Checkpoint Report

Nanxi Li nanxil1, Catherine Yu tianhony

Schedule (updated)

November 2
 * Discuss Idea with Instructors

November 4
 * Project Proposal Due Wednesday

November 5-11 (7 days)
 * Research and find useful libraries to use in C and C++

November 12-29 (18 days)
 * Implement the sequential solution

November 30
 * Checkpoint Report

December 1-3 (3 days)
 * Implement a baseline CUDA version

December 4-10 (7 days)
 * Add optimizations to the CUDA version
 * Parallel processing the dots on the lines when drawing a single line
 * Parallel processing the lines and compute L2 norms when adding/deleting lines

December 11-14 (4 days)
 * Benchmark CUDA and Write report

December 14:
 * Final Report Due

December 15 8.30-11.30am:
 * Virtual Poster Session

Work Completed until Checkpoint

We have researched different implementations for thread art online, and we were aiming to find an implementation that's suitable for coding in CUDA and parallelizable. The two references we found when proposing the project both had theoretically parallelizable algorithm, but it was hard to directly translate them to the baseline code we wanted. We looked into other implementations online as well, and we reached the conclusion that we need to write our baseline code from scratch.

We implemented the baseline code in two parts: (1) image pre-processing according to the algorithm proposed in paper by Birsak et al; (2) greedy computation to find the most accurate string representation of the image by adding and subtracting threads. We used the stb_image library in image pre-processing, since it is robust and easy to use. We implemented the greedy computation by storing all pixels and lines in int[], and only used arithmetic operations for all computation in terms of drawing/removing lines, analyzing images and generating L2-norms. The approach was difficult to implement in terms of correctness, because there are many non-trivial steps that are intertwined with each other. We chose this approach regardless because this approach is easy to optimize as CUDA code, which was what we proposed.

We now finished outlining CUDA code for part (2) described above. We want to find the pair of pins that maximizes reduction of the l2 norm, there are n(n-1) problems always, we use cuda to have each thread compute the l2 norm for each problem, and then reduce them to find the one with the largest l2 norm. We will also parallelize pixel processing of the line generation.

Goals past Checkpoint

Although we have spent more time on the baseline code than what we proposed, now we have a steady grasp of the algorithm and our implementation that we believe the optimization will take less time then what we expected. We expect to deliver the same as stated in PLAN TO ACHIEVE. We would not be able to achieve the NICE TO HAVES since we are tight on scheduling. We want to implement the edges solution(instead of a continuous thread) and optimize with CUDA.

PLAN TO ACHIEVE

- We plan to implement the edges solution(instead of a continuous thread) sequentially, in parallel using CUDA.
- We will benchmark their performances on GHC and lateday machines and analyze them.
- For the demo, we want to be able to take input images, and output the string art version of it using our parallel algorithms, and show precomputed speedup graphs.

Poster Session Plan

We will not be able to do a live demo since thread art projects usually take hours to synthesize. (14 hours as mentioned in the paper.) We want to show the images that we synthesized beforehand, and we also want to show our speedup analysis graphs during poster session.

Concerns

The main concern is that GHC machines kill the jobs that timed out. It would be difficult to benchmark our baseline solution on a high-resolution picture because it would take too long. We will try to benchmark the solution on low-resolution pictures, which will not have the resemblance we want, while running high-resolution pictures at the same time to ensure correctness.