Using a parallel computer for simple image denoising

Xing Cai

February 23, 2012

This is the first mandatory assignment of INF3380. Each student should work independently and submit as result an amply-commented MPI program that implements the required computations (to be described below).

1 Introduction

Image denoising refers to the removal of noises from a noise-contaminated image, such that the "smoothed" image more closely resembles the original noise-free image. Since noisy images are present in many real-life situations, image denoising has become an important task in modern use of computers. An example of image denoising is illustrated in Figure 1.

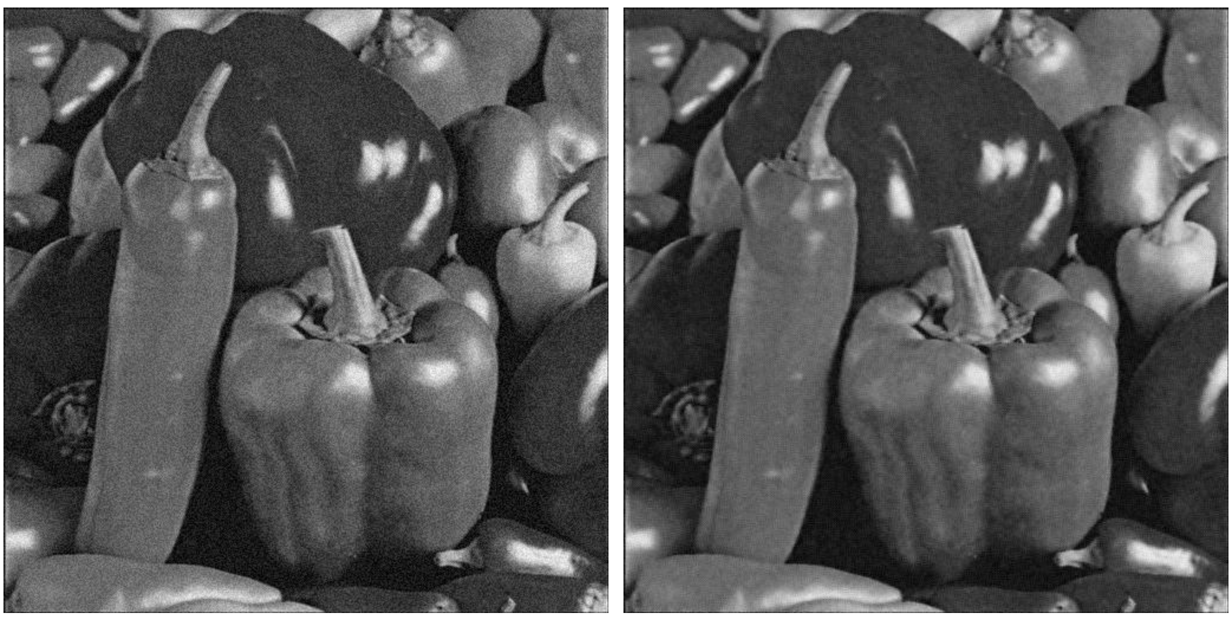

Figure 1: Left: a noisy image; Light: a denoised image after applying a simple diffusion operator.

The purpose of this assignment is to let the students get familiarized with the following important aspects:

1. Translation of simple mathematical formulas to a working code.
2. Compilation of existing C source codes into an external library.
3. Parallelization of a simple denoising algorithm via MPI programming.

2 Numerical algorithm

An image can be thought as a 2D array, containing $m \times n$ pixels,

$$\mathbf{u} = \begin{bmatrix} u_{m-1,0} & u_{m-1,1} & \cdots & u_{m-1,n-1} \\ \vdots & \vdots & \vdots & \vdots \\ u_{1,0} & u_{1,1} & \cdots & u_{1,n-1} \\ u_{0,0} & u_{0,1} & \cdots & u_{0,n-1} \end{bmatrix}.$$

A very simple denoising strategy is to apply the so-called *isotropic diffusion* a number of times. During each iteration of isotropic diffusion, we want to compute a "smoothed" version of $\mathbf{u}$ and denote it by $\bar{\mathbf{u}}$. More specifically, we can compute $\bar{u}_{i,j}$ using the following formula:

$$\bar{u}_{i,j} = u_{i,j} + \kappa\left(u_{i-1,j} + u_{i,j-1} - 4u_{i,j} + u_{i,j+1} + u_{i+1,j}\right).$$

Here, κ is typically a small constant (such as 0.1). Note that the above formula is used to compute the interior pixels of $\bar{\mathbf{u}}$, that is, $1 \leq i \leq m-2$ and $1 \leq j \leq n-2$. The boundary pixels of $\bar{\mathbf{u}}$ should simply copy the corresponding boundary pixels of $\mathbf{u}$.

Note. Before carrying out a new isotropic diffusion iteration, the newly computed pixel values of $\bar{\mathbf{u}}$ should be copied back into $\mathbf{u}$.

3 Using an external library for reading/writing JPEG images

In order to be able to read in a JPEG image before doing denoising as described above, we will make use of some existing serial C codes (non-MPI). More specifically, the following tar file

```
http://heim.ifi.uio.no/xingca/inf-verk3830/simple-jpeg.tar.gz
```

should be downloaded and packed out in a separate directory. This is actually an external C library package containing a set of header (`*.h`) files and C (`*.c`) files.

It is very important to compile all the `*.c` files and group the resulting object `*.o` files into a static library file (e.g. with name `libsimplejpeg.a`). Hint: the `ar` command should be used to generate `libsimplejpeg.a` based on all the `*.o` files.

In particular, the following two functions from `libsimplejpeg.a` will be used later:

```
void import_JPEG_file (const char* filename, unsigned char** image_chars,
                       int* image_height, int* image_width,
                       int* num_components);
void export_JPEG_file (const char* filename, const unsigned char* image_chars,
                       int image_height, int image_width,
                       int num_components, int quality);
```

These two functions can be used, respectively, to read and write a data file of the JPEG format. We remark that each pixel in a grey-scale JPEG image uses one byte, and a one-dimensional array of `unsigned char` (of total length mn) is used to contain all the pixel data of a grey-scale JPEG image. (In the case of a color JPEG image, a 1D array of `rgbrgbrgb`... values is read in.)

Moreover, the integer variable `num_components` will contain value 1 after the `import_JPEG_file` function finishes reading a grey-scale JPEG image. Value `1` should also be given to `num_components` before invoking `export_JPEG_file` to export a grey-scale JPEG image. (For a color JPEG image, the value of `num_components` is `3`.) We also remark that the last argument `quality` of the `export_JPEG_file` function is an integer indicating the compression level of the resulting JPEG image. A value of 75 is the typical choice of `quality`.

4 Data structure

It should be noted that a 1D array of type `unsigned char` is used by `libsimplejpeg.a` for reading and writing a JPEG image. A variable of type `unsigned char` always has an integer value between 0 and 255. This is not sufficient for doing accurate denoising computations. To this end, the following data structure can be used to store the $m \times n$ pixel values, in connection with denoising:

```
typedef struct
{
  float** image_data;  /* a 2D array of floats */
  int m;               /* # pixels in x-direction */
  int n;               /* # pixels in y-direction */
}
image;
```

Note. A conversion between `unsigned char` and `float` is needed when copying the pixel values between an object of type `image` and a 1D array of type `unsigned char`.

5 Three functions need to be implemented

```
void allocate_image(image *u, int m, int n);
void deallocate_image(image *u);
void iso_diffusion_denoising(image *u, image *u_bar, float kappa, int iters);
```

It should be clear that function `allocate_image` is supposed to allocate the 2D array `image_data` inside `u`, when `m` and `n` are given as input.

The purpose of function `deallocate_image` is to free the storage used by the 2D array `image_data` inside `u`.

The most important function that needs to be implemented is `iso_diffusion_denoising`, which is supposed to carry out `iters` iterations of the isotropic diffusion on a noisy image object `u`. The denoised image is to be stored in the `u_bar` object. Moreover, function `iso_diffusion_denoising` is supposed to be carried out by P MPI processes in collaboration. That is, the `u` and `u_bar` objects on each MPI process actually cover one rectangular region of a large image, which is partitioned and distributed to P processes. In other words, after each iteration of isotropic diffusion, data exchange (using MPI commands) is needed between neighboring MPI processes.

6 Skeleton of the main program

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

// make use of two functions from the simplejpeg library
void import_JPEG_file(const char *filename, unsigned char **image_chars,
                      int *image_height, int *image_width,
                      int *num_components);

void export_JPEG_file(const char *filename, unsigned char *image_chars,
                      int image_height, int image_width,
                      int num_components, int quality);

int main(int argc, char *argv[])
{
  int m, n, c, iters;
  int my_m, my_n, my_rank, num_procs;
  float kappa;
  image u, u_bar;
  unsigned char *image_chars;
  char *input_jpeg_filename, *output_jpeg_filename;

  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
  MPI_Comm_size (MPI_COMM_WORLD, &num_procs);

  /* read from command line: kappa, iters, input_jpeg_filename, output_jpeg_filename */
  /* ... */

  if (my_rank==0)
    import_JPEG_file(input_jpeg_filename, &image_chars, &m, &n, &c);

  MPI_Bcast (&m, 1, MPI_INT, 0, MPI_COMM_WORLD);
  MPI_Bcast (&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

  /* divide the m x n pixels evenly among the MPI processes */
  my_m = ...;
  my_n = ...;

  allocate_image (&u, my_m, my_n);
  allocate_image (&u_bar, my_m, my_n);

  /* each process asks process 0 for a partitioned region */
  /* of image_chars and copy the values into u */
  /*  ...  */

  iso_diffusion_denoising (&u, &u_bar, kappa, iters);

  /* each process sends its resulting content of u_bar to process 0 */
  /* process 0 receives from each process incoming values and */
  /* copy them into the designated region of image_chars */
  /*  ...  */

  if (my_rank==0)
    export_JPEG_file(output_jpeg_filename, image_chars, m, n, c, 75);

  deallocate_image (&u);
  deallocate_iamge (&u_bar);

  MPI_Finalize ();
```

```
  return 0;
}
```

7 A test case

The left noisy picture of Figure 1 can be used as a test case. The JPEG image file is downloadable from

```
http://heim.ifi.uio.no/xingca/inf-verk3830/noisy-paprika.jpg
```