HPC - 1. Assignment - Parallel Seam Carving using OpenMP

Kristjan Kostanjšek, Nejc Ločičnik

I. Introduction

Seam carving is a content-aware image resizing technique that preserves important visual content by iteratively removing low-energy paths, or seams, from an image [1]. While effective, the algorithm's computational complexity grows significantly with image size, particularly due to energy calculations, seam identification, and seam removal. Parallelization offers a promising approach to improving efficiency.

This report focuses on optimizing the sequential seam carving algorithm by parallelizing its key components—energy calculation, cumulative energy calculation, and seam removal—using OpenMP. Significant speed-ups were achieved, reaching up to 147.88x for the largest image with 128 threads. Notably, the greedy seam removal approach maintained consistent image quality even when removing half the pixels, demonstrating its robustness. The results indicate that further scalability is achievable with additional threads. The following sections detail the optimizations and their performance impacts.

II. Optimizations and Results

The following subsections detail the optimizations applied to the sequential code, along with the computational speed-up achieved for each optimization. The speed-up is calculated using the formula: $S = t_S/t_P$, where t_S represents the execution time of the sequential program, and t_P denotes the execution time of the parallel program. All reported times and speed-ups are averages of five runs. While the subsequent tables focus on the speed-up S, the sequential execution times t_S for each image size are provided in Table I 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.

It is worth noting that the compiler appears to perform some automatic parallelization, which required us to enforce single-threaded execution to obtain the sequential times reported above. When additional threads are utilized, the execution times are significantly reduced. For instance, an image of size `720x480` completes in approximately 2–3 seconds, compared to the 9–10 seconds observed in single-threaded execution, despite no manual parallelization being implemented. Additionally, the times reported above do not include parallelized seam removal (e.g., image copying).

A. Parallel Energy Calculation

The parallelization of the energy calculation is the most straightforward step. We achieve this by collapsing the two nested loops iterating over the image coordinates and introducing parallelization via OpenMP. This implementation was further optimized by removing the final squared function and the normalization division. Since these operations are monotonic, they do not affect the results when searching for minimum energy values in subsequent stages of the algorithm.

The computational speed-ups achieved through this optimization are presented in Table II.

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.

By parallelizing the energy calculation, we observe a significant improvement in the algorithm's speed. This performance gain is particularly pronounced for larger images, where parallelization has a more substantial impact. As anticipated, increasing the number of threads further enhances the algorithm's efficiency, with the speed-up scaling proportionally as additional computational resources are utilized.

B. Parallel Cumulative Energy Calculation

There are two primary approaches to parallelizing the cumulative energy calculation. The first and simpler method involves computing the cumulative energy row by row, where each row is divided among the threads. While straightforward, this approach yields only modest performance improvements. The second, more complex method, involves dividing the image into horizontal stripes and further decomposing each stripe into down-facing and up-facing (fill) triangles. This allows each thread to process multiple rows independently, as it has access to all dependent pixels. However, despite its theoretical advantages, this method introduces significant overhead and potential cache misses due to poor locality of the triangle elements, resulting in mediocre performance gains.

To mitigate this, the energy map triangles can be remapped into sequential linear arrays, enabling the cumulative energy calculation to be performed with fewer cache misses. After the calculation, the results can be remapped back to their original structure. While this optimization holds promise, we abandoned its implementation due to the complexity and challenges associated with the remapping process.

The computational speed-ups achieved through these optimizations (applied on top of previous improvements) are presented in Table III.

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.

As anticipated, the overhead associated with parallelization leads to slightly worse performance for smaller images, with modest improvements observed for larger images. We tested both the row-based and triangle-based methods, and the triangle method, even without cache optimization, yielded marginally better results. An additional parameter in the triangle method is the triangle size. We found that as long as the triangle width is between 16 and $I_W/(N_T * 2)$, the performance remains relatively consistent. If cache optimization were implemented, the triangle size could likely be fine-tuned to align with the cache size for further improvements.

C. Parallel Seam Removal

Parallel seam removal was implemented by dividing the image into vertical strips, with the number of strips corresponding to the number of available threads. The minimal path for each strip was identified in parallel, and the seams were subsequently removed from each strip, also in parallel. While this greedy approach significantly accelerates the seam removal process, it was anticipated to result in a noticeable degradation in output image quality.

Surprisingly, the image quality remained better than expected. As illustrated in Figure 1, even after removing half of the image (512 and 960 pixels), the quality remained relatively consistent. These examples were processed using 128 threads, as we initially hypothesized that thinner strips (and thus a more greedy algorithm with fewer paths to choose from) would lead to greater quality degradation. However, the results demonstrate that the approach remains robust even under these conditions.

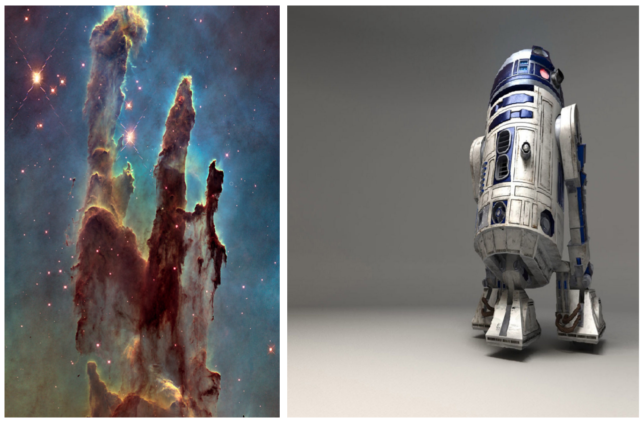
Figure 1. Image quality examples after removing half of the image with greedy seam removal.

Computational speed-ups are shown in table IV (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.

The speed-ups achieved here are substantial, with the most significant factor likely being the parallelization of image copying during seam removal. This part of the algorithm benefits greatly from an increase in the number of threads, as evidenced by the notable performance jump from 64 to 128 threads. This suggests that 128 threads is not the upper limit for performance gains, and further improvements might be possible with additional threads.

For reference, Table V provides the fastest computation times achieved for each image size, incorporating all three optimizations and utilizing 128 threads.

I Size	**720x480**	**1024x768**	**1920x1200**	3840x2160	7680x4320
Time [s]	0.0247	0.0394	0.0618	0.1318	0.4403

Table V
Fastest computation times for each image size.

III. Conclusion

In this report, we optimized the seam carving algorithm by parallelizing its key components—energy calculation, cumulative energy calculation, and seam removal—using OpenMP. Significant speed-ups were achieved, with parallel seam removal reaching up to 147.88x for the largest image using 128 threads. While parallel cumulative energy calculation provided only modest improvements due to overhead, the greedy seam removal approach maintained consistent image quality even when removing half the pixels, demonstrating its robustness.

Further improvements could be explored, particularly in optimizing cache usage for the cumulative energy calculation. Techniques such as remapping energy map triangles into sequential linear arrays could reduce cache misses and enhance performance. Additionally, fine-tuning thread counts and triangle sizes for specific hardware configurations may yield further gains. These optimizations highlight the potential for even greater scalability and efficiency in future implementations.

References

[1] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 609–617. 2023.