Project Report

Description

This project performs a fixed set of image convolution tasks using three different approahces. The first approach is sequential, as implemented in `scheduler/sequential.go`; the second approach is task-wise parallelization, as implemented in `scheduler/parfiles.go`; the third one is slice-wise parallelization for each image task with a sequential implementation for different tasks, as implemented in `scheduler/parslices.go`. We would like to measure the speedup in execution time of the two parallel implementations using different number of threads, and investigate its relationship to Amdahl's Law.

Performance Measurement and Speedup Graphs

I. Theoretical Speedup

By putting time stamps in `parfiles.go` and `parslices.go` to record the program execution time spent in parallelization and total time, we found that both `parfiles.go` and `parslices.go` have a 95%-100% parallelization portion, we therefore take p = 95% in Amdahl's Law, considering the possible effects of compiler optimization (Amdahl's

Law states that $\text{speedup} = \frac{1}{1-p+\frac{p}{n}}$, where p is the parallel portion of the program, n is the number of threads). The theoretical speedup table is calculated as follows:

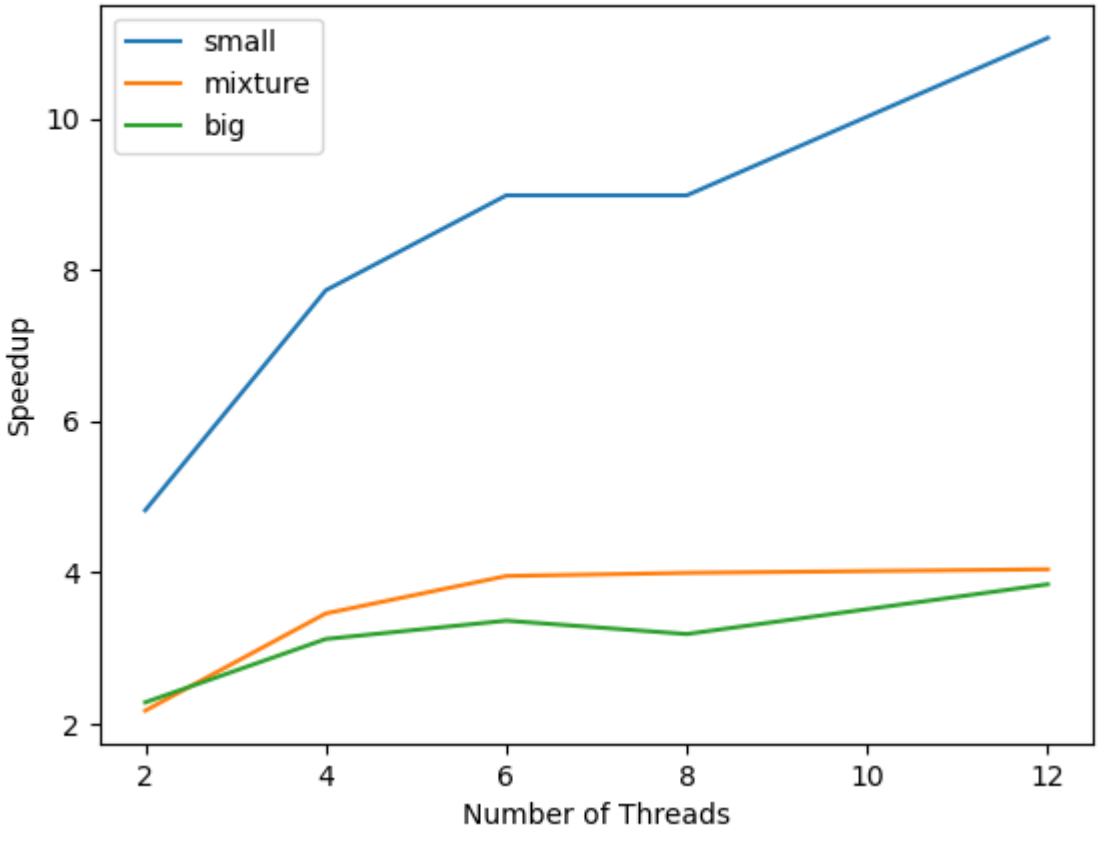
Speedup \ Number of Threads	N=2	N=4	N=6	N=8	N=10	N=12
`parfiles.go` and `parslices.go`	1.90	3.47	4.8	5.92	6.89	7.74

The speedup limit by an infinite number of threads in this case is 20.

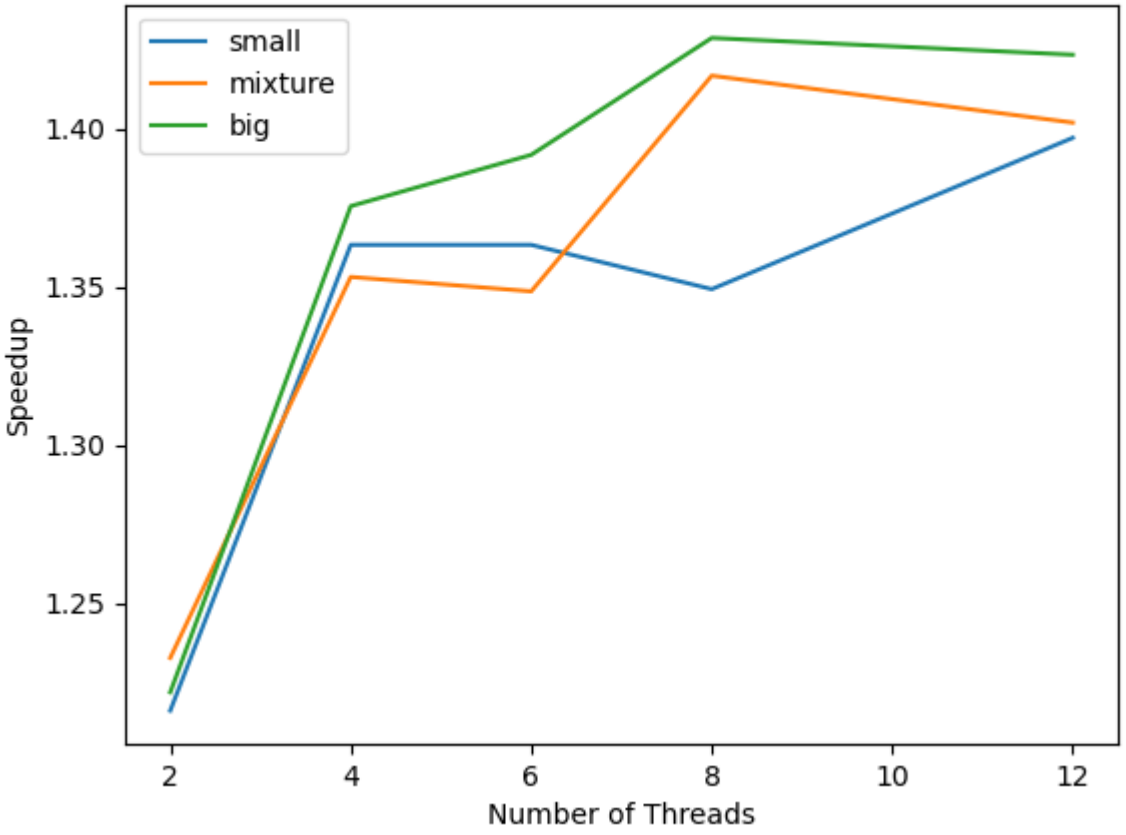
II. Experimental Speedup Graphs

We then calculated the actual speedup by $\text{speedup} = \frac{\text{program execution time by one thread}}{\text{program execution time by n threads}}$, with the same set of number of threads stated in part I, respectively, over `small`, `mixture`, and `big` image size input. The experiment is conducted on both `parfiles.go` and `parslices.go`. The result is plotted as follows:

Speedup Graph for `parfiles.go`



Speedup Graph for `parslices.go`



Performance Analysis

The sequential implementation in `sequential.go` has a huge hotspot on each image task processing, while others need to wait until the previous one is completely done. However, each image task is rather independent from the others, we therefore conclude that a naive sequential approach is not an efficient way to convolute different images. As we then adopted two parallelization approaches whose performance are shown in part II of the previous section, we found that `parfiles.go` achieves a larger speedup than `parslices.go`, with an average speedup of 4x times for a number of threads over 6. We therefore conclude that task-wise parallelization is best suitable for image convolution tasks.

More specifically, in the two parallel versions, we found that input image size affects `parfiles.go` but not `parslices.go`, as in the first graph the line for small image size is far above the two lines for mixture and big image size, while in the second graph three lines are interwoven.

Lastly we compared the experimental results to the theoretical results indicated by Amdahl's Law. We found that for `parfiles.go`, "mixture" and "big" lines are consistent

with Amdahl's Law when the number of threads is less than 7, while the "small" line seems to perform asymptotically consistently with the law, as it might reach the theoretical speedup limit of 20. For `parslices.go`, no line seems to act toward Amdahl's Law; there might be a cache line effect when different threads are reading the overlapping part of the image in order to perform convolution.

The big difference between `parfiles.go` and `parslices.go` might be caused by the cache line effects, as threads in `parfiles.go` are operating on completely different images, while threads in `parslices.go` are reading the same image. Cache line effects extend the efforts for memory consistency checks, and that may be where `parslices.go` has a limited speedup compared to the other.

How to run the automated test

To run the test and reproduce the above graphical results, run `sbatch benchmark-proj1.sh` command on the Linux cluster. The speedup graphs will be outputed to the same directory as `benchmark-proj1.sh`, i.e., in the `benchmark` directory.