MPCS-52060 – Project 1

Image Processing System Performance Experiment Report

1. Introduction

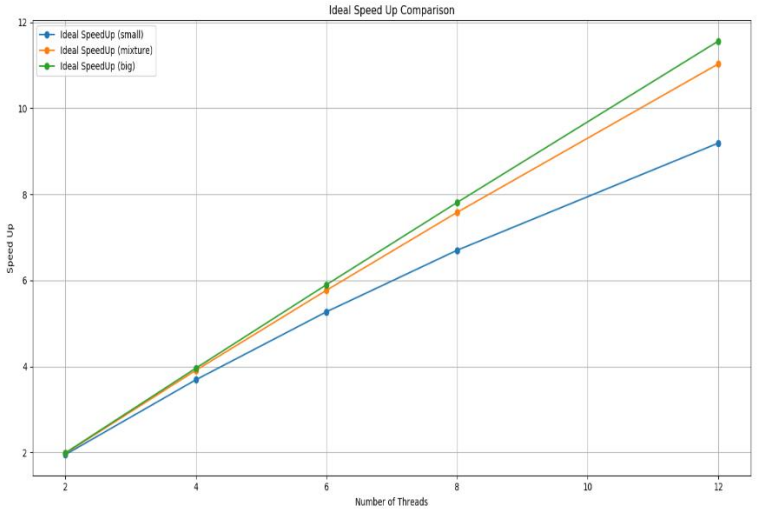
In this project, we created an image processing system, which can support four effects on the images: Blur, Edge Detection, Grayscale, and Sharpen. All the effects except the grayscale is implemented using the convolution calculation. We implemented three different versions for this image processing system. The basic one is the sequential version. In this version, all the images are processed sequentially without any parallel design. The second version is called “parfiles”. In this version, we used a queue structure to manage the task of various images. We implement parallel design to processing the different images concurrently to decrease the processing time. We used self-implemented ATS lock to prevent unexpected race condition caused by parallel design. In the third and last version called “parslices”, we used the same queue structure of the second version and followed the other deisgns. Instead of processing each image using one thread, we reprocess the images to different slices and process the slices concurrently. In this version, all effects are processed sequentially so that no image will be applied on the second effect if the first effect is not finished by all threads. Also, we used wait groups to ensure that the next image is processed only when the current image processing is finished. This performance report is based on these three different implmentations. In addition, we also used three image sets with same content but different sizes for this performance experiment. The images are classified as small, mixture, and large.

2. Experiment Design

To test the performance of the image processing system, we performed seven times of experiment by running the benchmark-proj1.sh test script: we observed and recorded the best result among the seven data output. To run the test script, simply using the command: [sbatch benchmark-proj1.sh]. The main test logic is included in the performanceTest.py located in benchmark directory. For each experiment, the test script will run small, mixture, and big images in “parfiles” and “parslices” mode using thread numbers two, four, six, eight, and twelve. After all the experiment completed. All the performance data are recorded and included in the generated graph as the experiment output by the test script. To obtain and calculate the ideal speed up for the experiments, we recorded the sequential execution time and sequential part execution time of the program (the sequential part execution time is the execution time of the part of program before entering the parfiles.go or parslices.go). This part of the program is expected to be fully sequential. We assume that the “parfiles.go” or “parslices.go” part is fully parallelized and obtain the parallel fraction using the following formula:

$$\frac{\text{sequential } time - sequential\ part\ time}{\text{sequential time}}.$$

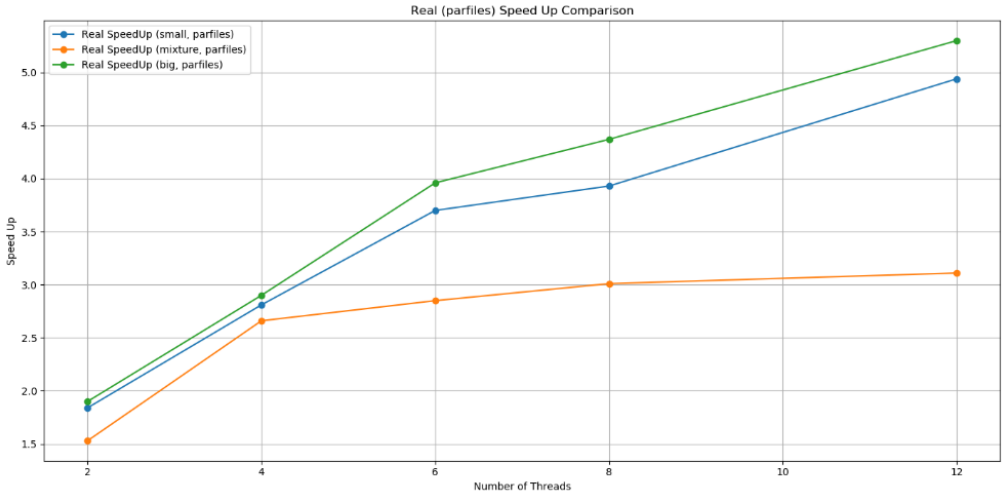
We then used Amdahl's law $\frac{1}{1-p+\frac{p}{n}}$ (for p is parallel fraction, n is thread numbers) to calculate the ideal speed up based on the parallel fraction obtained and thread numbers. All these steps are implemented automatically by the testing script. The following graph and table show the ideal speed up calculated for different experiment cases.

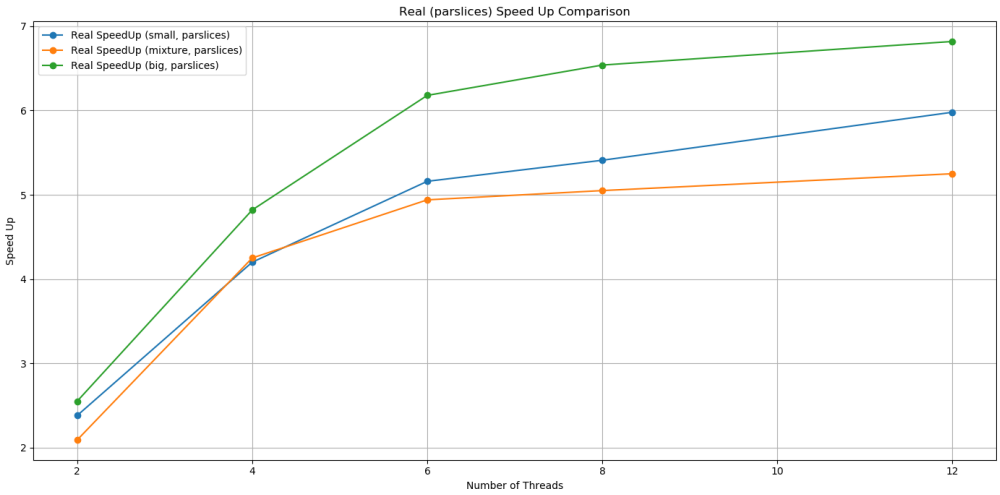


THREAD NUMBERS	IDEAL SPEED UP SMALL	IDEAL SPEED UP MIXTURE	IDEAL SPEED UP BIG
2	1.98	2.01	2.03
4	3.86	3.99	4.02
6	5.73	5.91	5.99
8	6.79	6.92	7.93
12	9.59	11.21	11.54

3. Experiment Result

The experiment result of the "parfiles" mode cases are showed in the following graphs. From the graphs below, we can conclude that the size is a significant effect when performing parallel computing. The big size speed up is the most significant, following by the small size image set. The linear relationship between speedup and number of threads is less significant on mixture size image set. This pattern appeared in both the "parslices" and "parfiles" mode. Moreover, all the experiment cases have speed up less than the ideal speed up, which is a expected result and will be analyzed in the Experiment Analysis section.





4. Experiment Analysis

From the experiment results, the possible reason causing the real speed up less than the ideal speed up is the parallel fraction. In the ideal situation, we assumed that the parallel part of program is full paralleled; however, in our program implementation, there is multiple part of the program is implemented sequentially. For example, when creating tasks for each image and put the task into queue structure, we load, read, and create the task for images sequentially. Additionally, in the "parslices" mode, we ensured that next image is processed on when the current image processing is complete. These are all the bottleneck of the parallel design. We also ensured that each effect is processed sequentially. All these sequentially implemented part in our parallel design is not calculated and included when obtaining the ideal speed up and calculate ideal parallel fraction, causing the real speed up significant less than the ideal speed up. There also exists some bottleneck and hotspot in the sequential part of the system. When reading, loading, and saving the copy of image. All file I/O is implemented sequentially and hard to be paralleled and bounded by the CPU computation power. The incorrect concurrency design of the File I/O could cause serious race condition or catastrophic incident to the system. The ATS lock is another important bottleneck. Thought the lock is simple to implement but it is not so efficient. The thread will always keep spinning and check the lock condition to enter the critical section when the lock is acquired, causing waste of computation power. We also note that size of pictures can affect the performance of parallel design. This is a expected result. The parallel design will effectively distribute the large amount of work when processing the large size image. However, when the thread numbers increase, each thread will still get enough work to process when dealing with the big image set, for the small image set, the same implementation may cause over heading leading to waste of computation power and decrease in performance. When dealing with mixture dataset, the system will have to frequently wait for the big size part of image to process; while the small size part of the image will still suffer from over heading, leading to a significantly decrease in performance. From the experiment result, we can see that the "parslices" performed better since it can create multiple threads to process different slices of a image instead of putting one image in one single thread. This design significantly increased the performance since each thread now responsible for smaller part of the work.

5. Future Implementation

In the future, to further increase the performance of our system, we can firstly use a more efficient lock instead of the ATS lock to avoid waste of computation power, such as MCS lock and ticket lock. Secondly, the parallel part of the program is not fully paralleled. There is still multiple sequential design discussed in

the experiment analysis section. Further concurrency design can increase the parallel fraction and leading to the increase of performance. It is also important to note that further concurrency implementation will greatly decrease the readability and increase the complexity of the code. We should implement this deisgn strictly based on the user case needs. Lastly, to solve the File I/O hotspot problem, we may create a cache system for frequently accessed file. This may not benefit in this project since the image set numbers is small; however this design will become beneficial when the image set number is significant large.