Parallelization of Content aware Image Resizing

RM Prakash Ramanathan[1], R Manimegalai[2*], D Sivaganesan[3]
, and Sk Noor Mahammad[4]

[1] PSG Institute of Technology and Applied Research, Coimbatore TN 641062, India,
d20z123@psgitech.ac.in
[2] PSG Institute of Technology and Applied Research, Coimbatore TN 641062, India,
drrm@psgitech.ac.in
[3] PSG Institute of Technology and Applied Research, Coimbatore TN 641062, India,
sivaganesan@psgitech.ac.in
[4] IIDM Kancheepuram,
noor@iiitdm.ac.in

Abstract. Image resizing is a common technique used in image editing and processing to reduce or increase the image's dimensions. The naive approaches, such as cropping or scaling, may result in image distortion or data loss. A more sophisticated method is content-aware image resizing, which employs algorithms such as Seam Carving. A path of connected pixels is called as seam. In Seam Carving the least important seam gets removed to increase or decrease the size of the image. The algorithm takes into account the image's gradient magnitude, which represents the changes in intensity between adjacent pixels. This is used to calculate the importance of each pixel. But the issue with seam carving is the execution time delay. In this work, parallelization techniques such as CUDA, MPI and OpenMP are applied to improve the performance and speed. No significant speedup is achieved from MPI and openMP, 2x speedup is achieved from CUDA.

Keywords: CUDA, OpenMP, MPI, Parallelization, Seam Carving, Image resizing

1 Introduction

In the seam carving algorithm, the word seam means a path of connected pixels, that range from top to bottom or from left to right. The seam will be identified using an energy function, image can be resized by removing seams in a particular direction in case of size reduction, or add the seams in one direction in case of size increase.

1.1 Energy Function and Dynamic Seam Table

Energy function is basically the pixel importance. To determine pixel importance, the average absolute difference between color values of six pixels that are directly adjacent to it are used. The energy function is computed in Equation (1). In Equation (1) E represents the energy value of the pixel, i,j indicate row and column number of the pixel [1]. The energy could also be calculated using

C. Kiran Mai et al. (eds.), *Proceedings of the Fourth International Conference on Advances in Computer Engineering and Communication Systems (ICACECS 2023)*, Atlantis Highlights in Computer Sciences 18,
https://doi.org/10.2991/978-94-6463-314-6_29

general edge detection kernels like Sobel edge detection [2], Prewitt Operator etc.

$$E(i,j) = |d/di(i,j)| + |d/dj(i,j)| \tag{1}$$

In the process of active image resizing, it becomes crucial to identify the most optimal seam. To identify the seam, seam carving utilizes a seam map. To determine the seam map, a dynamic programming approach is employed. A seam table is used, where the first row represents the exact values of the energy function, and for subsequent rows Equation (2) can be used.

In Equation (2), M is the seam table and E signifies the energy function, i and j represent rows and columns, the last row will have the cumulative value of the seam table. The seam table is used to select the seam that has the lowest energy. This method is very computationally intensive, a parallel approach would give better results.

$$M(i,j) = E(i,j) + min[M(i-1,j-1), M(i,j-1), M(i+1,j-1)] \tag{2}$$

Parallel programming is the process of running the program parallelly by using multi core, multi thread or GPU systems. In different parts of the paper, three different parallelization techniques are explored to parallelize the sections of code that are computation intensive. The first step is to profile the code. Next the concurrency in the computationally intensive parts are identified. The computationally intensive parts are then parallelized using OpenMP, MPI and CUDA [3].

2 Literature Review

The traditional method to resize an image such as cropping or scaling have drawbacks, with cropping valuable information could get lost, and with scaling the image undergoes distortion, but Shai Avidan and Ariel Shamir came with a novel technique called Seam carving [1], where for resizing, seams will be removed or added. The least important seam in the image can be figured out by using an energy calculation function that utilizes visual saliency and edge detection. But he main disadvantage of seam carving algorithm is its computational intensive and it requires a lot of execution time. But luckily most of the commercially available PCs nowadays use a processor that has multiple cores, or at least 2 cores [4]. So MPI could be used to increase the execution speed of the algorithm. But MPI could cause communication bottlenecks that could slow down the program.

Owing to the popularity of GPUs, GPGPUs can be used to parallelize the energy function of the Seam carving algorithm using CUDA [5].The 1D array of the image can be mapped into a 2D array in CUDA, the input thread is split into tiles consisting of consecutive rows. In the work presented by Duarte et al. [5], the authors explores three distinct approaches using CUDA. The attained speedup of energy calculation function was approximately 75x. However, the seam map

only exhibited a small speedup of 4x, because to its inherent dynamic nature. Though the above technique can efficiently parallelize the energy map, it leaves out the dynamic seam function. Since dynamic seam is also computationally intensive, it is preferable to parallelize it. To parallelize it a Non-Cumulative Seam Carving(NCSC) algorithm which omits dynamic programming can be used. The algorithm in [6] has reduced the communication overheads between GPUs, reduced memory transfer time and allocation time. But NCSC does have some of the same drawbacks of the original seam carving algorithm, geometrical shapes could get distorted.

Seam carving by itself is very effective for content aware image resizing, but it can sometimes lead to data loss. A combination of scaling along with seam carving algorithm can be applied to greatly improve the quality of the output. In this approach proposed by Dong et al, after the removal of each seam, the image gets scaled and the distance will be measured from the original distance [7]. The main drawback with this technique is its inability to preserve the ratio that is present in the original image. In [8], a comprehensive analysis of all the advancements that have been made related to this novel image resizing technique can be found. The studies related to seam carving could be classified into visual output improvement, computation time improvement and deductions on Seam Carving based Resizing. From analysis it is clear that all the proposed methods have different drawbacks, and most of those drawbacks can be avoided by using more efficient saliency techniques. A combination of all the above mentioned techniques can be used to improve the algorithms performance.

3 Parallelization

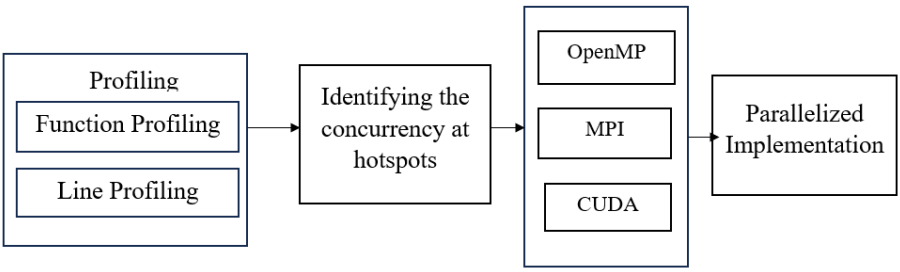


Fig. 1. Steps Involved in Parallelization of a Sequential Algorithm

Fig 1 illustrates the steps that are performed in order to parallelize an algorithm. The first step is profiling, where all the hotspots in the code are identified. In the next step, the concurrent relationships at the hotpots are identified. Finally, the code is parallelized using techniques such as CUDA, OpenMP and MPI.

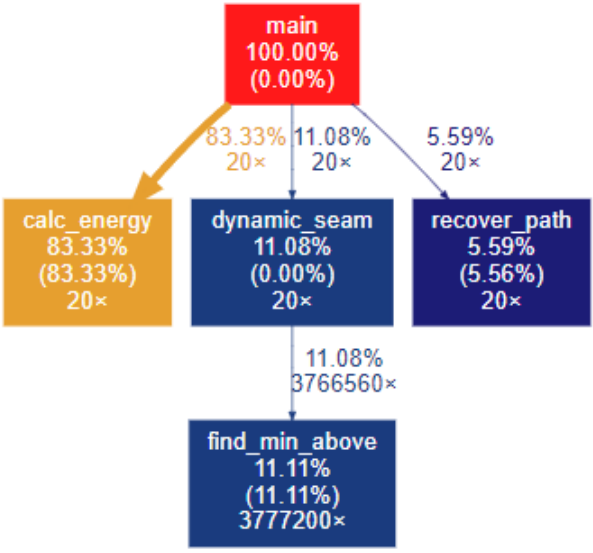


Fig. 2. Graphical Representation of Function Calls

The existing algorithm for Seam carving is analyzed using a profiling tool called GProf [9]. Profiling is used to identify the hotposts in the code, which are computation intensive. Flat profiling is used to find the functions that are getting called the most. GProf gives gmon.out which contains the code's complete profiling information. The gmon.out file can be represented in a graphical format called the call graph which is shown in Figure 2. Line profiling is performed using Gcov, in line profiling the lines in the code are numbered based on the number of times they get executed.

From the call graph, it is clear that the energy calculation function and the dynamic seam table calculation function are the most time consuming. The energy calculation function can be easily parallelized, since all the results are independent of each other, it executes nested loops for computing the value of the partial derivatives. It can be parallelized by using multi-threading. For a simpler implementation 2D array of the image can be converted into a 1D array, and in CUDA the 1D array is executed parallelly. For the Dynamic Seam calculation, dynamic programming is applied, hence all the concurrently calculated values are interdependent, to parallelize it the row values can be calculated parallelly, where each row can be implemented parallelly.

The idea behind parallel computing is to break down a larger task into separate independent tasks, and to execute them simultaneously [10]. There are a lot of parallel computing techniques, but in this work, three techniques have been mainly employed: Multi-threading, Multi-processing, and General Purpose Graphics Processing Unit(GPGPU) Computing. Multi-threading involves the creation of multiple threads within a single process, allowing tasks to be executed

concurrently. On the other hand, Multi-processing leverages multiple processors, each with its own memory space, to perform tasks in parallel. GPGPU Computing utilizes Graphics Processing Units (GPUs) for general-purpose computing. These techniques play significant role in optimizing performance and achieving efficient computing solutions for various applications.

4 Experimental Results

Parallelization using CUDA is implemented through Google Colab's Jupyter notebook, which is powered by two processors with identical specifications: Intel(R) Xeon(R) CPUs running at 2.20GHz. On the other hand, the OpenMP and MPI sections are implemented on a separate system running Ubuntu 20.04.5 LTS. This system is equipped with an Intel Xeon E-2224 CPU, which operates at a clock speed of 3.40 GHz and has four cores.

4.1 Performance Analysis of Parallelization using OpenMP

OpenMP is an API used for parallel programming. It is used by developers to parallelize specific regions of their code, without affecting other regions [11]. Using OpenMP API the energy calculation function is parallelized, different numbers of threads are used to identify the optimal thread pool for the function, the speedup peaked at 30 threads, and then the speedup started to decrease. Figure 4a compares the number of threads and the average speedup. Figure 4b compares the execution time to the number of threads. The program executes efficiently at 30 threads. It is observed that parallelization with more than 60 threads does not improve the performance. Images of 64 KB size are tested.

4.2 Performance Analysis of Parallelization using MPI

Message Passing Interface is used for parallel computation on distributed memory systems. It is used to divide a large computational task into smaller tasks, that can be run simultaneously [12]. The energy function is relatively easy to parallelize. The dynamic seam function is not parallelized. Figure 5a compares the number of processes and the average speed increase. Fig 5b compares the execution time to the number of processes. The most efficient speedup is obtained with seven processes. Images of 64 KB size are tested.

4.3 Performance Analysis of Parallelization by CUDA

Compute Unified Device Architecture(CUDA) is an API that can be used to perform certain general purpose computing problems in GPU. This technique is called as GPGPU. CUDA gives direct access to GPU's virtual instruction set and parallel computation features to compute kernels [13]. For the implementation in CUDA, the energy function can be easily parallelized. But the Dynamic Seam function is hard to parallelize since it follows dynamic programming,

4.4 Energy Function and Dynamic Seam

For the energy function, all the pixels by default in the program will be stored in an one dimensional array. The location of adjacent pixels can be mathematically calculated by just using the location of current pixel, height and width of the image. The 1 dimension image can be broken into a 1D thread block and each thread can be executed concurrently. In the energy function, there is no inter-dependence between pixels, so ideally the time complexity must go from O(n) to O(1) depending on the number of block and threads.

For the Dynamic seam, the parallelization is hard since it follows dynamic programming, the value of each pixel in a row depends on the previous row pixel value, but still each row can be executed parallelly by dividing them and synchronizing them after execution of a row, but this could create more delay because each row has to be synchronized. So significant speedup cannot be achieved. Figure 6a compares the number of threads and the speedup obtained with eight threads appeared to be optimum. The program without CUDA executes in 42 seconds and the program with CUDA and at optimal 8 threads executes in 22 seconds, 10MB size images are tested. Fig 6b compares the execution time to the number of threads. In Figure 3 it can be seen that the execution time for images of various sizes are reduced by half in the parallel implementation when compared to the serial implementation.

In OpenMP only minimal speedup is achieved, data synchronization and thread switching has greatly affected the maximum speedup that was targeted. For MPI significant speedup is not achieved due to the lack of cluster computers, and distributed memory systems. The speedup for MPI depends heavily on the underlying hardware. For CUDA 2x speedup is achieved when compared to the serial implementation. So GPGPU technique is the best approach for parallelization of seam carving algorithm.

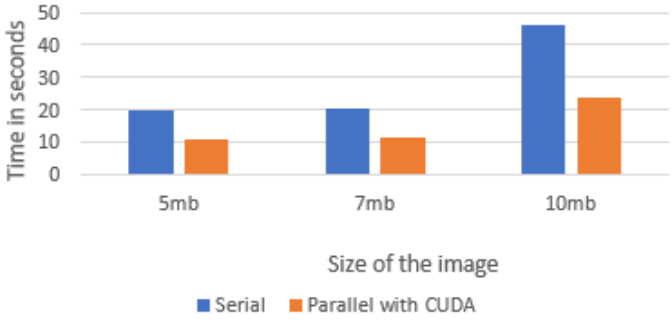
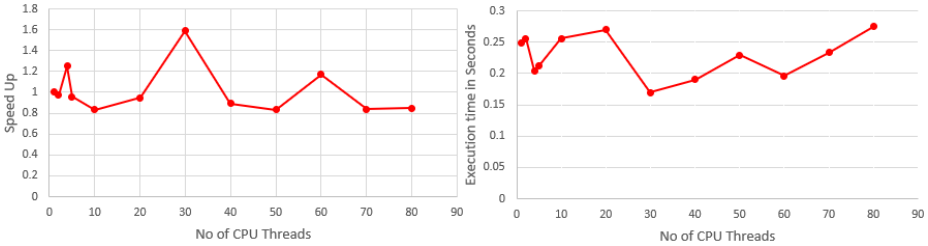
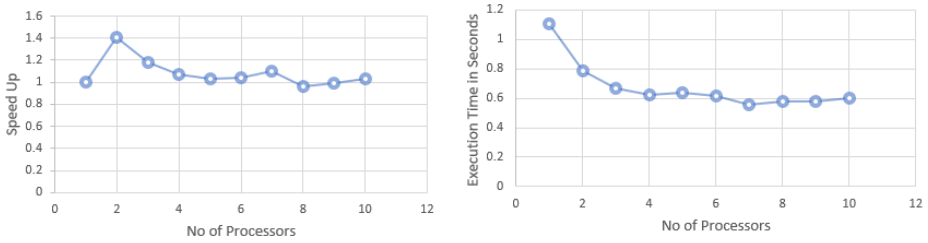


Fig. 3. CUDA vs Serial Comparison based on Execution Time



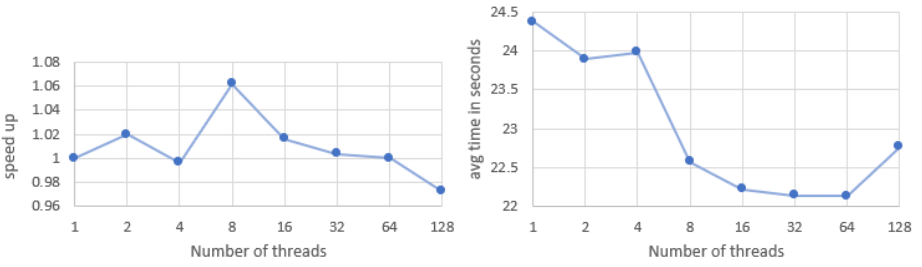
(a) Speedup vs Number of Threads (b) Execution Time vs Number of Threads

Fig. 4. Results using OpenMP based implementation



(a) Speedup vs No of Processors (b) Execution Time vs No of Processors

Fig. 5. Results using MPI based Implementation



(a) Speedup vs Number of Threads (b) Execution Time vs Number of Threads

Fig. 6. Results using CUDA based Implementation

5 Conclusions and Future Enhancements

From the experimental results, it becomes evident that Seam Carving is an incredibly efficient algorithm for performing content-aware image resizing. However, it does come with certain drawbacks. For instance, when dealing with very high-resolution images, the energy function calculation can be quite time consuming. Therefore, this paper suggests that a parallel implementation of the Seam Carving algorithm presents a viable solution. The results obtained from the parallel implementation demonstrates a significant reduction in execution time, particularly when utilizing GPGPU systems. To further enhance the algorithm, future work could focus on improving the parallelization of the Dynamic Seam function.

References

1. S. Avidan and A. Shamir, "Seam carving for content-aware image resizing," in *ACM Special Interest Group on Computer Graphics and Interactive Techniques 2007 papers*, pp. 10–es, 2007.
2. N. Kanopoulos, N. Vasanthavada, and R. L. Baker, "Design of an image edge detection filter using the sobel operator," *IEEE Journal of solid-state circuits*, vol. 23, no. 2, pp. 358–367, 1988.
3. T. Rauber and G. Rünger, *Parallel programming*. Springer, 2013.
4. J. Stultz and A. Edelman, "Seam carving: Parallelizing a novel new image resizing algorithm," *Project Report (Online) URL: http://courses.csail.mit.edu/18.337/2008/projects/reports/stultz-6338.pdf*, 2008.
5. R. Duarte and R. Sendag, "Accelerating and characterizing seam carving using a heterogeneous cpu-gpu system," in *PDPTA*, 2012.
6. I. Kim, J. Zhai, Y. Li, and W. Chen, "Optimizing seam carving on multi-gpu systems for real-time content-aware image resizing," *The Journal of Supercomputing*, vol. 71, pp. 3500–3524, 2015.
7. W. Dong, N. Zhou, J.-C. Paul, and X. Zhang, "Optimized image resizing using seam carving and scaling," *ACM Transactions on Graphics*, vol. 28, no. 5, pp. 1–10, 2009.
8. Z. K. Senturk and D. Akgun, "Seam carving based image retargeting: A survey," in *2019 1st international informatics and software engineering conference*, pp. 1–6, IEEE, 2019.
9. J. Fenlason and R. Stallman, "Gnu gprof," *GNU Binutils. Available online: http://www. gnu. org/software/binutils (accessed on 21 April 2018)*, 1988.
10. M. J. Quinn, "Parallel programming," *TMH CSE*, vol. 526, p. 105, 2003.
11. R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, and J. McDonald, *Parallel programming in OpenMP*. Morgan kaufmann, 2001.
12. W. Gropp, W. D. Gropp, E. Lusk, A. Skjellum, and A. D. F. E. E. Lusk, *Using MPI: portable parallel programming with the message-passing interface*, vol. 1. MIT press, 1999.
13. S. Cook, *CUDA programming: a developer's guide to parallel computing with GPUs*. Newnes, 2012.