Shengchen Zhou 400050783 zhous20 T01

CPU Time

run	reading	processing	writing
ein3A	6.4e-02	4.8e-02	9.6e-02
ein11A	6.0e-02	5.0e-01	5.5e-01
ein5M	6.0e-02	1.2e+00	1.2e+00
lux11A	3.6e+00	2.5e+01	2.8e+01
lux11M			

//main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "filter.h"

extern double getTime();

int main(int argc,char *argv[])
{
    double time0 = getTime();

    int *width, *height, *max;
    RGB *array=readPPM(argv[1],&width,&height,&max);

    double time1 = getTime();

    int n = atoi(argv[3]);
    if(*argv[4]=='A')
    {
        denoiseImage(width,height,array,n,MEAN);
        double time2 = getTime();

    writePPM(argv[2],width,height,max,denoiseImage(width,height,array,n,MEAN));
        double time3 = getTime();
        double n1=time1-time0;
        double n2=time2-time1;
        double n3=time3-time2;
        printf("Input file read in %.1e seconds\n", n1);
        printf("Image processed in %.1e seconds\n", n2);
        printf("Output file written in %.1e seconds\n", n3);
    }
    else if(*argv[4]=='M')
    {
        denoiseImage(width,height,array,n,MEDIAN);
        double time2 = getTime();

    writePPM(argv[2],width,height,max,denoiseImage(width,height,array,n,MEDIAN));
        double time3 = getTime();
        double n1=time1-time0;
        double n2=time2-time1;
        double n3=time3-time2;
        printf("Input file read in %.1e seconds\n", n1);
        printf("Image processed in %.1e seconds\n", n2);
        printf("Output file written in %.1e seconds\n", n3);
    }

    return 0;
}
```

//filter.h

```
#ifndef INCLUDED_FILTER_H
#define INCLUDED_FILTER_H

typedef enum { MEAN, MEDIAN } filter;

typedef struct { unsigned char r, g, b; } RGB;

RGB *readPPM(const char *file, int *width, int *height, int *max);

void writePPM(const char *file, int width, int height, int max, const RGB *image);

RGB *denoiseImage(int width, int height, const RGB *image, int n, filter f);
#endif
```

//read.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "filter.h"
RGB *readPPM(const char *file, int *width, int *height, int *max)
{
  FILE *fd;
```

```c
  char c;
  int red, green, blue, i;

  fd = fopen(file, "r");
  fscanf(fd, "%c%c",&c,&c);
  fscanf(fd, "%d%d%d", width, height, max);

  int size = (*width)*(*height);
  RGB *image = (RGB*)malloc(size*sizeof(RGB));

  for(i=0; i < size; i++)
  {
    fscanf(fd, "%d%d%d", &red, &green, &blue);
    image[i].r = red;
    image[i].g = green;
    image[i].b = blue;
  }
  return image;
}
```

//write.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "filter.h"
void writePPM(const char *file, int width, int height, int max, const RGB *image)
{
  int i;
  FILE *fd;
  fd = fopen(file, "w");

  fprintf(fd, "P3\n");
  fprintf(fd, "%d %d\n%d\n", width, height, max);

  for(i=0; i<height*width; i++)
  {
    const RGB *p = image+i;
    fprintf(fd, "%d %d %d ", p->r, p->g, p->b);
  }
}
```

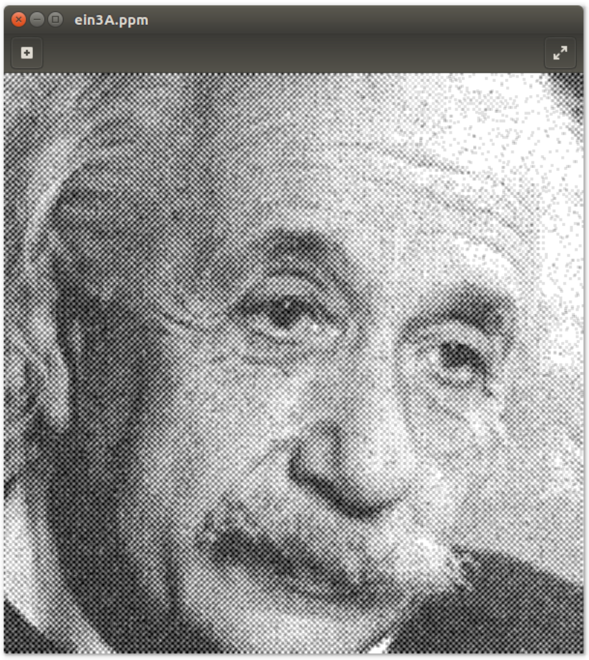
//denoise.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "filter.h"

RGB *denoiseImage(int width, int height, const RGB *image, int n, filter f)
{
    int size = width*height, i, j, k, l;
    RGB *ret = (RGB*)malloc(size * sizeof(RGB));
    if (f == MEAN) {
        for (i = 0; i < height; i++) {
            for (j = 0; j < width; j++) {
                double sumr = 0, sumg = 0, sumb = 0;;
                int count = 0;
                for (k = i - n / 2; k <= i + n / 2; k++) {
                    if (k>=0&&k<height) for (l = j - n / 2; l <= j + n / 2; l++) {
                        if (l >= 0 && l < width) {
                            sumr += image[k*width+l].r;
                            sumg += image[k*width + l].g;
                            sumb += image[k*width + l].b;
                            count++;
                        }
                    }
                }
                ret[i*width + j].r = sumr / count;
                ret[i*width + j].g = sumg / count;
                ret[i*width + j].b = sumb / count;
            }
        }
    }
    else {
        int *r, *g, *b;
        r = (int*)malloc(n*n * sizeof(int));
        g = (int*)malloc(n*n * sizeof(int));
        b = (int*)malloc(n*n * sizeof(int));
        for (i = 0; i < height; i++) {
            for (j = 0; j < width; j++) {
                int count = 0;
                for (k = i - n / 2; k <= i + n / 2; k++) {
                    if (k >= 0 && k<height) for (l = j - n / 2; l <= j + n / 2;
l++) {
                        if (l >= 0 && l < width) {
                            r[count]= image[k*width + l].r;
                            g[count] = image[k*width + l].g;
                            b[count] = image[k*width + l].b;
                            count++;
                        }
                    }
                }
                for (k = 0; k < count; k++) {
                    int min = k;
                    for (l = k + 1; l < count; l++) {
                        if (r[min] > r[l]) {
                            min = l;
                        }
                    }
                    int c = r[k];
                    r[k] = r[min];
                    r[min] = c;
                }
                for (k = 0; k < count; k++) {
                    int min = k;
                    for (l = k + 1; l < count; l++) {
                        if (g[min] > g[l]) {
                            min = l;
                        }
                    }
                    int c = g[k];
                    g[k] = g[min];
                    g[min] = c;
                }
                for (k = 0; k <count; k++) {
                    int min = k;
                    for (l = k + 1; l < count; l++) {
                        if (b[min] > b[l]) {
                            min = l;
                        }
                    }
                    int c = b[k];
                    b[k] = b[min];
                    b[min] = c;
                }
                if (count % 2 == 0) {
                    ret[i*width + j].r = (r[count / 2] + r[count / 2 - 1])/2;
                    ret[i*width + j].g = (g[count / 2] + g[count / 2 - 1]) / 2;
```

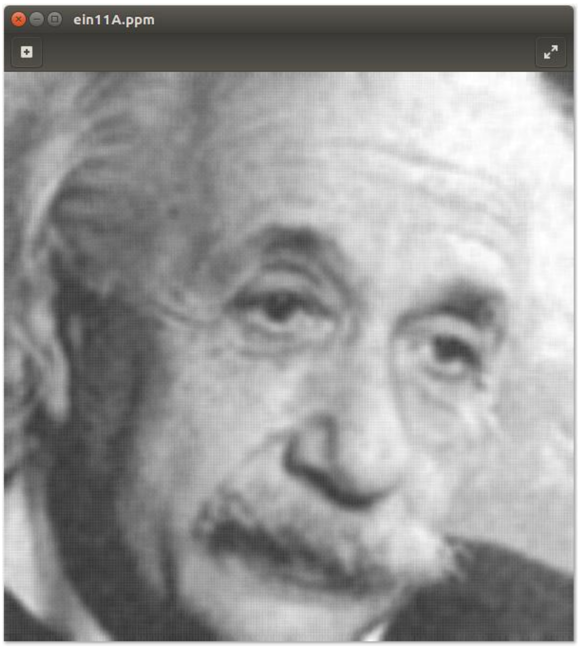
```
                    ret[i*width + j].b = (b[count / 2] + b[count / 2 - 1]) / 2;
                }
                else {
                    ret[i*width + j].r = r[count / 2];
                    ret[i*width + j].g = g[count / 2];
                    ret[i*width + j].b = b[count / 2];
                }
            }
        }

    }
    return ret;
}
```

//makefile

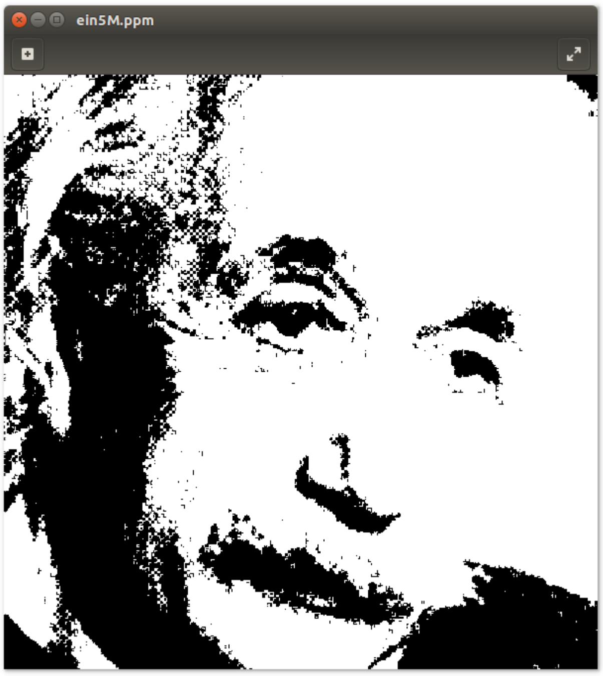
```
CLAGS=-Wall -g -ansi -pg -02 -std=c99
OBJS=main.o denoise.o read.o write.o timing.o
all = denoise

denoise: $(OBJS)
    $(CC) -o $@ $?

clean:
    rm -rf $(OBJS) $(all) *~
```



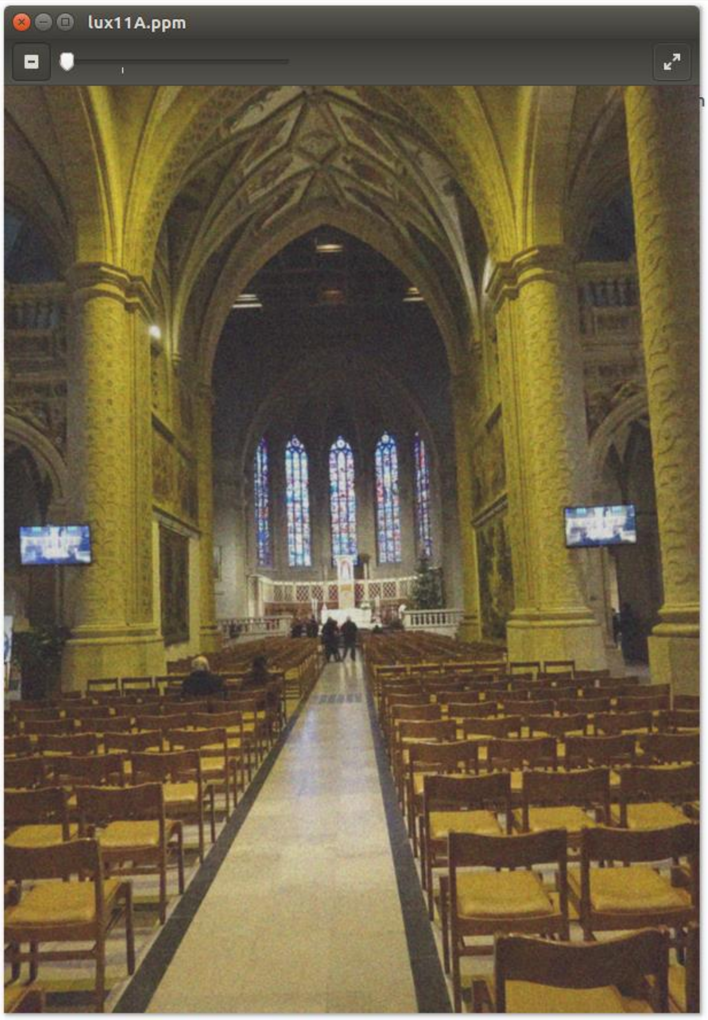
Above is ein3A



Above is ein11A



above is ein5M



above is lux11A

4. Profiling

a)

Processing and writing contributes most of the total execution time(see in write.c and denoise.c above)

In detail, the highlighted part in write.c and nearly whole part(loop part) in denoise.c contributed the exectution time.

b)

Because there are so many pixels in ppm image. And each line, each column needs loops to complete the write and denoise procedure, which takes the computer to do a large amount of assign and compute work. And as loops multiple, it becomes larger and more complex, growing in an incredible scale.