);
    if (err != 0) exit(err);
    const Image img = { .pixels=in_image, width, height, max };

/*
 * The "fuzzy" transformation
 *
 */
    Pixel* out_image_fuzzy = (Pixel *) malloc(PPM_MAX_PIXELS * sizeof (Pixel));
    assert(out_image_fuzzy  != 0);

```

```
56      int xradius = 5, yradius = 5;
57      printf("Neighborhood pixel radius |x y|: %d %d\n", xradius, yradius);
58      averaging_filter(img, xradius, yradius, out_image_fuzzy);
59
60      char* fuzzy_fname = append_to_basename(image_filename, "-serial-fuzzy.ppm");
61      err = write_ppm(fuzzy_fname, width, height, (char *) out_image_fuzzy);
62      if (err != 0) exit(err);
63
64
65  /*
66   * The thresholding
67   *
68   */
69      GPixel* out_image_thresh = (GPixel *) malloc(PPM_MAX_PIXELS * sizeof (GPixel));
70      assert(out_image_thresh != 0);
71
72      double intensity = threshold_filter(img, out_image_thresh);
73      printf("Average intensity is: %f\n", intensity);
74      fflush(stdout);
75
76      char* threshold_fname = append_to_basename(image_filename,
77              "-serial-threshold.pgm");
78      err = write_pgm(threshold_fname, width, height, (char *) out_image_thresh);
79      if (err != 0) exit(err);
80
81
82      return 0;
83  }
84
85  /* example:
86   *
87   *  append_to_basename("img.ppm", "-fuzzy.pgm")
88   *      ->"img-fuzzy.pgm"
89   */
90  char* append_to_basename(const char* filename, const char* appendix) {
91      size_t newlen = strlen(filename) + strlen(appendix);
92      char* newfile = (char *) malloc(sizeof (char) * newlen);
93      char* ft_extension = strrchr(filename, '.');
94      size_t basename_len = strlen(filename) - strlen(ft_extension);
95      strncpy(newfile, filename, basename_len); // without the file extension
96      strcat(newfile, appendix);
97      //strcat(newfile, ft_extension);
98      //printf("outfile name: %s\n", newfile);
99
100     return newfile;
101 }
```

Makefile

```
include ../mpi.mk
include ../clean.mk

vpath %.h      ./src
vpath %.c      ./src

clean: clean-ppm-images

all: transform-serial transform-parall test_filter

ppm.o :              ppm.c ppm.h
filter_ppm.o :       filter_ppm.c filter_ppm.h ppm.h

transform-serial.o : transform-serial.c ppm.h filter_ppm.h
transform-serial   : transform-serial.o ppm.o filter_ppm.o

transform-parall.o : transform-parall.c ppm.h filter_ppm.h
transform-parall   : transform-parall.o ppm.o filter_ppm.o

test_filter.o      : test_filter.c ppm.h filter_ppm.h
test_filter        : test_filter.o ppm.o filter_ppm.o

.PHONY:
test-filter: test_filter
    ./$<

.PHONY: test-serial
test-serial: transform-serial
    ./$<

.PHONY: test-parall
test-parall: transform-parall
    $(run-mpi)

.PHONY: test-parall-all
test-parall-all: transform-parall
    $(run-mpi) im1.ppm
    $(run-mpi) im2.ppm
    $(run-mpi) im3.ppm

# Run 4 times, drop the first for disk caching outliers
.PHONY: perf-q1
perf-q1: transform-parall transform-serial
    time ./transform-serial im1.ppm
    time $(MPIRUN) -np 1 ./transform-parall im1.ppm
    @echo -- Reject First, Keep Below ---------------------------------
    time ./transform-serial im1.ppm
    time $(MPIRUN) -np 1 ./transform-parall im1.ppm
    time ./transform-serial im1.ppm
    time $(MPIRUN) -np 1 ./transform-parall im1.ppm
    time ./transform-serial im1.ppm
    time $(MPIRUN) -np 1 ./transform-parall im1.ppm
```

```
53
54 .PHONY: perf-q3
55 perf-q3: transform-parall
56     time $(MPIRUN) -np 1 ./transform-parall im1.ppm
57     @echo -- Reject First, Keep Below --------------------------------
58     time $(MPIRUN) -np 1 ./transform-parall im1.ppm
59     time $(MPIRUN) -np 2 ./transform-parall im1.ppm
60     time $(MPIRUN) -np 4 ./transform-parall im1.ppm
61
62
63 # invoked by :clean
64 .PHONY: clean-ppm-images
65 clean-ppm-images:
66     rm im?-*.ppm im?-*.pgm
67
68 # Merge all 3 to a single before and after image
69 #   requires ImageMagick
70 assets/%-merged.png : %.ppm %-parall-fuzzy.ppm %-parall-threshold.pgm
71     magick montage -geometry 100% $^ $@
72
73 # old ImageMagick command
74 #    convert $^ +append $@
75 # from netpbm-package
76 #    pnmcat -lr <(pngtopnm 1.png) <(pngtopnm 2.png) | pnmtopng > all.png
77
78 project_label:=Lab5-6
79 ordered_docs := questions.md \
80                 src/transform-parall.c \
81                 src/filter_ppm.h   src/filter_ppm.c \
82                 src/ppm.h          src/ppm.c \
83                 src/transform-serial.c \
84                 Makefile
85
86 vpath %.png ./assets
87 document_assets:= im1-merged.png \
88                   im2-merged.png \
89                   im3-merged.png \
90                   Screenshot-serial.png \
91                   Screenshot-parall-all.png
92
93 TITLE:=$(shell cat title.txt)
94 include ../publish.mk
```