Multiprocessing-Enhanced Parallelization for LSB Image-in-Image Steganography

Andrea Stevanoska 226001

October 2024

Abstract

The parallelization of Least Significant Bit (LSB) steganography algorithms holds great potential for improving the efficiency of encoding and decoding processes. By leveraging multiprocessing techniques, we achieved significant improvements in processing times, particularly for large images. This paper compares sequential and parallel implementations, revealing substantial scalability and performance gains through parallelization. Experimental results show that the parallel approach can achieve speedups of approximately up to 17 times for large images, all while maintaining the visual quality of the original and hidden images in stego images.

Keywords: Parallel Computing, Steganography, Least Significant Bit, Multiprocessing, Image-in-Image Steganography

1 Introduction

Steganography is the practice of hiding information inside other data, like images, audio, or video files. The hidden information, often called the "secret message," is added in a way that makes it hard to notice. Steganography is widely used in scenarios requiring secure communication or data protection. For example, people might use steganography to send private information secretly, or companies might want to hide digital watermarks in images to show ownership.

In the case of images, one popular method is Least Significant Bit (LSB) substitution, where the least significant bits of the image's pixel values are replaced with the most significant bits of the corresponding pixels of the secret message. Since the change is small, the human eye usually can't see any difference.

Despite its effectiveness, traditional LSB steganography suffers from inefficiencies when dealing with large images. The encoding and decoding processes, which involve processing each pixel individually, are sequential and computationally expensive. As images grow in size and resolution, the time required to process them increases significantly, making the sequential LSB method impractical for real-time applications or for handling large datasets. Fortunately, parallelization can speed up the process. By using multiple processors or cores to handle different parts of the task simultaneously, the overall time for steganography can be reduced significantly while maintaining the integrity of the hidden content.

This paper focuses on parallelizing the LSB image steganography technique using Python's multiprocessing module. The goal is to speed up the process of hiding and extracting information from images by breaking the tasks into smaller chunks and processing them in parallel.

2 Related Work

Steganography has seen significant advances, particularly with parallelized image steganography methods. There is one paper that focuses on an LSB substitution-based technique for embedding secret images in cover images, exploring GPU-based and NOC-based parallelism approaches [1].

Parallel programming has proven beneficial for steganography and digital watermarking on multiprocessor systems. A study shows that parallelization optimizes the embedding and extraction of information, improving performance on multicore devices like laptops, desktops, and mobile phones [2]. Parallelization is also key to improving the efficiency of steganography over cryptography for securing messages. An OpenCL-based study demonstrates the speedup and quality retention of hiding large secret images, evaluated using PSNR, MSE, and other performance metrics [3].

Another study proposes parallel RSA with 2D-DCT and chaotic 2D-DCT steganography, achieving significant speedups of 1.6 and 3.18 times over serial methods, especially for larger messages [4]. Similarly, an OpenMP-based chaos-LSB implementation for multi-core CPUs shows linear scalability, with speedup as the number of cores increases from 1 to 16 [5]. OpenMP is also used to reduce embedding time by 50 percent, with PSNR values ranging from 30 to 50 after data embedding [6].

Combining LSB with Visual Cryptography and a Genetic Algorithm further enhances security. The secret message is encrypted and embedded in the LSB, with the Genetic Algorithm altering pixel locations. This approach distributes the workload across multiple systems, improving speed and scalability [7]. Additionally, a parallel feature selection algorithm based on Forward-Backward Selection improves classifier accuracy for detecting hidden messages by focusing on the most relevant image features [8].

A parallel Cellular Automata-based method for multiple secret image sharing with LSB steganography addresses the issue of noisy, encrypted images by embedding them in camouflage images. Using CUDA technology, this method achieves a 7x speedup in encoding and decoding compared to sequential methods, improving performance [9].

Another framework hides a color image in stereo images using DCT and

QIM-DM, with Arnold's Cat Map Transform added for security. Optimized for multi-core CPUs, it outperforms previous methods, achieving high PSNR and SSIM values and robustness against JPEG compression [10]. Lastly, a method using modular arithmetic and pixel intensity adjustments to hide multiple bits improves security and reduces distortion. By incorporating parallelism, the method enhances efficiency and performance [11].

These studies demonstrate how parallelism significantly improves the efficiency, security, and scalability of steganography systems, offering faster and more robust solutions for hiding secret messages.

3 System Architecture

Before we dive into the parallelization approach, it's important to first understand how image steganography works. Image steganography is a method used to embed a secret image into a cover image in such a way that the changes are undetectable to the human eye.

3.1 Standard Sequential Approach

In the standard approach to image steganography, the secret image is hidden by replacing the least significant bits of the cover image's pixels with the most significant bits of the secret image's pixels. This process is carried out sequentially, where each pixel of the cover image is processed one at a time. The secret image is embedded into the LSBs of the pixels of the cover image, and the decoding process reads the LSBs to reconstruct the secret image. The time complexity of this approach is $O(n)$, where n is the number of pixels in the image.

3.2 Parallel Approach

To improve the performance of the encoding and decoding processes, a parallelization approach is applied. By utilizing multiprocessing, the task is divided into smaller, independent subtasks that can be processed concurrently. The key idea is to break the image into smaller chunks, which can be processed simultaneously by multiple processes. Python's multiprocessing module enables the creation of separate processes that run concurrently, bypassing the Global Interpreter Lock (GIL) and allowing true parallel execution.

In the parallel encoding process, cover image and secret image are divided into smaller, non-overlapping chunks. The number of chunks is determined by the number of available CPU cores (in this case, 8 chunks are used to match the number of logical processors on the test machine). The chunks are of equal size, ensuring balanced workload distribution across processes. Each process handles a separate chunk of the image and embeds the bits of the secret image into the least significant bits (LSBs) of the corresponding cover image pixels. After processing its chunk, the process shares the results with others to reconstruct the final stego image. This coordination among multiple processes significantly accelerates the encoding process compared to the sequential approach.

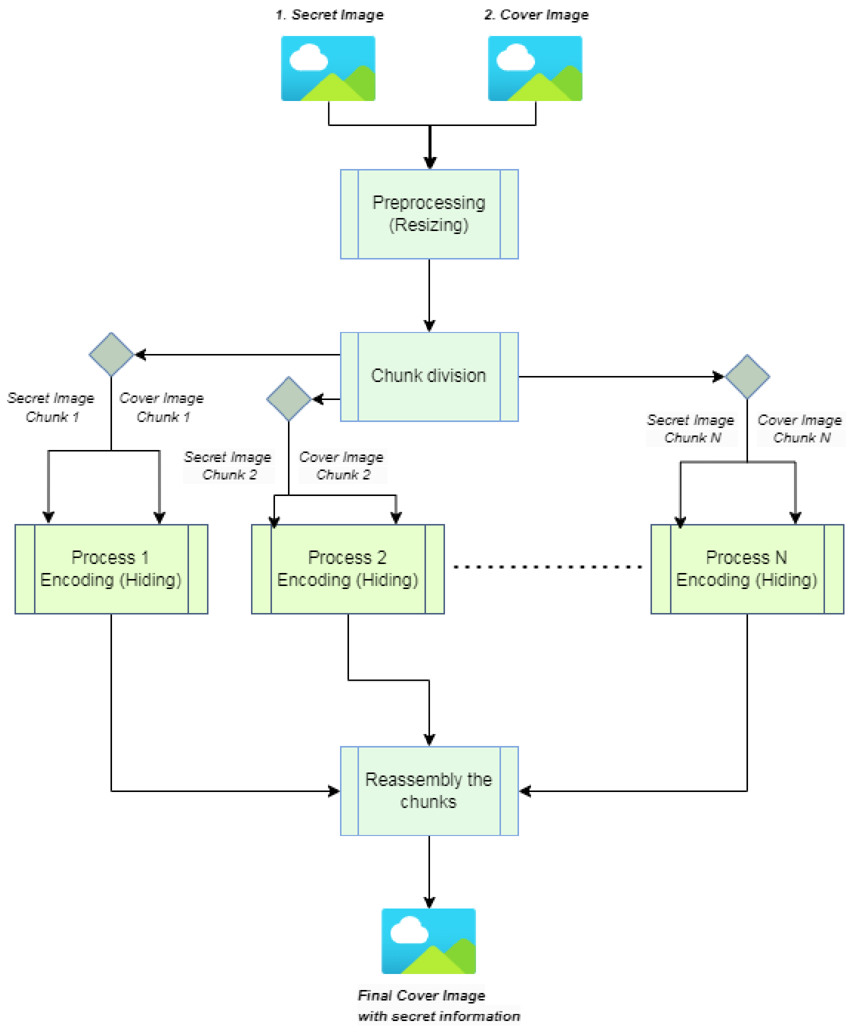


Figure 1: Parallelized Encoding (Hiding)

Similarly, the parallel decoding (extraction) process involves dividing the stego image into smaller chunks. Each process extracts the LSBs from its assigned chunk and reconstructs the secret image. Once all processes finish their extraction tasks, the results are merged to reconstruct the original secret image. By distributing the workload across multiple processes, the decoding process is sped up, especially for large images.

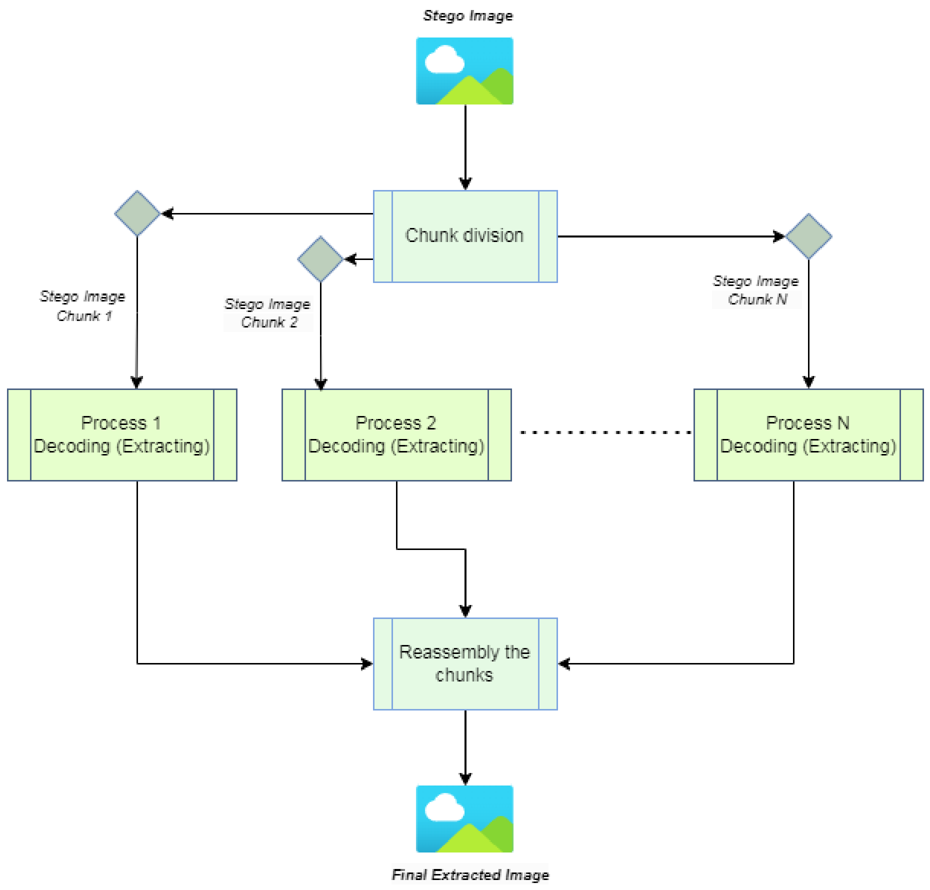


Figure 2: Parallelized Decoding (Extraction)

While parallelization offers substantial speed improvements, there are small trade-offs to consider. Because the image is divided into chunks, there may be instances where the optimal encoding or decoding path requires information from multiple chunks. This can potentially lead to minor inaccuracies or artifacts in the final stego image. However, these trade-offs are generally minimal and are outweighed by the performance gains.

3.3 Challenges and Solutions

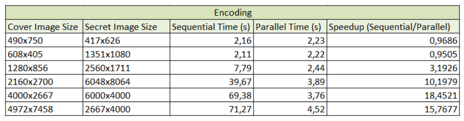
Large images require significant memory, which can lead to performance bottlenecks. To solve this, the images are divided into smaller chunks, reducing the memory footprint of each process. Since each chunk is processed independently, there is no need for explicit synchronization between processes. This simplifies the implementation and reduces overhead. Also for small images, the overhead of creating and managing multiple processes can outweigh the benefits of parallelization. This can be addressed by dynamically adjusting the number of chunks based on the image size.

4 Results

The performance evaluation highlights the differences between sequential and parallel implementations for encoding and decoding steganographic data. The use of multiprocessing played a key role in improving performance, especially for larger images, by splitting the workload across multiple CPU cores. This resulted in significantly faster processing times while maintaining the quality of the images, with no noticeable loss in visual clarity or distortion.

4.1 Encoding Performance Analysis

Encoding performance varied significantly between the sequential and parallel approaches, especially for larger images. Sequential processing exhibited a sharp increase in time as image size grew, revealing its limitations for computationally intensive tasks. The parallel implementation, enabled by multiprocessing, demonstrated remarkable efficiency by dividing the workload across multiple cores. This approach achieved speedups exceeding 15 times for larger images, with the most notable gains observed for medium-to-large images, where the computational workload was distributed more effectively.



Encoding				
Cover Image Size	Secret Image Size	Sequential Time (s)	Parallel Time (s)	Speedup (Sequential/Parallel)
490x750	417x626	2,16	2,23	0,9686
608x405	1351x1080	2,11	2,22	0,9505
1280x856	2560x1711	7,79	2,44	3,1926
2160x2700	6048x8064	39,67	3,89	10,1979
4000x2667	6000x4000	69,38	3,76	18,4521
4972x7458	2667x4000	71,27	4,52	15,7677

Figure 3: Encoding Time Comparison for Test Images.

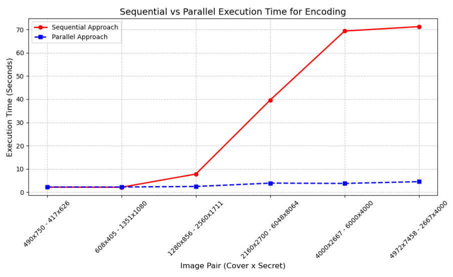
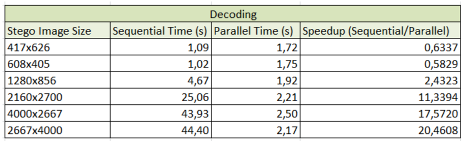


Figure 4: Encoding Seq vs Parallel.

4.2 Decoding Performance Analysis

Decoding followed similar patterns, with sequential processing times increasing steadily with image size. The parallel decoding process initially showed some overhead for smaller images due to the extra work needed to coordinate between processes, occasionally performing worse than the sequential approach. However, for larger images, the benefits of multiprocessing became clear, with speedups of more than 17 times. Importantly, the quality of the decoded images was consistent across both methods, with no visual differences detected.



Decoding			
Stego Image Size	Sequential Time (s)	Parallel Time (s)	Speedup (Sequential/Parallel)
417x626	1,09	1,72	0,6337
608x405	1,02	1,75	0,5829
1280x856	4,67	1,92	2,4323
2160x2700	25,06	2,21	11,3394
4000x2667	43,93	2,50	17,5720
2667x4000	44,40	2,17	20,4608

Figure 5: Decoding Time Comparison for Test Images.

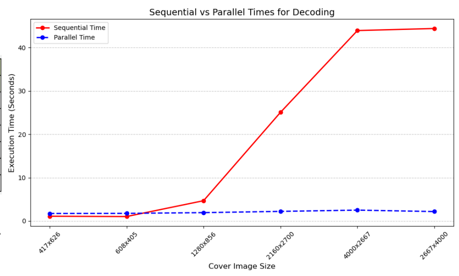


Figure 6: Decoding Seq vs Parallel.

4.3 Performance Comparison and Scalability

The sequential approach worked reasonably well for small images but struggled with larger ones. On the other hand, the parallel implementation scaled effectively with image size, becoming much faster for larger images. Even though performance gains leveled off for extremely large images due to system resource limits, the parallel approach still outperformed the sequential one.

5 Conclusion and Future Work

These results highlight the advantages of using multiprocessing for image steganography tasks. The parallel implementation significantly reduced processing times while preserving the visual quality of the images, making it a practical solution for applications that involve high-resolution images. The benefits of parallelization were particularly evident for medium-to-large image sizes, where the computational workload was effectively distributed across multiple cores. Although there were minor overheads for smaller images, the overall performance was still enhanced, demonstrating the practical utility of multiprocessing for parallelizing steganography algorithms.

While the current implementation successfully optimizes the LSB method, there are several areas that can be explored for future work. One potential direction is to investigate the use of Graphics Processing Units (GPUs) for parallel processing, which could further accelerate the encoding and decoding processes by exploiting their massive parallel computation capabilities. Additionally, exploring distributed computing frameworks such as Apache Spark or MPI (Message Passing Interface) could allow the algorithm to scale efficiently for large datasets or cloud-based applications.

Another avenue for future research could involve refining the algorithm's performance through more advanced parallelization strategies, such as task scheduling and load balancing, to reduce the impact of overheads.

In conclusion, the parallelization of LSB steganography provides a promising path toward improving the performance and applicability of image-based data hiding techniques, with potential advancements that could further optimize efficiency, scalability, and real-world usability.

References

[1] S.H. Dizaji, M. Zolfy Lighvan, A. Sadeghi, "Hardware-Based Parallelism Scheme for Image Steganography Speed up," in *International Conference on Innovative Computing and Communications*, vol. 1166, Springer, Singapore, 2021.

[2] S. Patel, "A Proposed Mechanism Using Parallelization Techniques in Steganography and Digital Watermarking for Multi Processor Systems," 2008.

[3] N.G. Kini, V.G. Kini, "A Parallel Algorithm to Hide an Image in an Image for Secured Steganography," in *Integrated Intelligent Computing, Communication and Security*, vol. 771, Springer, Singapore, 2019.

[4] G. Savithri, Vinupriya Mane, S. Saira Banu, "Parallel Implementation of RSA 2D-DCT Steganography and Chaotic 2D-DCT Steganography," in *Proceedings of International Conference on Computer Vision and Image Processing*, vol. 459, Springer, Singapore, 2017.

[5] G. Gambhir, J.K. Mandal, "Multicore implementation and performance analysis of a chaos based LSB steganography technique," *Microsyst Technol*, vol. 27, pp. 4015–4025, 2021.

[6] U. B. A. P. Karthikeyan, M. Thangavel, "High Efficient Data Embedding in Image Steganography Using Parallel Programming," in *Applications of Security, Mobile, Analytic, and Cloud (SMAC) Technologies for Effective Information Processing and Management*, June 2018.

[7] H. Maru, P. Pranav, Y. Miryala, B. Rudra, "Distributed Computing Solution for Steganography Using Visual Cryptography and Genetic Algorithm," in *Internet of Things and Connected Technologies*, vol. 1382, Springer, Cham, 2021.

[8] A. Guillén, A. Sorjamaa, Y. Miche, A. Lendasse, I. Rojas, "Efficient Parallel Feature Selection for Steganography Problems," in *Bio-Inspired Systems: Computational and Ambient Intelligence*, vol. 5517, Springer, Berlin, Heidelberg, 2009.

[9] A. Hernandez-Becerril, M. Nakano-Miyatake, M. Ramirez-Tachiquin, H. Perez-Meana, "A parallel implementation of multiple secret image sharing based on Cellular Automata with LSB steganography," *IEEE 12th International Conference on Intelligent Software Methodologies, Tools and Techniques*, Budapest, Hungary, 2013, pp. 191-196.

[10] D.O. Munoz-Ramirez, V.I. Ponomaryov, R. Reyes-Reyes, C. Cruz-Ramos, "Parallel steganography framework for hiding a color image inside stereo images," *Proc. SPIE 10223*, Real-Time Image and Video Processing 2017, 102230L, May 2017.

[11] B. Datta, S. Roy, S. Roy, et al., "Multi-bit robust image steganography based on modular arithmetic," *Multimed Tools Appl*, vol. 78, pp. 1511–1546, 2019.