CS 420, Parallel Programming, Fall 2019
MP3

Szaday, Justin Shah, Ishan Pankaj

Due Date: October 23rd, 2019, at Midnight

1 Overview

The purpose of this assignment is straightforward – for you to gain experience using `pthreads` and assess the performance of a parallel program. To achieve this, you will parallelize an implementation of the Mandelbrot Sequence and benchmark it. Students taking this course for 4 credit-hours will additionally use atomics to parallelize Histogram Sort.

2 Getting Started

2.1 Skeleton Programs

Pull the skeleton for MP3 from the `_release` repository: https://github-dev.cs.illinois.edu/cs420-fa19/_release

2.2 References

We encourage you to start with the following links if you would like any additional information about the topics discussed in this MP:

- https://en.wikipedia.org/wiki/Counting_sort
- https://en.wikipedia.org/wiki/Embarrassingly_parallel
- https://www.geeksforgeeks.org/fractals-in-cc/
- http://books.gigatux.nl/mirror/kerneldevelopment/0672327201/ch09lev1sec1.html
- https://en.cppreference.com/w/c/atomic

3 Part A, Parallelizing the Mandelbrot Set

3.1 The Mandelbrot Set

The Mandelbrot Set is a set of complex numbers whose computation represents an embarrassingly parallel problem; this means that no synchronization or communication among threads is required. We have provided a skeleton program, `mandelbrot`, that contains a serial implementation of the Mandelbrot Set which produces a PNG image, an example of which is shown in Figure 1. The parameters for the program are as follows:

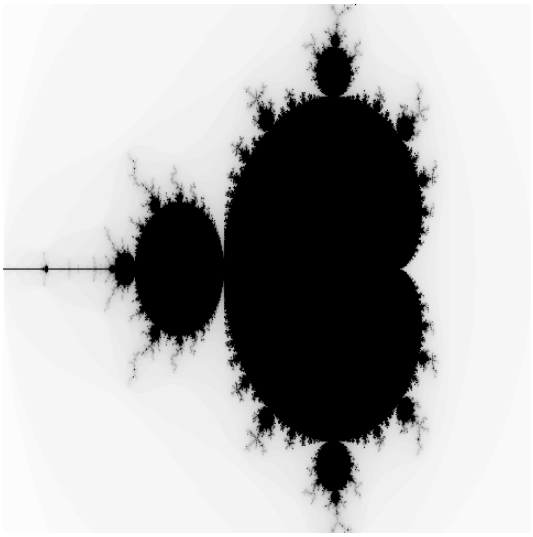

Figure 1: An example image of the Mandelbrot Set.

- `-o <file>` – Specify the output file (by default: `out.png`).
- `-w <width>` – Specify the width of the generated image in pixels (by default 480).
- `-h <height>` – Specify the height of the generated image in pixels (by default 480).
- `-n <N>` – Run the program with N threads (your task is to implement this).

3.2 Your Task

In the section marked in the skeleton, create `kData.numThreads` threads and wait for their completion; hint, you are expected to use `pthread_create` and `pthread_join`. These threads should run `generateMandelbrot` with `startX` and `endX` representing a unique, non-overlapping sub-range of `0` to `kData.width`. You are free to divide up this workload anyway you choose; however, it is worth noting that the number of threads may not evenly divide with the image width (i.e. `kData.width`).

Once you have implemented your parallel version and verified its correctness (by verifying the generated image matches the serial version), you are to run the small benchmarking program we have included. To do this, load Python as follows:

```
module load python/3 && unset PYTHONPATH
```

Then run the benchmark with:

```
python3 benchmark.py <ImgSize>
```

This will run the program with a number of threads, and produce results along the lines of:

```
Generating a XxX mandelbrot with thread counts: [1, 2, 4, ..., 20]
The serial version ran for X s.
```

```
The parallel version, with 1 thread(s), ran for X s, a speedup of Xx.
The parallel version, with 2 thread(s), ran for X s, a speedup of Xx.
The parallel version, with 4 thread(s), ran for X s, a speedup of Xx.
...
The parallel version, with 20 thread(s), ran for X s, a speedup of Xx.
```

NOTE: For accurate results, you should not run the benchmark program on the login nodes, create a job script and run it using the same methodology from MP1 and MP2.

Try running the benchmark program with the following data sizes: 256, 512, 1024, 2048, 4096. Briefly comment about how changing the data size affects the performance of the parallel version with four threads, and include the output of the benchmarking program for data size 2048 in your report.

For data size 2048... Plot *Speedup vs. Number of Threads* using the results of the benchmark program.

For data size 2048... How did the performance of the parallel version with a single thread compare to the serial version? Briefly explain your results.

For data size 2048... How did the performance of the parallel version with multiple threads compare to serial version? Briefly explain your results.

Using the campus cluster documentation to find the number of cores for the compute nodes, what happened when the number of threads exceeded the number of available cores? What would you expect to happen?

4 Part B, Histogram Sort (4 Credits Only)

4.1 Your Task

To gain additional experience with pthreads and learn about atomic operations, you will parallelize histogram sort. Atomic operations are operations in which the value cannot be modified between the instant its value is read (to be returned) and the moment it is modified. This alleviates the need for a critical section, since data races cannot arise when using atomic operations. Like for Part A, a serial version has been provided for use as the basis of your parallel version. To parallelize it, you will have to:

- Spawn `numThreads` threads and divvy up the workload.
- Change the type of the `counts` array to `atomic_int`.
- Replace the decrement operation on `counts` in `populate_sorted` with a fetch-and-subtract operation.

- Replace the increment operation on `counts` in `compute_counts` with a fetch-and-add operation.
- Use the atomic operations from the C11 `stdatomics` library.

We will be evaluating this part of the assignment based on the correctness of your parallel implementation.

4.2 Histogram Sort Program

The program sorts a randomly generated string of uppercase letters ranging from 'A' to 'Z'. The parameters for the program are as follows:

- `-s <seed>` – Specify the seed used to generate random values.
- `-l <length>` – Specify the length of the string to be sorted.
- `-v` – Specify that the program should print the sorted and unsorted strings (verbose output).
- `-n <N>` – Run the program with N threads (your task is to implement this).

5 Submission Guidelines

5.1 Expectations For This Assignment

The graded portions of this assignment are your parallelized Mandelbrot program and answers to the short-answer questions. Four credit hour students are additionally expected to complete Part B.

5.1.1 Points Breakdown

- $5 * 10 pts.$ — Each Short Answer Question
- $50 pts.$ — Parallelized Mandelbrot Program
- $50 pts.$ — Parallelized Histogram Sort (4 cr. only)

5.2 Pushing Your Submission

Follow the git commands to commit and push your changes to the remote repo. Ensure that your files are visible on the GitHub web interface, your work will not be graded otherwise. Only the most recent commit before the deadline will be graded.