Parallel Image Convolution

In this assignment, you will parallelize a standard image convolution function to run in a shared-memory parallel environment.

1 Read

Read the *Image Convolution* article on the course web site.

2 Set Up

Follow these steps to get set up.

- This assignment uses the Lode PNG library to load and store PNG images. You can use the original source of `lodepng` if you'd like. In addition, I've prepared a minor fork of Lode~PNG that renames the main file to work as C~source (instead of C++). You are welcome clone this version from my TU Github page.
- Download the sequential implementation of convolution that is available on the course web site.
- Compile the sequential version and link it with the `lodepng.o` file. Because this is not (yet) threaded code, you shouldn't have to link against any additional libraries.
- Try out the sequential version on a PNG image. Run with no arguments or the `-h` flag to get a usage message. Use two different kernels to verify the behavior. The first kernel you try should probably be the `identity` kernel, which just makes a copy of the original image.

3 Code

1. (35 points) Rewrite the convolution program to run with multiple threads.

   Following the PCAM design approach, your first job will be to identify a strategy for *p*artitioning the image into smaller pieces that can be convolved simultaneously by each thread. Likely candidates would be a 1-D partitioning either horizontally or vertically, or a 2-D partitioning into blocks. Take cache behavior into consideration when making your decision.

   Consistent with the sequential implementation I've supplied, I am not requiring that you handle the pixels at the very edges of the image (i.e., the single row/column of pixels on each of the four sides). (But see the extra credit points below.)

2. Find three (*tasteful*) PNG images sized as follows:

   "small" The small image should be small enough that a single partition of the image can fit entirely in the cache of a single core of your CPU. On modern Intel CPUs, this would mean either the L1 data cache or the L2 shared cache, both of which are dedicated to a single core.

"large" An image too large for a single partition to fit into core-local cache (i.e., the opposite of the "small" case). If possible, find an image that won't even fit into L3 cache in its entirety.

"medium" Something in between "small" and "large"

(a) (5 points) For each of your images, list the image dimension in pixels and the total image size in bytes.

(b) (5 points) Based on your earlier analysis report the *core* and *cache* configuration of your lab machine, including the cache configuration (i.e., how many levels of cache; how cache is allocated to cores) and the sizes of each cache.

(c) (5 points) Explain how the images you have chosen and your CPU architecture satisfy the requirements for the images discussed above.

For each image, run convolution experiments with varying numbers of threads. If your CPU has c cores, include runs for at least $p = 1, 2, \ldots, c$ and $2c$ threads. Measure the time t_p for each run. Be sure that t_p includes *all* overhead due to parallelization (e.g., thread creation, thread joins).

(a) (10 points) Report your performance results as illustrated in Table 1.

Image	p	t_p (s)	s	e
`kitten.png`	1	10.0	1.0	1.0
	2	5.0	2.0	1.0
	⋮			
`firefly.png`	1	8.0	1.0	1.0
	⋮			

Table 1: Format for performance report

(b) (10 points) Following the pattern shown in class, create three plots that show t_p, s, and e for your experiments. The horizontal axis of all three plots should reflect the number of threads in the range $1 \ldots 2c$. Each of the three plots should contain three curves, one for each image.

3. Discuss the overall performance results for t_p, s, and e as reported for question 2.

(a) (5 points) How did performance results vary with number of threads and image size?

(b) (5 points) What, if anything, surprised you about the results you obtained? Why?

(c) (5 points) Describe your partitioning scheme. Seeing your performance results, would you choose the same scheme again? Why or why not?

(d) (5 points) Based on thread counts and image sizes, describe how your experimental data reflects the caching behavior on your CPU.

4. (10 points (bonus)) Add code to handle the edges of the image. For pixels that are "outside" the image, you may choose to use a fixed color or replicate the pixels at the edge of the image.

Run another set of experiments with the same p values on your *smallest* image. Plot t, s, and e. Discuss any variation you see with the additional computation in this version. Is performance better? Worse? Neither? Why?

Question	Points	Bonus Points
1	35	0
2	35	0
3	20	0
4	0	10
Total:	90	10