HW2: SIFT

Due: Thu., 2025/10/17 23:59

Problem Description

In this assignment, you are asked to implement the simplest SIFT (Scale-Invariant Feature Transform) algorithm to extract features from images and parallelize your program with MPI and OpenMP libraries. We hope this assignment helps you to learn the following concepts:

- The differences between Static Scheduling and Dynamic Scheduling.
- The importance of Load Balancing and Arithmetic Intensity.
- The importance of parallel algorithms and libraries.

Please note that you are not allowed to use mathematical techniques to bypass or reduce the computational complexity and the iteration process of calculating SIFT (i.e., do not modify the parameters in the sample code).

```
//****************************************
// SIFT algorithm parameters, used by default
//****************************************

// digital scale space configuration and keypoint detection
const int MAX_REFINEMENT_ITERS = 5;
const float SIGMA_MIN = 0.8;
const float MIN_PIX_DIST = 0.5;
const float SIGMA_IN = 0.5;
const int N_OCT = 8;
const int N_SPO = 5;
const float C_DOG = 0.015;
const float C_EDGE = 10;

// computation of the SIFT descriptor
const int N_BINS = 36;
const float LAMBDA_ORI = 1.5;
const int N_HIST = 4;
const int N_ORI = 8;
const float LAMBDA_DESC = 6;

// feature matching
const float THRESH_ABSOLUTE = 350;
const float THRESH_RELATIVE = 0.7;
```

Environment Setup and Sequential Code

Please note that the sample test cases and programs might contain bugs.
If you spotted a potential one and are not quite sure about it, please raise a question on NTU COOL.

1. Copy the folder with the provided materials with the following command:

```
cp -r /work/b10502076/pp25/hw2 .
```

2. Create a Miniconda Environment for validation:
(The environment needs to be created only once)

```
module load miniconda3
conda create -n hw2 python=3.12 -y
conda activate hw2
pip install --upgrade pip
pip install opencv-python
pip install scikit-image
```

Note: You can run source scripts/conda.sh directly that contains all the commands above.

3. Load the required modules for the program:
(You need to set the environment each time you open a new terminal)

```
module purge
module load miniconda3
conda activate hw2

module load gcc/13
module load openmpi
export UCX_NET_DEVICES=mlx5_0:1
```

Note: You can run source scripts/env.sh directly that contains all the commands above.

4. Compile and execute the example code:

```
cd ~/hw2/
make
srun -A ACD114118 -N 2 -n 4 -c 4 --time=00:03:00 ./hw2 ./testcases/xx.jpg
./results/xx.jpg ./results/xx.txt
```

Note:

1. The default Makefile for this homework is provided at /work/b10502076/pp25/hw2/Makefile. If you wish to change the compilation flags, please include the Makefile in your submission.
2. Remember to set the time limit when you submit your slurm job.

5. Validate the output image

```
python3 validate.py ./results/xx.txt ./goldens/xx.txt ./results/xx.jpg
./goldens/xx.jpg
Pass / Wrong
```

Output Verification

You can use `hw2-judge` in the directory that contains your .cpp / .h / .hpp files & Makefile to check the correctness of your program on all test cases and submit your result to the scoreboard http://140.112.187.55:7771/scoreboard/hw2/. The time limit for each input test case is 45 seconds.

If you did not submit your result to the scoreboard in HW1, run the following command. Otherwise, you don't need to run it again.

```
bash /work/b11902043/PP25/setup.sh
source ~/.bashrc
```

Please note that your homework is graded based on the code submitted to NTU Cool, the statistics on the scoreboard will not be considered, and also remember to load the modules mentioned above before using the auto-judge.

Report

Your report must contain the following contents, and you can add more as you like. You can write in either English or Traditional Chinese:

1. Name and Student ID

2. Explain your implementation, especially in the following aspects:
    - How do you partition the task?
    - What scheduling algorithm did you use: static, dynamic, guided, etc.?
    - What techniques do you use to reduce execution time?
    - Other efforts you make in your program.
    - What difficulty did you encounter in this assignment? How did you solve them?
3. Analysis:
    - Design your own plots to show the load balance of your algorithm between threads/processes.
    - Analyze your program's scalability in the following aspects:
        - number of nodes
        - number of processes per node
        - number of CPU cores per process
    - Other things worth mentioning.
4. Conclusion:
    - What have you learned from this assignment?
    - (Optional) Any feedback or suggestions for this assignment or spec.

Grading

1. (30%) Correctness. Proportional to the number of test cases solved.
    - Additional hidden test cases will be used to test your program.
    - If the output PNG file of your program fails to match 98% of the ground truth, you will get 0 points for that test case.
2. (40%) Performance.
    - Your score is based on the total time you take to solve all the test cases, relative to the fastest time in the class.
    - You are required to produce the right answer to get the performance points.
3. (30%) Report.

Submission

Please zip the following files to a single archived file named <studentID>.tar:

- All `.hpp / .h / .cpp files` – the source code of your implementation.
- `report.pdf` – your report.
- `Makefile` – optional. Submit this file if you want to change the build command.

Please note that the first character of your student ID in the file name MUST be lowercase, and you should run the archiving command on the directory (also in lowercase), not the files themselves. Any submission violation would lead to a point deduction.

For instance, the correct way to archive your files is as follows:

- `mkdir b10000000`
- `cp <your files> b10000000/`
- `tar cvf b10000000.tar b10000000`
- Submit the file `b10000000.tar` to NTU COOL