HPC - 1. Assignment - Parallel Seam Carving using OpenMP

Kristjan Kostanjšek, Nejc Ločičnik

I. Introduction

TODO

II. Optimizations and Results

The subsections below go through the optimizations performed on the sequential code, reporting on computational speed-up for each optimization. The computational speed-up is calculated according to the following formula: $S = t_S/t_P$, where t_S is the time of the sequential program, while t_P is the time of the parallel program. Each time or speed-up reported is an average of 5 runs. As the following tables only include the speedup S, the t_S for each image size is shown in table II for reference.

I Size	**720x480**	**1024x768**	**1920x1200**	3840x2160	7680x4320
Time [s]	9.63	21.58	64.49	229.01	932.31

Table I
Sequential times for reference.

Just to note, but the compiler seems to be doing some parallelization on it's own, I had to force it to run on a single thread to get the times above. If I had more threads, the times get substantially reduced, for example, image of size `720x480` needs around 2-3 seconds from 9-10, even though there is no manual parallelization implemented.

A. Parallel Energy Calculation

The parallel energy calculation is the most straightforward; we simply collapse the x, y for loops and introduce parallelization through OpenMP. This part was additionally improved by modifying the energy calculation by removing the final squared function and normalization division as they are monotonic and will not change the result when we are searching for minimums in the following parts of the code.

Computational speed-ups are shown in table II-A.

T\I	**720x480**	**1024x768**	**1920x1200**	3840x2160	7680x4320
4	4.88	6.22	7.95	7.13	7.36
8	8.87	9.20	12.49	11.32	10.34
16	10.92	10.77	12.39	15.09	14.31
32	12.35	11.87	14.41	18.77	23.90

Table II
Parallel energy calculation speedups S.

comments on the results

B. Parallel Cumulative Energy Calculation

- briefly explain triangles: horizontal stripes, down triangles in parallel, fill stripe with up triangles in parallel - lots of overhead so it performs worse on smaller images, improves slightly on larger ones - results are mediocre because we are missing cache hacking

Parallel cumulative energy calculation has room for further optimization by remapping the energy map triangles into sequential linear arrays, doing the cumulative energy calculation and remapping back. By doing this we can reduce the amount of cache misses, which this triangle structure is very susceptible to. We attempted doing this, but quickly gave up as the remapping is too headache inducing.

Computational speed-ups are shown in table II-B (NOTE: This is on top of previous optimizations).

Result comments...

T\I	**720x480**	**1024x768**	**1920x1200**	3840x2160	7680x4320
4	4.82	6.25	7.63	8.14	8.78
8	8.72	9.13	12.43	13.84	15.05
16	10.08	10.25	13.35	19.78	14.86
32	9.83	10.75	13.56	22.85	25.35

Table III
Parallel cumulative energy calculation speedups S.

C. Parallel Seam Removal

- find minimal path in each vertical stripe (directly depending on number of threads) in parallel - remove the seam in each vertical stripe in parallel

Computational speed-ups are shown in table II-C (NOTE: This is on top of previous optimizations).

T\I	**720x480**	**1024x768**	**1920x1200**	3840x2160	7680x4320
4	17.72	17.42	22.38	28.20	32.76
8	58.86	60.14	62.53	67.04	90.43
16	27.32	21.99	20.34	28.12	30.06
32	49.82	42.90	47.10	68.84	83.25
64	32.51	39.43	59.80	50.93	82.75
128	35.67	50.84	84.19	86.68	147.88

Table IV
Parallel greedy seam removal speedups S.

Result comments...

Reduction of quality from greedy approach example ... (currently waiting for 128 cores to free up on arnes)

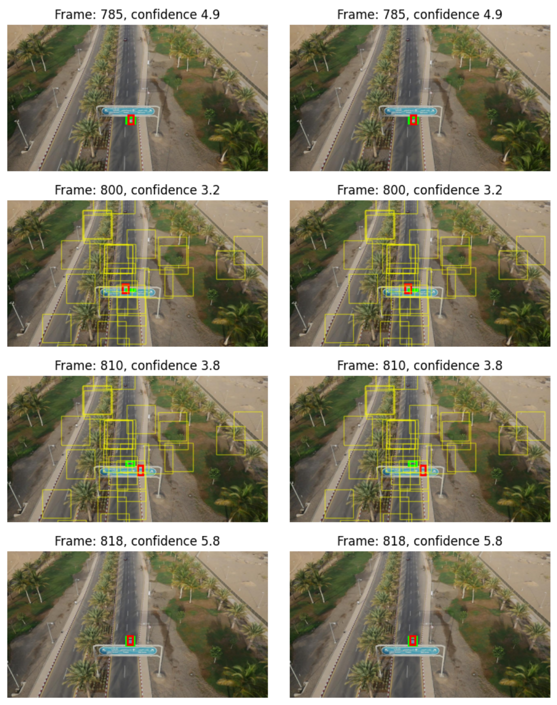
Frame: 785, confidence 4.9
Frame: 785, confidence 4.9
Frame: 800, confidence 3.2
Frame: 800, confidence 3.2
Frame: 810, confidence 3.8
Frame: 810, confidence 3.8
Frame: 818, confidence 5.8
Frame: 818, confidence 5.8

Figure 1. PLACEHOLDER

III. Conclusion

TODO