The project implements three versions of image editor that apply convolution effects on given images. The sequential version processes images one at a time without parallelism. Each image is fully loaded, effects are applied in-order using sequential convolution operations, and results are saved before moving to the next image. This version serves as the baseline for performance comparisons. The parfiles version spawns goroutines that compete to pull image tasks from a shared task queue protected by a TAS lock. After a goroutine acquires the lock, the image task along with a sequence of effects will then be executed independently. The parslices version processes an individual image by splitting it into slices. This version has each goroutine apply the same effect on their own slices, wait for all slices to be completed between effects, and move on to the next effect instruction together. The parfiles version attempts to maximize hardware utilization when processing many images, while the parslices version tries to accelerate single large image processing.

Instruction on generating performance testing plots, run: `sbatch benchmark-proj1.sh` at `…/proj1/benchmark`. The test runs each combination of parallel version, number of threads, data_dir of image five times, and outputs the results into txt files at `benchmark/results`.

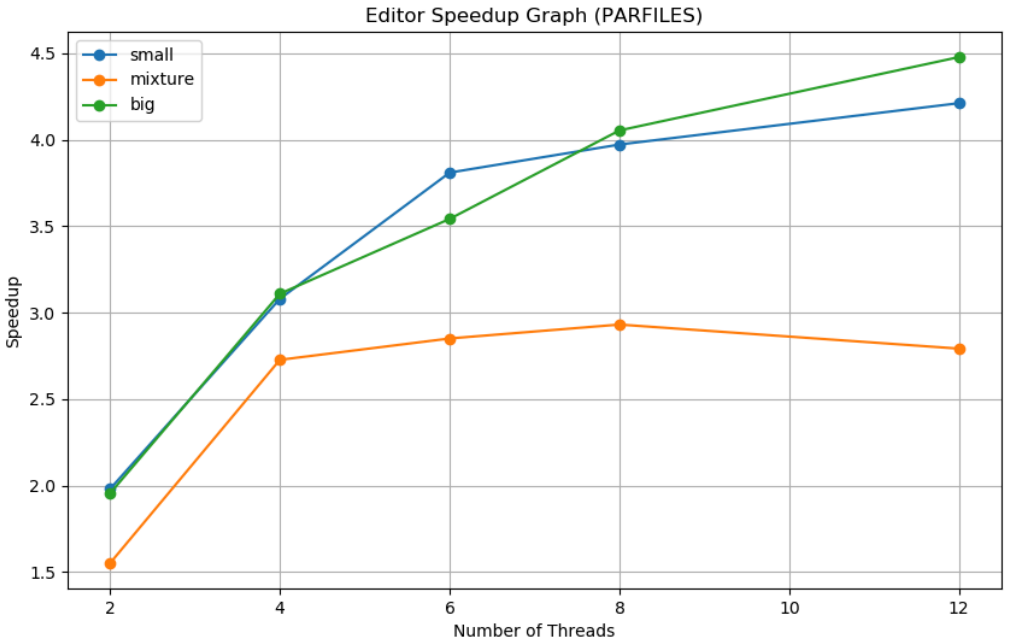
Parallel_fraction	parfiles (n=1)	parslices (n=1)
small	0.99 (4.64 / 4.65)	0.31 (1.46 / 4.67)
mixture	0.99 (28.17 / 28.47)	0.32 (8.91 / 28.57)
big	0.99 (71.76 / 71.77)	0.32 (23.41 / 72.33)

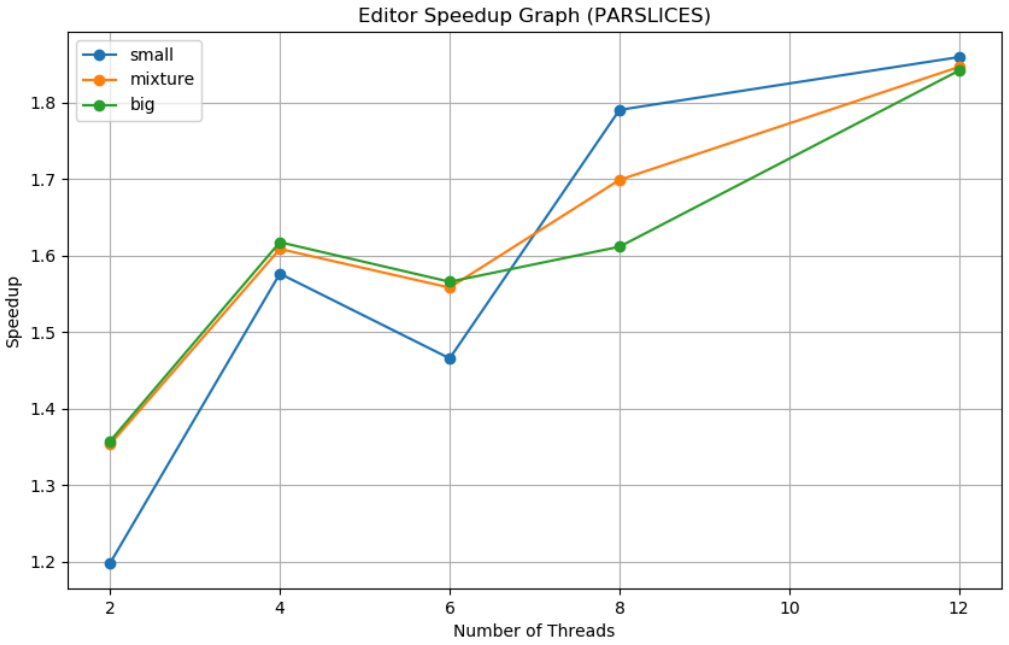
Amdahl's Speed_up	n=2	n=4	n=6	n=8	n=12
parfiles (p=0.99)	1.98	3.88	5.71	7.47	10.81
parslices (p=0.32)	1.19	1.32	1.36	1.38	1.42

Observation

1. **For sequential version:**
   The main hotspot in the sequential program is the convolution operation, which requires multiple nested loops and kernel calculations for each pixel. File I/O operations (reading/writing PNG files) create sequential bottlenecks since loading and writing large image files create latency.

2. **Comparison between two parallel versions:**
   The parfiles version is faster than the parslices version because unlike the later version, the former one has less synchronization overhead since each image is processed independently. Also, the later one creates sequential bottlenecks while loading and saving large images.





3. **Image size impact:**
    a. Parfiles: Mixture dataset performs worse than both, I suspect there exists overhead from handling varying image sizes?
    b. Parslices: Image size doesn't matter much eventually when we use 12 workers. The program reaches a ~1.8x speedup with 12 threads for all types of dataset.

4. **Amdahl's Law Analysis:**
    a. For the parfiles version, the theoretical speed-up should be near-linear. This

is due to the fact that each image is processed independently by a single thread/goroutine. Each goroutine handles its own file I/O individually and doesn't have to wait between image effects. However, the discrepancies from the actual speed-up data could be derived from contention from shared system resource contention (`SBATCH --nodes=1`). In this sense, memory bandwidth also becomes a bottleneck since all threads share the same memory bus, thus affecting performance when multiple threads access different parts of memory. File I/O is also a bottleneck when multiple threads try to read/write images simultaneously.

b. For the parslices version, the actual speed-ups align closer to theoretical values.

5. **Improvement:**
   For parslices, instead of processing effects sequentially (image → slices → E1 → image → slices → E2…), we can create a pipeline in which multiple slices of the image move through the pipeline concurrently so that the program doesn't have to wait between effects (multiples slices → E1 → E2 → E3 → E4 → image). And I think this can be done by the second option provided in the project instruction part3.