CS 133 Spring 2013 Project

(Due electronically to Courseweb by June 7th, 11:59pm)

For the course project, you will work in groups of three students to implement a small pipeline of three basic medical imaging applications. You are responsible to choose teammates. Each student is asked to implement one of the three applications in the following four languages:

- OpenMP
- CnC (Concurrent Collection)
- MPI
- OpenCL

Each team needs to cover the entire medical image processing pipeline. The sequential program for each application in the pipeline will be given to you, as well as a set of 2D black-and-white input pictures. After implementing the application in all four platforms, you should discuss the performance optimizing techniques you used for each platform as part of the team report. Each group should provide a single report of all three applications in the pipeline with discussions of each application in the pipeline (provided by individuals) and the comparison between all four platforms.

Applications

For each application in the pipeline, you will be given a sequential version of the kernel function. Your task is to implement the parallel version of this function use the four parallel frameworks/platforms. Your kernel functions should carry out the same result as the sequential version.

The sequential code of the given medical imaging pipeline is in the following directory of cs133.seas.ucla.edu.

```
/usr/local/cs133/project
```

The corresponding main function including the input/output utilities for each application is provided to you to test the correctness of your kernel functions.

1. 2D MRI Reconstruction

The data obtained from the MRI machine contains the spectrum of the image obtained during the imaging process. The input data in our medical imaging pipeline has reduced sample rate, which means the spectrum input is sampled from the original data. To reconstruct the image with incomplete data, we use a compressive sensing algorithm that optimizes the reconstructed image quality with a constraint of total variation.

The algorithm used is Split Bregman method to optimize reconstructed image quality with an error constraint. The details of the algorithm can be found in the following paper:

Goldstein, T. and Osher, S., "The Split Bregman Method for L1-Regularized Problems", SIAM Journal on Imaging Sciences 2009 2:2, 323-343

The input data will be read from files of a given format, and stored as an array of complex numbers. A complex struct consists of two floating-point numbers representing the real and imaginary part of that complex number. The function of this application is specified below:

```
int mri_recon2d(float *img, complex* f, float* mask, int n1, int n2);
```

`img:`	Array of size n1*n2, stores output image as floating-point numbers.
`f:`	Array of size n1*n2, stores MRI spectrum input obtained from the MRI machine
`mask:`	Array of size n1*n2, stores the mask to select valid spectrum. Some spectrum data is lost in under-sampling, this is done be masking the corresponding spectrum element as 0.
`n1, n2`	Image size, row and column respectively
`return:`	0 by default

The code used to parse the input data and write to BMP image is given, please refer to the main() function of the sequential benchmarks. The input and output format of this application is as follows:

```
$ ./mri <input_file> <output_file>
```

2. Image de-noise

The functionality of the image de-noise is reducing the noise of the image reconstructed by the MRI reconstruction stage as well as applying image smoothing. The algorithm used in this stage is Rician de-noise method that models the pixels of the noisy image as i.i.d. Rician distributed, and optimize the smoothness of the image using total variation. The details of the algorithm can be found in the following paper:

Rudin, L. I.; Osher, S.; Fatemi, E. (1992). "Nonlinear total variation based noise removal algorithms". Physica D 60: 259–268.

The input image is read from a BMP file, and stored as normalized floating-points ranged from 0 to 1. The function of this pipeline stage is specified below:

```
int riciandenoise(float *u, const float* f, int n1, int n2);
```

`u:`	Array of size n1*n2, stores output image as floating-point numbers.
`f:`	Array of size n1*n2, stores input image from MRI reconstruction
`n1, n2`	Image size, row and column respectively
`return:`	0 by default

The code used to read and write BMP image is given, please refer to the main() function of the sequential benchmarks. The input and output format of this application is as follows:

```
$ ./denoise <input_file> <output_file>
```

3. Image segmentation

The functionality of the image segmentation is identifying the significant regions in the image. In the medical imaging context, this stage is useful to identify the critical region such as brain lobes or blood vessel nodules. The algorithm used in this stage is Active Contour method that iteratively adapts region contour to the part of the image with maximum variation. The details of the algorithm can be found in the following paper:

Chan, Tony F., and Luminita A. Vese. "Active contours without edges." Image Processing, IEEE Transactions on 10.2 (2001): 266-277

The input image is read from a BMP file, and stored as floating-points ranged from 0 to 255. The output of the function is a region identifier called ϕ. This output is then passed to another function that identifies the contour and store the image as a BMP file. The function of this pipeline stage is specified below:

```
int segmentation(float *phi, const float* u, int n1, int n2);
```

`phi:`	Array of size n1*n2, stores output regions of image
`u:`	Array of size n1*n2, stores input image from de-noise pipeline
`n1, n2`	Image size, row and column respectively
`return:`	0 by default

The code used to read and write BMP image is given, please refer to the main() function of the sequential benchmarks. The input and output format of this application is as follows:

```
$ ./seg <input_file> <output_file> <Max_Iter>
```

The input `Max_Iter` determines how many iterations the segmentation algorithms is going to run. This parameter affects the quality of the generated contour.

Requirements

1. Program code

Each group should provide one package that includes all implementations of each application in the pipeline. For the details on the organization of the code, please see "What to turn in" section later in this document. Each team member should make sure the submitted code compiles and follows the input/output format. The implementations of an application should be compiled to separate executable files, which means you need to provide separate `main()` functions for each implementation.

2. Correctness

Program correctness should be the first priority of your implementation. Without correctness, the performance of your implementation will not be considered. Since the outputs of all three pipeline applications are 2D images, the correctness of each of the applications are judged by image quality. The general rule of correctness is:

The parallel implementation should produce the same image as the sequential benchmarks, for all possible input.

However, due to potential numerical errors of floating-point numbers, the value of each pixel need not be exactly the same. As long as the difference is not visualize-able, the results are considered to be correct.

3. Performance

The performance of each implementation is measured of the entire program, using the Linux "time" command.

In your project report, the performance results you used for discussion or justification of your implementations can be measured by whatever means you preferred. However, the performance for grading will be measured by us, with zero system workload. The grading for performance part is based on the rank in the class of each application and its implementations. For example, if your CnC implementation of denoise ranks the first in the class but your MPI

implementations ranks the last, you will get full credit for your CnC implementation but no credit for your MPI implementation.

4. Project Write-up

Each group should submit a project write-up report together with the implementation codes. The report contains both individual reports as well as the team report. The report should contain the following sections:

1. Task division. Please identify in this part which pipeline stage is implemented by which group member. We will be grading the individual reports based on the task division.
2. Individual sections for each pipeline applications.

   This part should be completed by each member of the group individually, and combined in the team report.

   The individual report should include the following parts:

   1) Algorithm analysis and program profiling: identify the performance bottleneck of the application, propose the methodology for parallelize the program such as data/task distribution and reduction method used.
   2) Implementation for each framework/platform. In this part please discuss the implementation details of your parallel program. Please identify the similarity as well as the difference in your implementation with each parallel framework/platform.
   3) Experimental results. This part should include the performance comparison between the sequential program and your parallel program implemented by each parallel framework/platform. You should also discuss the performance optimizations you did specifically for each framework/platform, and present the related results.
3. Team report

   This section includes the comparison of each parallel framework/platform regarding to different applications. The team should provide discussions on the advantages and disadvantages of each parallel framework/platform. The group section may also include the team's experience in the project implementation.

Milestones

Time	Milestones
4/22	Project assignment
4/24	Group forming, team task scheduling due online
4/26	Sequential code profiling, verify environment settings, report problems during discussion session
5/1	Algorithm understanding, parallelization proposal
5/8	Milestone: OpenMP implementation
5/15	Milestone: CnC implementation
5/22	Milestone: MPI implementation
5/29	Milestone: OpenCL implementation
6/7	Project code and report due

What to turn in

Submit to Courseweb a single tarball (with extension .tar.gz or .tgz) containing a directory named project

Inside this directory should reside the following folders and files:

- `./mri`
    - `./mri/omp` – OpenMP implementation of MRI image reconstruction
    - `./mri/cnc` – CnC implementation
        - under this folder please include a *.cnc file that is the graph description
        - ./mri/cnc/inputs – please provide all the necessary step/main files (*.hc) and corresponding Makefile, so that when overwrite the files the cnc translator generated, your program can be compiled to performance correct functionality.
    - `./mri/mpi` – MPI implementation
    - `./mri/ocl` – OpenCL implementation
- `./denoise` – following the same subfolder organization as mri implementation
- `./seg` – following the same subfolder organization as mri implementation
- `./` -- root directory
    - a PDF file named "`report.pdf`", which is the group report following the requirements
    - a TXT file named "`README`", in which please explain the necessary information on how to compile all the codes, especially if you used specialized libraries.

Grading Policy

- Individual portion: 85%
    - Meeting all milestones: 10%
    - Implementation: 55%
        - Source code meets format requirements, compiles correctly: 20%
        - Results correctness: 20%
        - Performance: 15%
    - Project write-up: 20%
        - Justification of parallel strategies: 5%
        - Implementation details: 10%
        - Performance comparison and analysis: 5%
- Group report: 15%
        - Discuss the pros and cons of each parallel programming language and rank the four languages from most favorite to least favorite