DTS202TC Foundation of Parallel Computing

Group Assignment 1

Group number：Group AG

Group member：

Yaqi Yu19(1930080)

Zihan Tang19(1928732)

Chenxin Zhang19(1928786)

Analyse the performance of your serial program by printing out the execution time of the core algorithm. (5 marks)

1 Serial program

1.1 Execution time of the core algorithm

```
elapsed time: 4.712356
```

1.2 Performance analysis of blur algorithm

1.1.1 The scale of the input data

The blur algorithm is aimed at blurring the image, and the scale of the problem is the size of the image:

$$height * width$$

1.1.2 Elapsed time

The measurement time shown above will be affected by computer performance, so the time complexity is used to measure the unit of the serial blur algorithm.

Our group analyzed the inner and outer two-layer for loops of the algorithm, and considered the scale of the problem: n.

In addition, although the program has multiple iterations of the for loop which contains some judgment statements and constant terms, these factors do not affect the order of magnitude of the time complexity, so the time complexity is:

$$O(n\text{^}2)$$

Provide a design of a solution to speed up using parallel programming. (25 marks)

2 Parallel programming

2.1 The possibility and necessity of parallelization

Traverse in rows and columns and the blurred image can be parallelized. It is feasible to reduce the scale of the problem through parallelization to reduce the time loss.

Next, our group uses Amdahl's law to estimate the upper limit of parallelization speedup. If the $([0,0],[0,width],[high,0],[high,width])$ surrounding the image are parallelized together, then there will be uneven distribution of tasks to each processor. Therefore, the task amount for each column/row is n.

Then the proportion of non-parallelizable tasks is $4n/n*n = 4/n$.

Through Amdahl's law:

$$S = \frac{T_{serial}}{T_{parallel}} = \frac{T_{serial}}{(1-f)T_{serial} + f \cdot \frac{T_{serial}}{p}} = \frac{1}{(1-f) + f \cdot \frac{f}{p}}$$

where $f = \frac{n-4}{n}$, thus speedup will never exceed S.

2.2 Parallel design

According to the Foster method, the following are four parallelization design steps. Since parallelization does not consider IO, the focus of parallelization is the blur algorithm.

Input:

- Dimensions (two-dimensional array): image.height, image.width
- Grayscale: All elements of data[ ][ ]

Output:

- Grayscale: The gray value of the blurred image in the new two-dimensional array new_data[ ][ ]

In order to describe the process more clearly, our group designed the following data structure:

```
struct  PGMstructure
{
    int maxVal;          // Image maximum threshold
    int width;           // Used to record the height of the image
    int height;          // Used to record the width of the image
    int** data;  // Used to store the array to be sorted, data[][] as a temporary variable
    int** new_data;
};
```

2.2.1 Partitioning

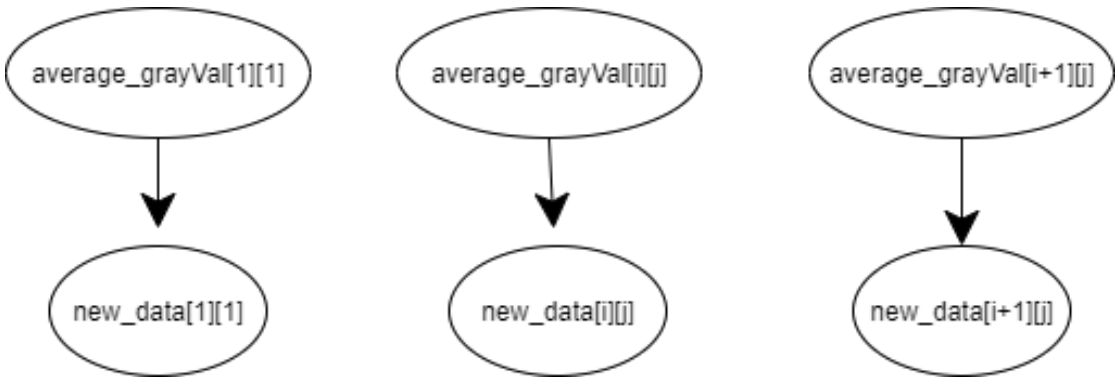
Three categories:

1) **Accumulate the gray value surroundings.**
2) **Average the accumulated values.**
3) **Store in a new array.**

The data that can be divided are all gray values.

2.2.2 Communication

The two tasks in partitioning are to first accumulate the gray value and then store the average gray value into a new array, there is communication between the two.



where average_grayVal and new_data are tasks 2) and 3) in Partitioning above, respectively, and the black arrows represent communication.

In this case, if each gray value update is assigned to each process as a task, many values of the array must be passed to the process that needs to complete the task in each communication, and communication is required each time, which will cause lots of communication overhead and transmission space overhead.

Therefore, using message passing is not a wise choice for distributed memory systems. Our group plans to **use the data[ ][ ] and new_data[ ][ ] as shared variables, which will save communication overhead.**

2.2.3 Agglomeration

Task 2) must be carried out after task 1) is completed, so our team designed the **task 1 and 2 are integrated together.**

Consider data aggregation:

```
for (row = 0; row < hi; row++) {
            for (col = 0; col < wd ; col++) {
                compute average_pixel
            }
}
```

For each process/thread, $local\_col$ and $local\_wd$ are defined, then the tasks and data are redistributed.

The data is allocated to each process/thread according to the number of $col\_numbers$, so that the data obtained by each process/thread is equal.

2.2.4 Mapping

Each process/thread updates the corresponding data in the shared memory new_data[ ][ ] according to the allocated data.

3 Source code

```
#include <stdio.h>
#include <stdlib.h>
#define it 20                    // 20 iterations
#ifndef _TIMER_H_
#define _TIMER_H_

#include <sys/time.h>

/* The argument now should be a double (not a pointer to a double) */
#define GET_TIME(now) { \
    struct timeval t; \
    gettimeofday(&t, NULL); \
    now = t.tv_sec + t.tv_usec/1000000.0; \
}

#endif

struct    PGMstructure
{
      int maxVal;              // Image maximum threshold
      int width;               // Used to record the height of the image
      int height;              // Used to record the width of the image
      int** data;              // Store original grey values array
      int** new_data;          // Store updated grey values array
};
typedef struct PGMstructure PGMImage; // Declare new type names and pointers to character
variables
```

```
void getPGMFile(char fileName[],PGMImage* img);       // Read the pgm file
void blur(PGMImage* image);                            // Use filters to smooth the image
void save(PGMImage* image);                            // Save the pgm file

int main(int argc, char *argv[])
{
    double start, finish, elapsed;
    PGMImage* im = malloc(sizeof(PGMImage));
    // Please change the path according to the specific location of your file
    getPGMFile("C:\\Users\\86133\\Desktop\\Ass_Fin\\im.pgm",&(*im));                    //
Obtain pgm file
    GET_TIME(start);                                   // Timing before blur()
    blur(im);                                          // Perform mean filtering on the input
image src
    GET_TIME(finish);                                  // Timing after blur()
    save(im);                                          // Save the image
    elapsed = finish - start;                          // Runtime
    printf("elapsed time: %.6f\n", elapsed);
    return 0;
}

void getPGMFile(char fileName[],PGMImage* img){
    FILE* in_file;
    char version[100];
    int row,col,i;
    // Open a binary file, only allow reading and writing data
    in_file = fopen(fileName,"rb");

    /*File none error*/
```

```
if (in_file == NULL)
{
        fprintf(stderr, "Error: Unable to open file %s\n\n", fileName);
        exit(1);
}
printf("\nReading image file: %s\n", fileName);



/*Read image information*/
fscanf(in_file,"%s",version);
fscanf(in_file,"%d",&(img->width));
fscanf(in_file,"%d",&(img->height));
fscanf(in_file,"%d",&(img->maxVal));

/*Show image information*/
printf("\n%s", version);
printf("\n width    = %d",img->width);
printf("\n height = %d",img->height);
printf("\n maxVal = %d",img->maxVal);
printf("\n");



/*Allocate dynamic array memory*/
img->data = (int**) malloc(sizeof(int*)* img->height);
img->new_data = (int**) malloc(sizeof(int*) * img->height);
for (i = 0; i < img->height; i++) {
        img->data[i] = (int*) malloc(sizeof (int)* img->width);
        img->new_data[i]=(int*) malloc(sizeof(int)* img->width);
}
```

```
	/*Read image data*/
	for (row = 0; row < img-> height ; row++) {
		for (col = 0; col < img->width; col++) {
			fscanf(in_file,"%d",&(img->data[row][col]));
		}
	}

	fclose(in_file);
	printf("\nDone read file information and data\n");

}

// Blur the picture
void blur(PGMImage* image){
	int row,col;
	// Define temporary variables to store processed data
	int temp;
	int k;
	int hi = image->height;
	int wd = image->width;
	// Repeat the blurring process 20 times
	for (k = 0; k < it; k++) {
		// Image boundary processing
		// update four corners of the image
		image->new_data[0][0] = (int)
(image->data[1][0]+image->data[0][1]+image->data[1][1])/3;
		image->new_data[0][wd-1]=(int) (image->data[0][wd-2]+image->data[1][wd-
1]+image->data[1][wd-2])/3;
		image->new_data[hi-1][0] = (int)(image->data[hi-1][1]+image->data[hi-
```

```
2][0]+image->data[hi-2][1])/3;
        image->new_data[hi-1][wd-1]=(int)(image->data[hi-1][wd-2]+image->data[hi-
2][wd-1]+image->data[hi-2][wd-2])/3;
        for (row = 0; row < hi; row++) {
            for (col = 0; col < wd ; col++) {
                /*update the first row*/
                if(row==0&&(col!=0 && col!=(wd-1))){
                    temp = image->data[row][col-1]+
                            image->data[row][col+1]+
                            image->data[row+1][col-1]+
                            image->data[row+1][col]+
                            image->data[row+1][col+1];
                    image->new_data[row][col]=(int) temp/5;
                }
                /*update the last row*/
                if(row==(hi-1)&&(col!=0 && col!=(wd-1))){
                    temp =   image->data[row-1][col-1] +
                             image->data[row-1][col] +
                             image->data[row-1][col+1] +
                             image->data[row][col-1] +
                             image->data[row][col+1];
                    image->new_data[row][col]=(int) temp/5;
                }
                /*update the first column*/
                if((row!=0 && row!=(hi-1))&&(col==0)){
                    temp = image->data[row-1][col]+
                            image->data[row+1][col]+
                            image->data[row-1][col+1]+
                            image->data[row][col+1]+
                            image->data[row+1][col+1];
```

```
                image->new_data[row][col]=(int) temp/5;
            }
            /*update the last column*/
            if((row!=0 && row!=(hi-1))&&(col==(wd-1))){
                temp = image->data[row-1][col]+
                        image->data[row+1][col]+
                        image->data[row-1][col-1]+
                        image->data[row][col-1]+
                        image->data[row+1][col-1];
                image->new_data[row][col]=(int) temp/5;
            }
            /*update other data*/
            if(row!=0 && row!=(hi-1) && col!=0 && col!=(wd-1)){
                temp =    image->data[row-1][col-1] +
                          image->data[row-1][col] +
                          image->data[row-1][col+1] +
                          image->data[row][col-1] +
                          image->data[row][col+1] +
                          image->data[row+1][col-1] +
                          image->data[row+1][col] +
                          image->data[row+1][col+1];
                image->new_data[row][col] = (int) temp/8;
            }
        }
    }

    /*each iterations update data array*/
    for (row = 0; row < hi; row++) {
        for (col = 0; col < wd; col++) {
            image->data[row][col] = image->new_data[row][col];
```

```
                }
            }
        }
}
// Save the blurred image
void save(PGMImage* image){
        FILE* ou_File;
        int row, col;
        int hi = image->height;
        int wd = image->width;
        int maxV = image->maxVal;

        // Open and name the file
        // Please change the path according to the specific location of your file
        ou_File = fopen("C:\\Users\\86133\\Desktop\\Ass_Fin\\im-blur.pgm", "w+");

        // Print header information of pgm file
        fprintf(ou_File,"P2\n");
        fprintf(ou_File,"%d %d\n",wd,hi);
        fprintf(ou_File,"%d \n",maxV);
        // Write blurred image data
        for (row = 0; row < hi; row++) {
            for (col = 0; col < wd; col++) {
                fprintf(ou_File,"%d ",image->new_data[row][col]);
            }
            fprintf(ou_File,"\n");
        }
        // Close the pgm file
        fclose(ou_File);
}
```