Journal of Computer and Communications, 2024, 12, 110-119
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

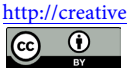
Parallel Technologies with Image Processing Using Inverse Filter

Rahaf Alsharhan, Areej Muheef, Yasmin Al Ibrahim, Afnan Rayyani, Yasir Alguwaifli

College of Computer Science and Information Technology (CCSIT), Imam Abdulrahman Bin Faisal University (IAU), Dammam, Saudi Arabia
Email: 2200004939@iau.edu.sa, 2200003186@iau.edu.sa, 2200004830@iau.edu.sa, 2200004475@iau.edu.sa, ymalguwaifli@iau.edu.sa

How to cite this paper: Alsharhan, R., Muheef, A., Al Ibrahim, Y., Rayyani, A. and Alguwaifli, Y. (2024) Parallel Technologies with Image Processing Using Inverse Filter. *Journal of Computer and Communications*, **12**, 110-119.
https://doi.org/10.4236/jcc.2024.121007

Received: December 15, 2023
Accepted: January 26, 2024
Published: January 29, 2024



Abstract

Real-time capabilities and computational efficiency are provided by parallel image processing utilizing OpenMP. However, race conditions can affect the accuracy and reliability of the outcomes. This paper highlights the importance of addressing race conditions in parallel image processing, specifically focusing on color inverse filtering using OpenMP. We considered three solutions to solve race conditions, each with distinct characteristics: #pragma omp atomic: Protects individual memory operations for fine-grained control. #pragma omp critical: Protects entire code blocks for exclusive access. #pragma omp parallel sections reduction: Employs a reduction clause for safe aggregation of values across threads. Our findings show that the produced images were unaffected by race condition. However, it becomes evident that solving the race conditions in the code makes it significantly faster, especially when it is executed on multiple cores.

Keywords

Parallel, Parallelization, Image Processing, Inverse Filtering, OpenMP, Race Conditions

1. Introduction

Welcome to our project on parallel technologies with image processing using inverse filter! In today's digital age, image processing plays a crucial role in various industries such as healthcare, security, entertainment, and more [1]. It involves manipulating digital images to enhance their quality, extract valuable information, or identify certain patterns. However, image processing algorithms can be computationally demanding, leading to longer processing times. Image processing

is the task of analyzing and manipulating images to enhance their quality or extract useful information. It is a rapidly growing field with a wide range of applications in various industries, such as filmmaking, diagnostic devices, manufacturing, and weather prediction [1].

The process of improving an image and extracting relevant information from it is known as image processing [1]. As the demand for real-time digital image processing increases, the emphasis is being shifted towards parallel/pipeline processing technologies, which are essential for handling large and complex images, such as medical scans, satellite images, or facial recognition [2]. Parallel technologies, such as parallel libraries and algorithms, are being increasingly integrated with image processing to expedite the restoration and manipulation of images. One such technique used in image restoration is inverse filtering, which is a fundamental concept in the field of image processing and is increasingly being integrated with parallel computing to accelerate the restoration process [1].

The integration of parallel technologies with image processing using inverse filter holds great promise for accelerating image restoration and manipulation, thereby contributing to advancements in various domains, including healthcare, entertainment, and manufacturing. The use of parallel computing in image processing can lead to more efficient results, as it supports data, task, and pipeline parallelism, which are beneficial for several image processing techniques, such as edge detection, histogram equalization, noise removal, image registration, picture segmentation, feature extraction, and many optimization strategies [1].

Image processing is an essential area of study that plays a crucial role in various applications. Digital images are manipulated and analyzed by utilizing mathematical approaches and algorithms to improve, restore, analyze, and comprehend images more effectively.

In image processing, color images, in particular, involve an additional challenge. The goal of color processing techniques is to modify the color channels to extract useful information and produce desired results. A noteworthy technique in image processing is color inverse filtering. It is used to describe the technique of reversing the intensity levels of each color channel in an image to reverse its colors. Color inverse filtering generates interesting effects, such as reversing color contrasts or producing a negative image.

In conclusion, this project investigates image processing with an emphasis on color inverse filtering and the integration of parallel technologies [1]. The purpose is to contribute to existing knowledge and shed light on the possible advantages of this technique. By leveraging parallel computing and inverse filtering, image processing tasks can be accelerated, enabling real-time analysis, manipulation, and restoration of digital images. This has significant implications for various industries, allowing for faster and more efficient image processing in applications such as medical diagnosis, security surveillance, entertainment, and manufacturing. The integration of parallel technologies with image processing

using inverse filter holds great potential for advancing the field and driving innovation in the years to come.

2. Literature Review

Tang *et al.* [3] developed a multifractal detrended fluctuation analysis (MF-DFA) program which involves image preprocessing. Then, the performance characteristics of each MF-DFA module were then examined, compared and they explored its parallelism. Eventually, they proposed a parallel optimization approach based on OpenMP for the MF-DFA. In their code, the added #pragma omp parallel to make sure that the iterations are divided between the threads equally. They used OpenMP clause reduction in order to aggregate all the results of each thread by the main thread. Their experimental outcomes demonstrate that the proposed parallel optimization approach has better performance.

Mallegowda, M *et al.* [4] developed an algorithm that uses image processing concept to improve the original bone scan so that it may be easily interpreted. In order to achieve higher performance for the same outcomes, OpenCV and OpenMP will both be used in the development of this program. At each stage of the procedure, they also keep an eye on the present condition of the input image and gain a deeper comprehension of the practical implications of the underlying mathematical ideas. Ultimately, the serial and parallel execution of the process are compared depending on how long each step takes to complete. Their experimental results show that the proposed parallel computing approach exceeds the serial computing approach in terms of performance.

Han Xiao *et al.* [5] focuses extensively on the parallelization of image processing algorithms, particularly emphasizing the advantages of parallel computing using OpenCL. In the context of the weighted mean filtering algorithm, parallelization is crucial for achieving efficient and rapid processing of large-scale image datasets. Their paper recognizes the escalating demand for real-time data processing in the face of increasing image data volumes, asserting that traditional single-processor or multiprocessor computing equipment falls short in meeting these demands. The proposed OpenCL-based parallel algorithm operates at two levels: workgroup and work-item, tapping into the parallel processing capabilities of modern high-performance GPUs.

This study delves into the intricacies of image discrete convolution computing and the multi-layer logic architecture of high-performance computers, strategically leveraging these features to optimize task mapping from the computing model to computing resources. Throughout the paper, the term "parallelism" is recurrent, underlining the researchers' commitment to exploring and harnessing parallel computing capabilities.

Greg Slabaugh *et al.*'s paper titled "Multicore Image Processing with OpenMP" discusses the use of OpenMP (Open Multi-Processing) in parallelizing image processing applications on multicore processors [6]. It highlights the benefits of multicore processors in achieving higher performance and explores how pro-

grammers can leverage parallelism to optimize their code for multiple cores. OpenMP is introduced as an industry-standard API for parallel programming on shared memory multi-processors. The paper explains the basics of OpenMP, including its directives and clauses for specifying parallel regions and loop-level parallelism. It emphasizes the simplicity and effectiveness of OpenMP in implementing parallel image processing operations. The paper also touches upon topics such as variable scope, scheduling mechanisms, and the availability of OpenMP in various compilers and operating systems. Several examples, including loop parallelization and image warping, are provided to demonstrate the application of OpenMP in image processing. Overall, the paper serves as a high-level overview of OpenMP and its potential for optimizing image processing algorithms on multicore CPUs.

3. Methods

OpenMP allows for the implementation of shared-memory multiprocessing, which boosts computational efficiency in many fields, for example the image processing field. OpenMP enables parallel image processing by employing multiple threads to distribute workload among available CPU cores. We set the number of threads to 1, 2, 4, 8, and 16 using omp_set_num_threads (number of threads). Because of this parallelization, several image processing methods run faster and perform better. Through the offering of environment variables, library routines, and compiler directives that enable shared-memory multiprocessing, OpenMP improves the implementation of parallelism. This makes it simple for developers to distribute the effort associated with image processing across several threads, making better use of the processing power that is available. Image processing methods can benefit from the parallel nature of today's processors by utilizing OpenMP, which can result in a noticeable speed increase and enhanced real-time processing capabilities. OpenMP allows developers to concentrate on algorithm design and high-level code optimization by simplifying the process of using parallelism without having to learn the ins and outs of low-level multiple threads. Additionally, OpenMP is a cross-platform approach that works with several different operating systems and programming languages, including Fortran, C, and C++. This adaptability maximizes the advantages of parallel processing over a broad range of hardware architectures by enabling image processing programs to be developed and implemented on several platforms.

In this paper, we employ color inverse filtering in our code, which is an image processing technique that flips an image's colors to produce an "opposite" effect. With shared-memory multiprocessing, OpenMP may greatly facilitate the color inverse filtering process. Image processing processes can be split up across several threads by utilizing OpenMP, which enables parallel execution on available CPU cores. The process of color inverse filtering is flipping an image's pixel values in each of the three-color channels usually red, green, and blue. Because of OpenMP's parallelization features, the workload may be distributed efficiently

among several threads, enabling them to process multiple parts of the image at the same time. The processing time needed for color inverse filtering is greatly decreased by using this parallel technique, especially for large images or real-time applications.

To run the code, we used a Ryzen 7 5800H CPU with 16 logical processors and 8 cores running at 4.4 GHz and 16 GB RAM. Also, on a Windows system, utilize the built-in Microsoft Visual Studio 2022 OpenMP compiler by selecting "C/C++" as the programming language and setting the "OpenMP support" option to yes in the project's property page. With the support of outside sources and references, we took the already developed sequential code for image processing utilizing color inverse filtering and converted them into parallel using OpenMP [7]. The essential libraries, such as iostream, opencv2/opencv.hpp, and omp.h, which provides OpenMP main functionality, are included. To execute the code in parallel we make sure to include #pragma omp parallel to demonstrate the inverse filtering in parallel without any problems. Several significant variables, including image1, image2, output_image1, output_image2, and pixel, are involved in the color inverse filtering image processing process in this code. A thorough analysis of the methodology's use of these variables is provided below:

1) Image1 and image2: The input images for processing are stored in these variables. The code assumes that the cv::imread() function was used to correctly load these images. The function won't produce the wanted output if these variables aren't properly initialized with legitimate image data.

2) Output_image1 and output_image2: The output images for processing are stored in these variables. The size and type are adjusted to correspond with the input images.

3) Pixel: pixel is important for accessing and modifying the pixel values in the image. The pixel variable's separate channel values (pixel [0], pixel [1], pixel [2]) are then modified to carry out several operations, like color inverse filtering and applying a sharpening filter. The neighbor pixels in the immediate vicinity provide the basis for these changes.

Moreover, The execution time is measured by first recording the current time with omp_get_wtime() and then subtracting the recorded start time from the current time with omp_get_wtime() after the parallel sections. So, in order to enhance computational efficiency, the #pragma omp parallel parts directive is used to parallelize the image processing processes. This enables the execution of the directive's parts at the same time. Furthermore, the race condition is a significant concern in the process.

We concentrated on how these race conditions impacted overall performance and execution time. We attempted to carefully assess several race condition solutions utilizing varied numbers of processing threads. This research also investigates how the number of threads affects the time it takes for the program to perform image processing. This demonstrates our methodical approach

to dealing with race condition issues, analyzing visual outcomes, and optimizing performance with various threads and solutions. The three solutions to the race conditions problem will be discussed in greater detail in the following section.

4. Results

In our code the variables output_image1 and output_image2 (Figure 1 & Figure 2) produce race condition and we tried to fix the race using various options like critical, atomic and reduction, and after we run the code before and after the race condition, we conclude that the images did not get effected by these variables only the execution time get decrease.

This is the result we get after solving the race condition (Figure 3):

- Solving the race condition by #pragma omp atomic.
- As we can see in Figure 4 the time decreases after solving the race condition. Therefore, the code without a race is 0.002353 seconds faster.

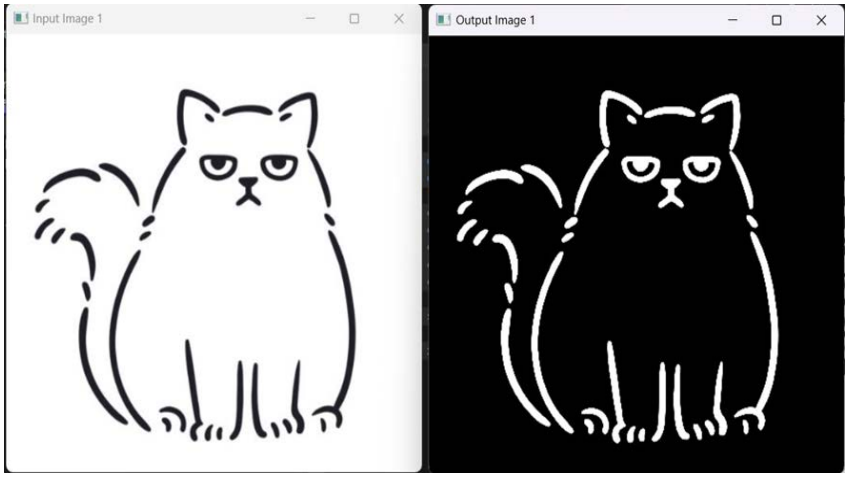


Figure 1. Result of image 1.

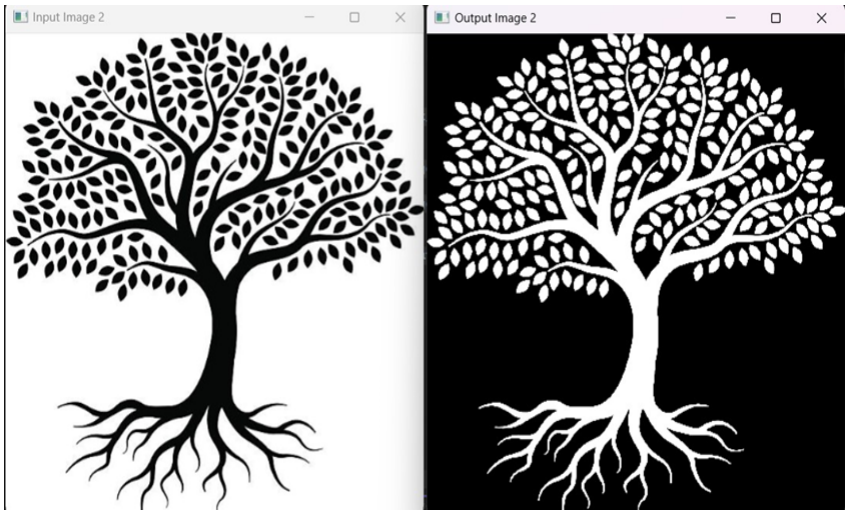


Figure 2. Result of image 2.

- Solving the race condition by #pragma omp critical.
- As we can see in Figure 5 the execution time is decreased a lot after solving the race condition by critical. Therefore, the code without a race is 0.002517 seconds faster.
- Solving the race condition by pragma omp parallel sections reduction (+: output_image1) reduction (+: output_image2).
- As we can see in Figure 6 the execution time is decreased after solving the race condition by reduction. Therefore, the code without a race is 0.0037033 seconds faster.

While the code's visual output remains unaffected, it's crucial to note that race conditions can still significantly impact performance and execution time. Figure 1 and Figure 2 demonstrate that image processing proceeds correctly even in the presence of race conditions. However, as illustrated in Figure 3, execution time can get affected compared to the results after solving the race condition in Figures 4-6. Further analysis of execution time using multi-threading, as depicted in Figure 7, reinforces this impact. Therefore, addressing this issue is paramount

Figure 3. Execution time with race conditions.

Figure 4. Execution time after atomic.

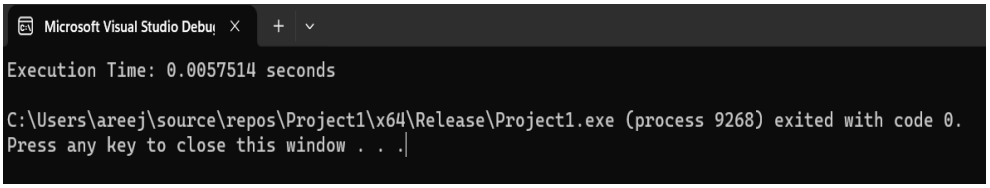


Figure 5. execution time after critical.

Figure 6. Execution time after reduction.

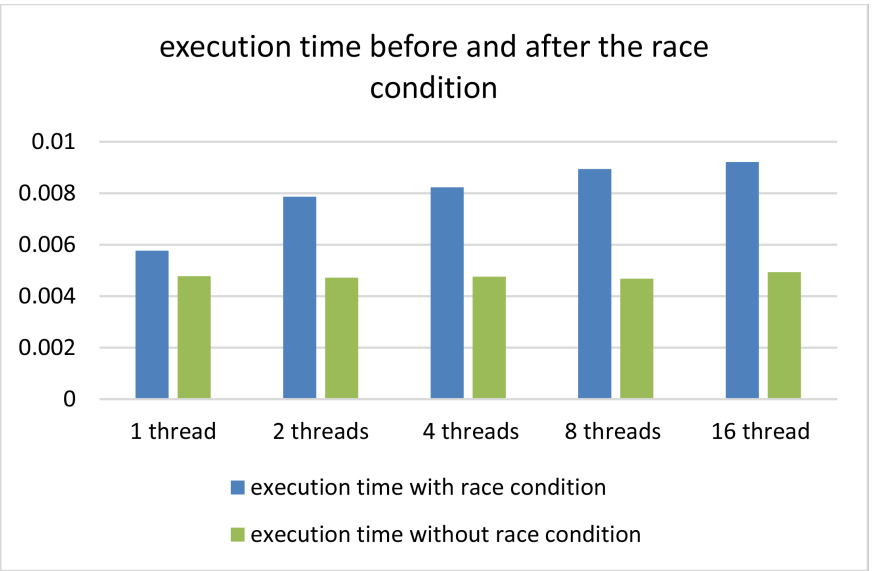


Figure 7. Execution time before and after the race condition.

to ensure optimal code performance and program speed.

5. Findings

Parallelizing image processing algorithms using OpenMP offers several advantages in terms of computational efficiency and real-time processing capabilities. However, race conditions pose a significant challenge in obtaining precise and trustworthy results. This paper highlights the importance of addressing race conditions in parallel image processing with an emphasis on color inverse filtering using OpenMP. Moreover, we considered three solutions: #pragma omp atomic, #pragma omp critical, and #pragma omp parallel sections reduction.

Our results demonstrate that while race condition can affect performance and the execution time, the produced images were unaffected by the race conditions that were present in our code. In terms of computational efficiency, our results demonstrated significant improvements in execution time after implementing the race condition solutions. We noticed a decrease in execution time for all three solutions. Particularly, the #pragma omp atomic solution resulted in 0.002353 seconds faster, the #pragma omp critical solution was faster by 0.002517 seconds, and the #pragma omp parallel sections reduction solution was faster by 0.0037033 seconds. These time reductions show that the code's computational efficiency was significantly enhanced by eliminating race conditions.

Our findings reveal that race conditions were successfully eliminated from the code for color inverse filtering without having a negative impact on the quality of the produced images. The benefits of mitigating race conditions are further highlighted by the observed improvements in execution time.

6. Conclusion

In today's image-driven world, fast and efficient processing is crucial. Our project

Table 1. The result of literature reviews.

image processing concepts	Techniques	Results	References
MATLAB used to turn the rape leaf image into a grayscale image	Their technique based on OpenMP.	Their optimization approach significantly improves the performance and the maximum speedup achieved can reach 1.59.	Tang *et al.* [3]
Serial and Parallel Technologies with Image Processing Using Inverse Filter	Their technique makes use of OpenMP and OpenCV.	They achieved a speed-up and a reduction in processing time by using a parallel computing approach to bone scan image processing compared to a serial computing approach.	Mallegowda, M *et al.* [4]
Image Mean Filtering Based on OpenCL	Their technique involves a hierarchical weighted mean filtering parallel algorithm specifically designed for OpenCL, which leverages multi-layer GPU architecture to efficiently distribute image processing tasks across workgroups and work-items for accelerated performance.	By parallelizing their image processing algorithms on GPUs with OpenCL, the researchers achieved efficient and high-speed processing of large datasets, preparing their work for real-time applications in the face of ever-growing data volumes.	Han Xiao *et al.* [5]
Multicore image processing using OpenMP	Their technique used OpenMP	By leveraging parallelism through OpenMP, programmers can modify their single-threaded code to run efficiently on multiple cores, thereby potentially enhancing the performance of image processing algorithms.	Greg Slabaugh *et al.* [6]

successfully combined parallel technology (OpenMP) with image processing via color inverse filtering. This sped up image restoration and manipulation. By parallelizing tasks, we significantly reduced processing time, especially for large images. Careful handling of race conditions ensured optimal performance. This research paves the way for further exploration of parallel technologies in image processing, unlocking a world of exciting possibilities.

Acknowledgments

We would like to thank DR. YASIR ALGUWAIFLI for his essential guidance and support throughout this project. His skills and feedback were critical in influencing the course of our research. Furthermore, we thank our institution, Imam Abdulrahman Bin Faisal (IAU), particularly our College of Computer Science and Information Technology, for providing us with a valuable and interesting course on Parallel Computer Architecture.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Algorithms (2023) How Can You Incorporate Parallel Computing into Algorithmic Research for Image Processing? https://www.linkedin.com/advice/0/how-can-you-incorporate-parallel-computing-algorithmic-ektkc

[2] Tutors India (n.d.) Parallel Computing in Image Processing. https://www.tutorsindia.com/our-sample-works/parallel-computing-in-image-processing/

[3] Tang, X., Yang, X. and Wu, F. (2019) Multifractal Detrended Fluctuation Analysis Parallel Optimization Strategy Based on openMP for Image Processing. *Neural Computing and Applications*, **32**, 5599-5608. https://doi.org/10.1007/s00521-019-04164-2

[4] Mallegowda, M., and Karthik, N. and Anvith, A. (2023) Serial and Parallel Computation of Bone Scan Image Processing. *Proceedings of the International Conference on Innovative Computing & Communication* (*ICICC*) 2022. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4361148 https://doi.org/10.2139/ssrn.4361148

[5] Xiao, H., Guo, B.Y., Zhang, H.Y. and Li, C.L. (2021) A Parallel Algorithm of Image Mean Filtering Based OpenCL. https://www.researchgate.net/publication/350365944_A_Parallel_Algorithm_of_Image_Mean_Filtering_Based_on_OpenCL

[6] Slabaugh, G., Boyes, R. and Yang, X.Y. (n.d.) Multicore Image Processing with OpenMP. https://www.eecs.qmul.ac.uk/~gslabaugh/publications/OpenMP_SPM.pdf

[7] Lajoie, L. (2023) Python: Python-Based Inverse Filtering. https://copyprogramming.com/howto/inverse-filtering-using-python