Parallel Image Inpainting

15-618 Final Project

Saileshwar Karthik (saileshk)
Ayushi Bansal (ayushib)

Summary of work done

Firstly, we implemented a working C++ based version of the PatchMatch image inpainting algorithm (The code was adapted from this open-source repo). It uses the OpenCV library to do the image parsing/loading/storing functions. We were also successful in getting this to work both on our local machines (Silicon Macs) and the GHC machines. This was Checkpoint 2 in the tentative schedule we put up in our proposal.

We initially planned on resorting to OpenMP only parallelism, but then we decided to explore GPU based approaches too by splitting work amongst ourselves. The propagation stage within the iterative refinement stage involved dependencies where computing the value of a certain pixel required the computed values of neighbouring pixels. We were able to parallelize this using red-black ordering and got good speedups with both CPU and GPU based approaches:

Progress update

We have yet to start work on the image submission/masking feature (this was initially in our checkpoint 3 todo list). This was mainly because we shifted our attention towards GPU parallelism which was more interesting and was also briefly touched upon within our reference paper (PatchMatch)

Apart from this task, we are on track as per the rest of the schedule we put up on our project proposal.

Progress on our “nice to haves”:

1. Allow users to paint over any image of their choice to apply the masks - We plan on getting this done
2. Implement a parallelised version using CUDA - Already working on this

We plan to hit “nice to have” task #1 for the poster session.

Poster Session Deliverables

We plan on showing both a live demo + set of graphs depicting the speedups against the different approaches we parallelised.

Preliminary Results

Input:

- Image resolution: 640x360
- Mask size: 43287 pixels

original image

image with mask applied

single threaded inpainted image

Multithreaded (8 threads) inpainted image

CPU Results:

Runtime of propagation phase using 8 threads.

Performance:

- CPU Processing time (1 thread): 8.20s
- CPU Processing time (8 threads): 4.65s
- Speedup observed overall: 1.76x
- Speedup observed (NNF pyramid bottom): 7.7x

Quality (compared to sequential code):

- Mean pixel diff: 2.09716
- Max pixel diff: 109.000000
- Stddev pixel diff: 6.63323

GPU Results:

Run using 256 threads.

Performance:
- CPU Processing time (1 thread): 12.466884s
- GPU Processing time: 4.735513s
- Speedup observed: 2.63x

Quality (compared to sequential code):
- Mean pixel diff: 1.152865
- Max pixel diff: 79.000000
- Stddev pixel diff: 3.014311

Note: Only the propagation stage runs within a kernel. The random search + voting phases run sequentially (yet to be parallelised)

As observed from the stats presented, when considering CPU parallelism during the NNF (propagation) phase, we can achieve close to linear speedup at the finest resolution. We also see deviations in per-pixel values theoretically, even as the output image looks visually coherent. We plan on exploring the effects of these deviations on some more test cases and do an analysis of the visual and theoretical quality tradeoffs, and potentially explore other parallelism strategies.

Issues/Concerns

We plan on emailing/meeting with the course staff regarding the issues that concern us.