High-Performance Parallel Acceleration of Image Processing Filters on Multicore CPUs Using OpenMP

Gabriel Diniz Cremel
dinizcremelgabriel@gmail.com
Centro Universitário Avantis - UNIAVAN
Balneário Camboriú, Santa Catarina, Brasil

Luiz Fernando M. Arruda
luiz.arruda@ieee.org
Centro Universitário Avantis - UNIAVAN
Balneário Camboriú, Santa Catarina, Brasil

Abstract

This work evaluates the acceleration of image-processing filters on multicore CPUs using the OpenMP parallel programming model. Sequential and parallel implementations were analyzed across different resolutions and filter types to measure speedup, efficiency, and scalability. Results show that OpenMP significantly reduces execution time, especially for high-resolution images, where parallel processing compensates for thread-management overhead. The study also demonstrates the feasibility of integrating OpenMP-based preprocessing into AI pipelines, confirming that multicore CPUs remain a viable and accessible solution for real-time computer vision tasks.

Keywords

Computer Vision; OpenMP; Convolution; Parallel Computing; Image Processing;

1 Introduction

Advances in Artificial Intelligence (AI), machine learning, and computer vision have transformed how modern systems interpret and interact with visual data [1]. From autonomous vehicles [2] and medical diagnostics [3] to smart agriculture [4], smart cities [5], and industrial automation [6], the demand for reliable, fast, and scalable image-processing solutions continues to grow [7]. As these applications evolve, they increasingly rely on large volumes of high-resolution images and real-time video streams, placing substantial pressure on computational resources [8]. Ensuring low latency while maintaining accuracy has therefore become a central challenge in contemporary computer-vision pipelines [9].

Before visual information reaches deep-learning models such as convolutional neural networks (CNNs), YOLO, SSD, or Faster R-CNN, it undergoes a sequence of preprocessing operations designed to enhance quality, suppress noise, normalize illumination, and highlight relevant features [10]. These operations, although conceptually simple, are computationally demanding because they must be applied to every pixel of the image [11]. In high-resolution formats such as Full HD, 4K, or even 8K, a single convolution pass may involve billions of arithmetic operations. As a result, sequential preprocessing becomes insufficient in applications that operate under strict time constraints, where even minor delays can compromise system performance or safety [12].

The increasing availability of multicore processors offers a promising solution to this challenge. By distributing the workload across multiple threads, it becomes possible to significantly reduce processing time without altering the underlying algorithms [13]. OpenMP emerges as an accessible and efficient parallel-programming model for shared-memory architectures, enabling developers to incorporate parallelism through simple compiler directives. Because many image-processing operations, especially convolution, are highly parallelizable, OpenMP allows substantial performance gains with minimal code restructuring [14]. This makes it a desirable choice for embedded systems, latency-sensitive applications, and environments where GPUs are unavailable or inappropriate [15].

Given this context, the goal of this work is to evaluate the performance impact of accelerating image-processing operations using OpenMP. Sequential and parallel implementations are compared across multiple resolutions and filters, examining speedup, parallel efficiency, scalability, and the influence of image size on parallel performance. The study aims to provide practical evidence of how multicore CPUs can be effectively leveraged to accelerate preprocessing stages in modern computer vision workflows.

The main contributions of this work are summarized as follows:

- An experimental evaluation of OpenMP-based parallelization applied to classical image preprocessing filters on multicore CPUs, highlighting the impact of thread-level parallelism on execution time.
- A comparative analysis of speedup and parallel efficiency across different filter types and image resolutions, demonstrating how workload characteristics influence scalability on shared-memory architectures.
- Practical insights into the conditions under which OpenMP provides the greatest performance benefits, particularly for computationally intensive filters and high-resolution images, using a hybrid Python–C++ experimental framework.

The remainder of this article is organized as follows: Section 2 provides an overview of image processing techniques and the role of convolutional filters. Section 3 details the experimental setup, including the computational environment, image datasets, and performance results. Finally, Section 4 summarizes the findings and suggests directions for future work.

2 Image Processing

Image processing is a foundational component of computer-vision systems and plays a crucial role in transforming raw visual information into structured data suitable for higher-level analysis. Real-world images often exhibit imperfections such as illumination variations, sensor noise, compression artifacts, and irrelevant background content, as illustrated in Fig. 1 [16]. If not addressed early in the pipeline, these issues propagate and degrade the performance of subsequent stages such as object detection, segmentation, and classification. For this reason, preprocessing is a vital stage that enhances the reliability and stability of computer-vision applications.

Figure 1: Raw (left) and preprocessed (right) Lena image [17].

The typical computer-vision workflow (Fig. 2) begins with image acquisition, followed by a sequence of operations designed to improve quality, normalize intensity distributions, and emphasize relevant structures. These transformations not only improve human interpretability but also enhance the robustness and accuracy of AI-based algorithms. Neural networks, in particular, are sensitive to the statistical characteristics of their inputs, and minor improvements in preprocessing can lead to measurable gains in inference performance.

Another critical aspect of image processing is its role in reducing the dimensionality and complexity of data. By emphasizing informative features such as edges, textures, and contours, image-processing methods support feature extraction techniques and lighten the computational load on machine-learning models. These mechanisms are essential in large-scale systems such as autonomous vehicles, medical imaging platforms, and industrial inspection systems, where performance and reliability are critical [18].

Because preprocessing tasks are applied to every pixel of an image, they account for a significant portion of the total computational cost in modern vision pipelines. This makes the development of efficient and parallelizable processing algorithms a central challenge. With the increasing prevalence of high-resolution images and real-time requirements, strategies to accelerate preprocessing, such as those based on OpenMP, have become indispensable [19].

To illustrate the practical impact of preprocessing and early-stage filtering, Fig. 3 presents an example in which computer-vision methods are applied to detect potholes on road surfaces. This type of application depends heavily on well-designed preprocessing steps, such as noise reduction, contrast normalization, and edge enhancement, to ensure that the subsequent detection algorithms can reliably isolate structural irregularities. Without these initial corrections, variations in lighting and background texture can mask relevant features or lead to false detections, reinforcing the importance of efficient and accurate preprocessing pipelines in real-world scenarios [7].

2.1 Pointwise Operation

Pointwise operations are among the most straightforward yet most versatile techniques in image processing. They operate independently on each pixel without considering spatial relationships. This independence provides both conceptual clarity and computational

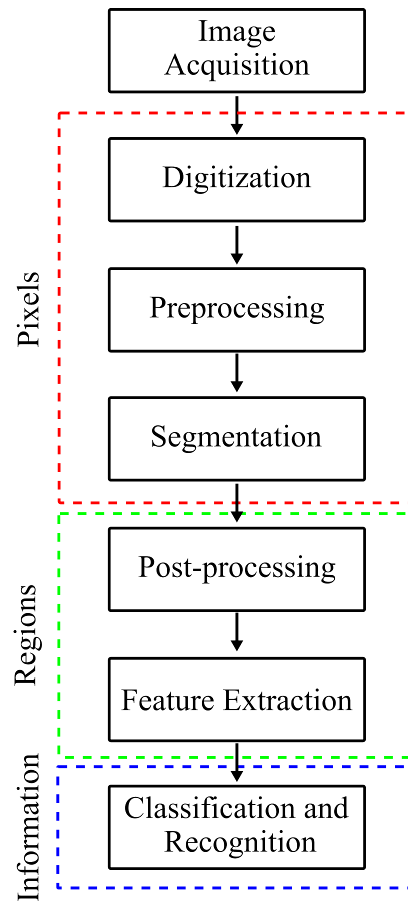

Figure 2: Typical workflow of a computer-vision system.

Figure 3: Example of computer-vision techniques applied to pothole detection on road surfaces [7]

efficiency, making pointwise transformations a natural starting point for any image-processing pipeline. Examples include brightness adjustment, gamma correction, thresholding, normalization, and various mathematical transformations applied directly to each pixel's intensity value [20].

Fig. 4 illustrates two forms of pointwise processing: (a) pixel-wise operations, where each pixel is transformed independently, and (b) pointwise convolutions (1x1 convolutions), which operate

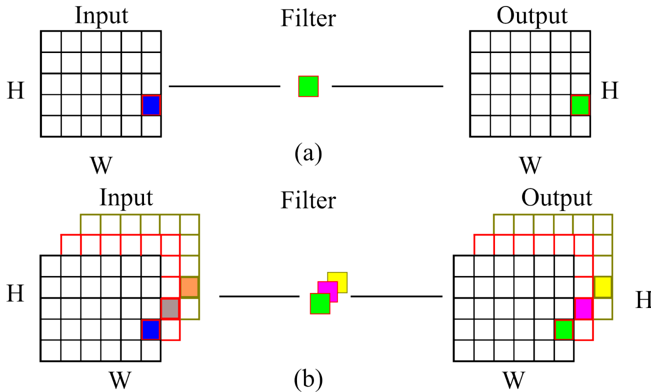

Figure 4: Pointwise operations: (a) pixel-wise transformation and (b) pointwise (1x1) convolution across feature channels.

on individual pixel locations while combining information across feature channels.

Mathematically, a pointwise operation can be expressed as:

$$O(x, y) = f\big(I(x, y)\big). \tag{1}$$

Equation 1 defines a pointwise operation in which each output pixel $O(x, y)$ is obtained by applying a scalar function $f(\cdot)$ to its corresponding input pixel value $I(x, y)$. A key characteristic of this formulation is the complete independence between pixels: the value of $O(x, y)$ depends solely on $I(x, y)$ and is unaffected by $I(u, v)$ for any $(u, v) \neq (x, y)$. This contrasts strongly with convolution-based filtering, where each output pixel depends on a localized neighborhood of pixels.

Because pointwise operations do not rely on spatial neighborhoods, they exhibit perfect data parallelism. Each pixel can be processed independently and simultaneously, making these operations highly suitable for multicore CPUs, GPUs, and shared-memory programming models such as OpenMP. This property enables near-linear scaling with the number of processing threads, particularly for high-resolution images where millions of pixels can be transformed in parallel [21].

The computational efficiency of pointwise operations also contributes to their widespread use in embedded systems and mobile devices. These platforms often operate under strict energy and resource constraints, yet still require fast and reliable preprocessing. Pointwise operations strike an excellent balance between simplicity, speed, and robustness, making them suitable for deployment across a broad range of hardware architectures [21].

2.2 Convolution Operation

Convolution represents the core of many spatial filtering techniques and serves as one of the most fundamental operations in image processing. Unlike pointwise transformations, convolution examines the spatial relationships between pixels (Fig. 5).

It applies a kernel or mask over a local neighborhood, generating an output that reflects the structural characteristics of the nearby region [22]. This makes convolution the basis for edge detection, smoothing, sharpening, texture extraction, and frequency analysis.

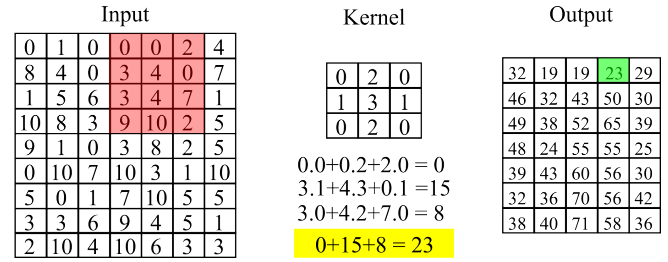

Figure 5: 2D convolution operation, illustrating the element-wise multiplication between the input window and the convolution kernel in a typical image-processing workflow.

Mathematically, convolution for an image $I(x, y)$ and a kernel $K(i, j)$ is expressed as:

$$O(x, y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i, j)\, I(x - i, y - j), \tag{2}$$

where $(2a + 1) \times (2b + 1)$ defines the kernel dimensions. This operation must be computed for every pixel in the image, resulting in an algorithm whose complexity grows proportionally with both the image size and the kernel size. In practical applications, a single convolution may require millions of multiplications and additions [23].

Convolution is indispensable because it captures local patterns that are not evident from global operations. Smoothing filters, such as the Gaussian filter, reduce noise while preserving important features. Edge detectors such as Sobel and Prewitt identify significant transitions in intensity. Sharpening filters enhance delicate structures and improve local contrast. More complex kernels, such as Gabor filters, enable frequency and orientation analysis, supporting texture classification and biometric applications [24].

However, convolution is computationally intensive, especially for high-resolution images or large kernels [25]. The repetitive and localized nature of the computations makes the operation highly parallelizable; however, an efficient parallel implementation requires attention to memory locality, data layout, and cache behavior. When properly parallelized using OpenMP, convolution achieves substantial speedups, making it suitable for real-time imaging applications that demand fast responses.

2.3 Image Filters

Image filtering plays a central role in digital image processing, offering mechanisms to reduce noise, enhance perceptual structures, or extract meaningful information from visual data. Although filtering methods are often grouped under the same umbrella, their goals can differ substantially. Denoising techniques focus on recovering the original appearance of an image by suppressing unwanted disturbances, while edge-detection and segmentation filters intentionally highlight transitions, contours, or semantic regions. This contrast helps clarify why noise-removal filters emphasize detail preservation under distortion, whereas other filters are designed to accentuate structural boundaries [26].

A widely used reference for illustrating filtering behavior is the Lena image, which serves as a long-standing benchmark in image-processing research. When heavily corrupted by salt-and-pepper noise, such as the 80% degradation shown in Fig. 6, and different filters exhibit distinct abilities to restore visual quality. Classical approaches, such as the Median Filter (MF) and its extensions, including the Decision-Based Weighted Median Filter (DWMF) and the Boundary Discriminative Noise Filter (BPDF), rely on statistical ordering to suppress impulsive noise while preserving edges. More adaptive variants, such as the Switching Median Filter (SMF), Adaptive Median Filter (AMF), Noise-Adaptive Fuzzy Switching Median (NAFSM), Modified Decision-Based Unsymmetric Trimmed Median Filter (MDBUTMF), and Decision-Based Adaptive Median Filter (DAMF), adjust their behavior dynamically according to noise intensity, offering stronger robustness under severe corruption [27].

Figure 6: Filtering results for the Lena image corrupted by 80% salt-and-pepper noise: (a) free-noise image, (b) MF, (c) DWMF, (d) BPDF, (e) SMF, (f) AMF, (g) NAFSM, (h) MDBUTMF, (i) DAMF [27].

In practice, filters operate by applying specific algorithms to pixel intensities, often taking into account the spatial relationships between neighboring pixels. Linear filters, such as Gaussian smoothing or Laplacian sharpening, compute weighted combinations of surrounding values through convolution, making them mathematically predictable and computationally efficient. Nonlinear filters, including median and bilateral filters, rely on more sophisticated selection rules that better preserve structural details at the cost of increased computational effort [28]. Due to this trade-off, the choice of filtering method must strike a balance between noise characteristics, desired visual structures, and processing constraints.

Filtering also plays a strategic role in modern machine learning and real-time imaging pipelines. Although convolutional neural networks learn their own filtering kernels, well-designed preprocessing filters can improve convergence, reduce sensitivity to noise, and enhance the quality of learned representations, benefits that are especially valuable in areas such as medical diagnostics, remote sensing, and surveillance. At the same time, many practical systems, including industrial inspection platforms, autonomous navigation modules, and adaptive surveillance networks, must process large volumes of visual data under stringent timing requirements. In these scenarios, efficient and parallelizable filtering algorithms, particularly those accelerated with OpenMP, enable both classical and adaptive methods to scale effectively across multicore CPUs, ensuring reliable real-time performance even when operating on high-resolution images [29].

2.4 Single-Core and Multi-Core Architectures

The characteristics of the underlying hardware architecture significantly influence the performance of image-processing pipelines. Early computer-vision systems relied on single-core processors, which execute instructions sequentially. While adequate for low-resolution images and simple operations, single-core architectures struggle when dealing with high-resolution images, multiple frames per second, or complex filtering routines. As a result, sequential execution often becomes a bottleneck that limits system responsiveness [30].

Modern multicore processors address these limitations by allowing multiple threads to execute simultaneously (Fig. 7). This architectural shift represents a paradigm change in software development, particularly for applications like image processing that benefit from data-level parallelism. By distributing independent computations across cores, multicore systems accelerate algorithms and reduce latency in real-time applications [31].

However, achieving optimal performance on multicore architectures is not trivial. Developers must carefully consider thread scheduling, cache coherence, memory locality, and load balancing. For instance, convolution involves repeated access to neighboring pixels, which can lead to cache misses if memory is not accessed efficiently. Techniques such as tiling, blocking, and cache-aware scheduling can mitigate these issues and improve performance [33].

When these optimizations are applied, multicore architectures enable significant acceleration in image-processing tasks. This capability is essential in domains like robotics, autonomous vehicles, and embedded systems, where decisions must be made within tight time constraints. Consequently, multicore processing has become the standard in modern computer-vision solutions, providing both scalability and efficiency.

2.5 Parallelization with OpenMP

OpenMP provides a simple yet powerful approach for introducing parallelism into image-processing algorithms running on shared-memory systems. Instead of rewriting entire applications, developers can add parallel behavior through compiler directives that extend C, C++, and Fortran in a nonintrusive way. This design makes OpenMP particularly appealing for accelerating legacy codebase and computational kernels that naturally expose data-level

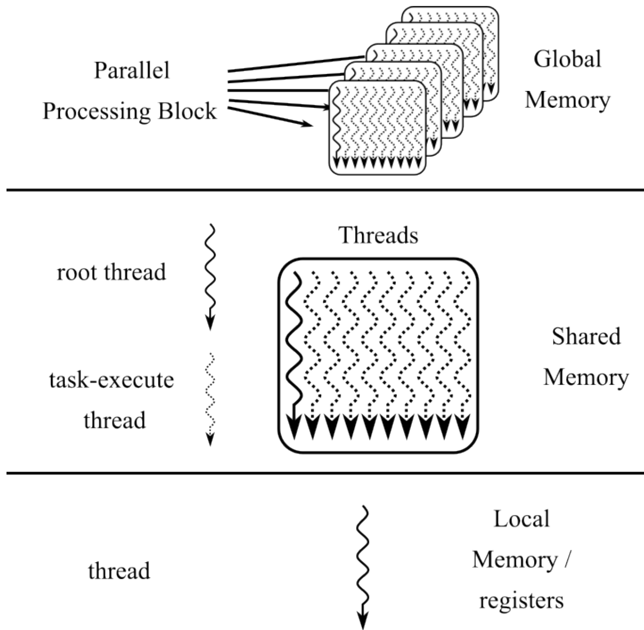

Figure 7: Overview of parallel processing: global memory, shared memory with thread blocks, and local thread registers [32].

parallelism, such as convolutions, filters, and pointwise operations, without requiring specialized hardware or complex programming models. By relying on lightweight threads and a well-structured runtime environment, OpenMP enables developers to scale workloads across multiple CPU cores while maintaining readable and maintainable code [34].

OpenMP offers a directive-based model that integrates parallelism with minimal syntactic overhead. The most widely used construct,

```
#pragma omp parallel for
```

automatically distributes loop iterations across available threads, making it especially suitable for image-processing tasks that involve nested loops over pixel coordinates. By parallelizing these loops, computationally intensive routines such as smoothing, edge-preserving filtering, and feature extraction can take full advantage of multicore CPUs. This combination of simplicity, portability, and efficiency makes OpenMP one of the most practical tools for accelerating image-processing pipelines on general-purpose processors [34].

Fig. 8 shows how pointer-based tasks can be dispatched in parallel using OpenMP, allowing independent memory regions to be processed simultaneously [35]. In image processing, this approach naturally extends to dividing an image into rows, columns, or blocks, with each partition assigned to a separate thread. By ensuring that each thread operates on its own isolated region, the need for synchronization is greatly reduced, improving scalability on multi-core systems. Still, effective parallelization requires careful attention to memory-access behavior: poor data locality or overlapping cache lines can introduce significant overhead. Using contiguous buffers, aligning data structures, and avoiding false sharing are therefore essential practices for achieving high throughput and fully leveraging the available parallelism [36].

```
1   #pragma omp parallel
2   {
3       /* a single thread traverses the list */
4       #pragma omp single
5       {
6           p = listhead;
7           while(p) {
8               /* create a task for each element */
9               #pragma omp task
10              process(p);
11              p = next(p);
12          }
13      }
14  }
```

Figure 8: 9. Parallel pointer chasing using task [35].

2.6 Performance Metrics

Performance evaluation in image processing requires metrics that quantify not only execution time but also scalability, efficiency, and overall effectiveness. The most fundamental metric is the execution time of sequential and parallel implementations, measured under controlled conditions to ensure consistency [37]. These measurements form the basis for calculating speedup:

$$S = \frac{T_1}{T_\mathrm{p}} \tag{3}$$

where T_1 denotes the execution time using a single thread, and T_p represents the execution time when p threads are employed. Larger values of S indicate greater acceleration and improved parallel performance.

Another important metric is parallel efficiency:

$$E = \frac{S}{N}, \tag{4}$$

where N represents the number of threads, efficiency reveals how well computational resources are utilized. Values close to 1 indicate excellent scaling, while lower values suggest that overhead, imbalance, or contention are degrading performance.

Scalability metrics evaluate how performance changes as the image size increases or the number of threads increases. Large images generally offer better scaling because they contain more work per thread, reducing the relative impact of overhead. Conversely, small images may exhibit limited speedup due to thread-management costs dominating computation. In addition to classical metrics, modern pipelines also evaluate end-to-end throughput, measuring the number of images processed per second when integrating preprocessing and inference. This holistic evaluation offers a realistic perspective on how OpenMP acceleration enhances overall system performance, bridging the gap between algorithmic optimization and practical deployment [37].

3 Methodology and Application

3.1 Computational Environment

The experiments were conducted on a workstation equipped with an Apple M2 processor featuring 8 physical CPU cores and 8 GB of RAM, running the Darwin operating system. Python was used exclusively for experiment orchestration, data loading, and result visualization, while the computationally intensive image-processing routines were implemented in C++ and parallelized using the OpenMP programming model. This hybrid approach combines the flexibility of Python with the performance of native multicore execution, enabling efficient evaluation of parallel image-processing workloads.

3.2 Image Dataset

The dataset employed in this study was obtained from the Kaggle platform [38]. It comprises approximately 50 images distributed across 16 bird species in high resolution.These images were selected to evaluate the performance of various image preprocessing techniques in enhancing the quality and features of bird images for subsequent analysis.

3.3 Performance Results

The performance evaluation of the implemented image preprocessing filters was conducted by measuring the execution time for each filter across multiple runs. The results are presented in Figures 9 to 13, showcasing the performance metrics for each filter. Additionally, Figures 15 and 14 provide a comparative analysis of efficiency and speedup achieved through the optimizations implemented.

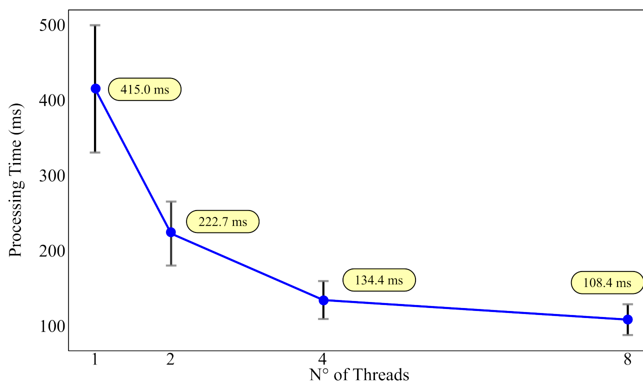

Figure 9: Performance analysis of the Blur filter.

Fig. 9 shows how the Blur filter responds when more threads are added during preprocessing. Using just one thread, the process takes 415.0 ms, but this time drops to 108.4 ms when eight threads are used. The improvement is quite noticeable: the task becomes faster as the work is shared among the cores, showing how naturally this filter benefits from parallel execution.

In Fig. 10, the CLAHE filter shows a similar pattern, starting at 1668.8 ms with one thread and falling to 244.9 ms with eight, the filter clearly becomes more responsive as more threads are introduced. Since CLAHE works by dividing the image into smaller regions, distributing these regions among several threads helps the algorithm progress much more smoothly.

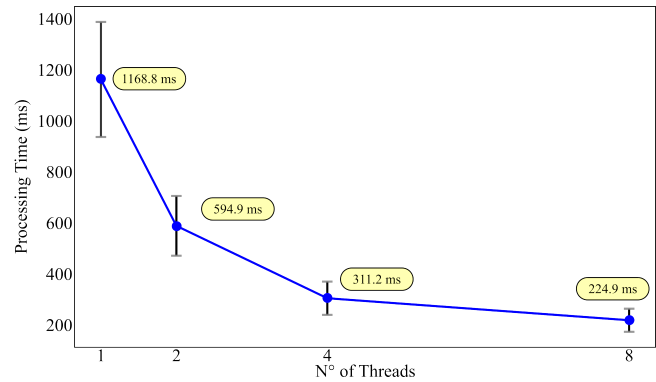

Figure 10: Performance analysis of the CLAHE filter.

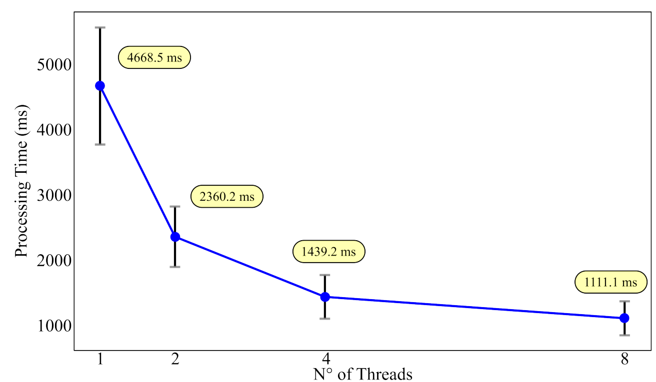

Figure 11: Performance analysis of the Denoise filter.

Fig. 11 reflects the behavior of the Denoise filter, which is the most time-consuming among those evaluated. Even so, it experiences one of the most expressive improvements: a drop from 4668.5 ms to 1111.1 ms when moving from one to eight threads. This reduction shows how parallelism helps lighten the workload of a filter that, by nature, requires many operations before reaching a clean result. As a result, the Denoise operation maintains good scalability, even as it approaches the hardware's parallel processing limits.

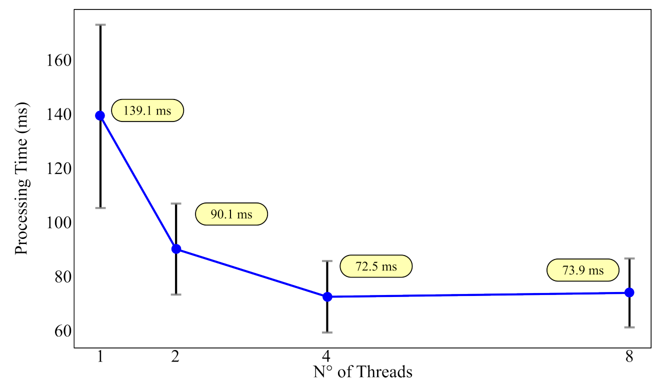

Figure 12: Performance analysis of the Edge filter.

In Fig. 12, the Edge filter already starts with a relatively short execution time (139.1 ms). Because of that, the improvement when adding more threads is more modest, reaching 72.5 ms with eight threads. Still, even a small gain is meaningful, especially considering that lighter filters tend to be more sensitive to the internal overhead of managing multiple threads.

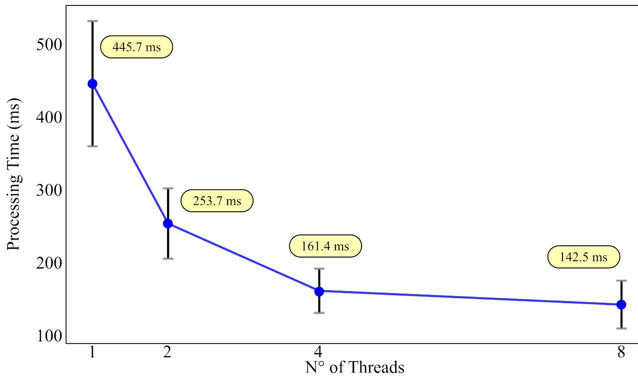

Figure 13: Performance analysis of the Sharpen filter.

Lastly, Fig. 13 shows the Sharpen filter, which moves from 445.7 ms to 142.5 ms as the thread count increases. The improvement is steady and reflects how well this filter adapts to parallel work. Each part of the image can be enhanced simultaneously, making the whole process feel faster and more efficient. This consistent reduction in execution time also indicates that the filter distributes its computational workload evenly across threads.

Across all preprocessing methods evaluated, the execution time exhibits a clear exponential decreasing trend as the number of threads increases. This behavior reflects the natural scalability of these operations under parallel execution, where distributing the workload among multiple cores leads to rapid initial performance gains before gradually approaching a saturation point.

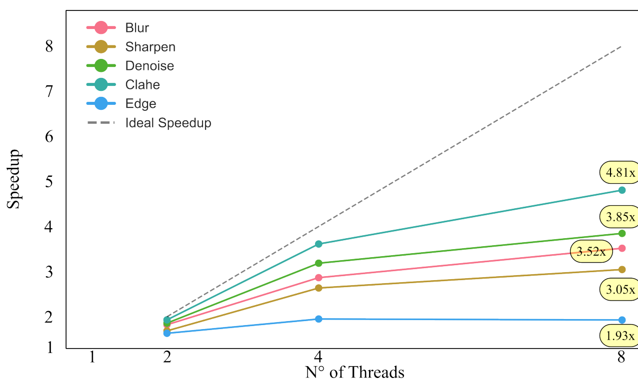

Figure 14: Comparison of speedup across different image preprocessing filters.

Fig. 14 shows the speedup achieved by each filter as the number of threads increases. The CLAHE and Denoise filters exhibit the most significant speedups, reaching up to approximately 4.81x and 3.85x respectively with eight threads. This indicates that these filters benefit greatly from parallelization due to their computational intensity and the ability to divide their workload effectively across multiple threads.

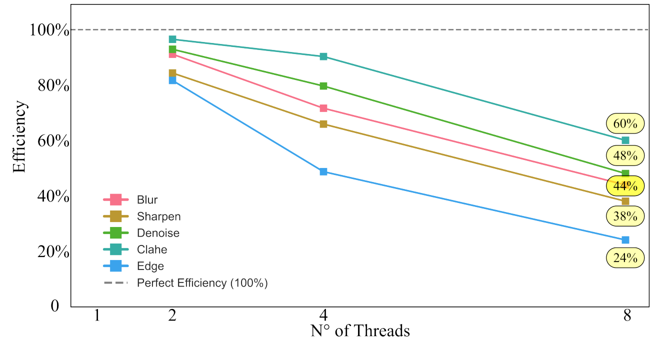

Figure 15: Comparison of Efficiency

Fig. 15 shows a consistent percentage decrease in parallel processing efficiency as the number of threads increases for all evaluated image processing operations. This behavior can be attributed to synchronization overheads, memory contention, and reduced cache efficiency. At lower thread counts, all methods achieve high efficiency, indicating effective parallelization. However, as the level of concurrency increases, the impact of parallel overheads becomes more pronounced, leading to divergent scalability behavior among the algorithms.

CLAHE and denoising exhibit superior scalability, maintaining approximately 60% and 48% efficiency at eight threads, respectively, due to their higher computational intensity and improved data locality. In contrast, edge detection experiences a significant efficiency drop, reaching only 24% at eight threads, suggesting memory-bound behavior with limited computational workload per thread. Blur and sharpen operations show intermediate performance, reflecting moderate scalability constrained by increasing parallel overheads.

4 Conclusion

This work evaluated the impact of OpenMP-based parallelization on the performance of image preprocessing and convolutional filters executed on multicore CPUs. The experimental results demonstrated that properly parallelized implementations achieve significant reductions in execution time, particularly for high-resolution images and computationally intensive filters, where the workload per thread is sufficient to amortize parallelization overhead.

The analysis showed that both image resolution and filter complexity play a central role in determining achievable speedup and efficiency. While lightweight operations exhibit limited scalability due to thread-management overhead, more demanding filters benefit substantially from multicore execution. These findings reinforce the practicality of OpenMP as an effective solution for accelerating image preprocessing on general-purpose CPUs.

Although this study focused on isolated preprocessing stages, the observed performance gains indicate the potential feasibility of incorporating OpenMP-accelerated filters as preprocessing components within larger computer vision and AI workflows. Future work

may investigate full end-to-end pipelines, including integration with deep learning models, real-time video streams, and heterogeneous CPU–GPU execution, as well as the impact of parallelization on energy efficiency.

References

[1] Mohd Javaid, Abid Haleem, Ravi Pratap Singh, Shanay Rab, and Rajiv Suman. Exploring impact and features of machine vision for progressive industry 4.0 culture. *Sensors International*, 3:100132, 2022. ISSN 2666-3511. doi: https://doi.org/10.1016/j.sintl.2021.100132.

[2] Wael A. Farag and Mohamed Fayed. Advancing vehicle detection for autonomous driving: integrating computer vision and machine learning techniques for real-world deployment. *Journal of Control and Decision*, 0(0):1–18, 2025. doi: https://doi.org/10.1080/23307706.2025.2469893.

[3] Charleen, Cheryl Angelica, Hendrik Purnama, and Fredy Purnomo. Impact of computer vision with deep learning approach in medical imaging diagnosis. In *2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI)*, volume 1, pages 37–41, 2021. doi: https://doi.org/10.1109/ICCSAI53272.2021.9609708.

[4] V.G. Dhanya, A. Subeesh, N.L. Kushwaha, Dinesh Kumar Vishwakarma, T. Nagesh Kumar, G. Ritika, and A.N. Singh. Deep learning based computer vision approaches for smart agricultural applications. *Artificial Intelligence in Agriculture*, 6:211–229, 2022. ISSN 2589-7217. doi: https://doi.org/10.1016/j.aiia.2022.09.007.

[5] Deep Kothadiya, Aayushi Chaudhari, Ruchita Macwan, Krishna Patel, and Chintan Bhatt. The convergence of deep learning and computer vision: Smart city applications and research challenges. In *Proceedings of the 3rd International Conference on Integrated Intelligent Computing Communication & Security (ICIIC 2021)*, pages 14–22. Atlantis Press, 2021. ISBN 978-94-6239-428-5. doi: https://doi.org/10.2991/ahis.k.210913.003.

[6] Muhammad Hussain. Sustainable machine vision for industry 4.0: A comprehensive review of convolutional neural networks and hardware accelerators in computer vision. *AI*, 5(3):1324–1356, 2024. ISSN 2673-2688. doi: https://doi.org/10.3390/ai5030064.

[7] Luiz Fernando Mello, Evandro L Viapiana, and Luiz Fernando M Arruda. Análise comparativa de algoritmos de detecção de objetos para reconhecimento de buracos em vias e estradas. *Anais do Computer on the Beach*, 15:118–125, 2024. doi: https://doi.org/10.14210/cotb.v15.p118-125.

[8] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9992–10002, 2021. doi: https://doi.org/10.1109/ICCV48922.2021.00986.

[9] Jiasi Chen and Xukan Ran. Deep learning with edge computing: A review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019. doi: 10.1109/JPROC.2019.2921977.

[10] Min Li, Zhijie Zhang, Liping Lei, Xiaofan Wang, and Xudong Guo. Agricultural greenhouses detection in high-resolution satellite images based on convolutional neural networks: Comparison of faster r-cnn, yolo v3 and ssd. *Sensors*, 20(17):4938, 2020.

[11] Jiss Kuruvilla, Dhanya Sukumaran, Anjali Sankar, and Siji P Joy. A review on image processing and image segmentation. In *2016 International Conference on Data Mining and Advanced Computing (SAPIENCE)*, pages 198–203, 2016. doi: https://doi.org/10.1109/SAPIENCE.2016.7684170.

[12] Yiming Gan, Yuxian Qiu, Lele Chen, Jingwen Leng, and Yuhao Zhu. Low-latency proactive continuous vision. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, pages 329–342, 2020.

[13] Christian Terboven, Dieter an Mey, and Samuel Sarholz. Openmp on multicore architectures. In Barbara Chapman, Weiming Zheng, Guang R. Gao, Mitsuhisa Sato, Eduard Ayguadé, and Dongsheng Wang, editors, *A Practical Programming Model for the Multi-Core Era*, pages 54–64, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-69303-1.

[14] Mekhriddin Rakhimov, Jamshid Elov, Utkir Khamdamov, Shavkatjon Aminov, and Shakhzod Javliev. Parallel implementation of real-time object detection using openmp. In *2021 International Conference on Information Science and Communications Technologies (ICISCT)*, pages 1–4, 2021. doi: https://doi.org/10.1109/ICISCT52966.2021.9670146.

[15] Minyu Cui and Miquel Pericàs. Evaluation and mitigation of performance variability of openmp applications on modern multicore systems. In *2025 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1271–1273, 2025. doi: https://doi.org/10.1109/IPDPSW66978.2025.00209.

[16] Asha Rani and Rosepreet Kaur Bhogal. Efficient real-world image denoising using multi-scale gaussian pyramids. *Scientific Reports*, 15(1):40086, 2025. doi: https://doi.org/10.1038/s41598-025-23942-8.

[17] Rafael C Gonzalez. *Digital image processing*. Pearson education india, 2009.

[18] Mingxing Tan, Ruoming Pang, and Quoc V. Le. EfficientDet: Scalable and Efficient Object Detection . In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10778–10787, Los Alamitos, CA, USA, June 2020. IEEE Computer Society. doi: https://doi.org/10.1109/CVPR42600.2020.01079.

[19] Semra Aydin, Refik Samet, and Omer Faruk Bay. Real-time parallel image processing applications on multicore cpus with openmp and gpgpu with cuda. *J. Supercomput.*, 74(6):2255–2275, June 2018. ISSN 0920-8542. doi: 10.1007/s11227-017-2168-6.

[20] Zheng Liu, Meng Hao, Weizhe Zhang, Gangzhao Lu, Xueyang Tian, Siyu Yang, Mingdong Xie, Jie Dai, Chenyu Yuan, Desheng Wang, et al. Optimizing depthwise separable convolution on dcu. *CCF Transactions on High Performance Computing*, 6(6):646–664, 2024.

[21] David B. Kirk and Wen-mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., 3rd edition, 2016. ISBN 9780128119877. doi: 10.5555/1841511.

[22] Jianxin Wu. Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, 5(23):495, 2017.

[23] Arohan Ajit, Koustav Acharya, and Abhishek Samanta. A review of convolutional neural networks. In *2020 international conference on emerging trends in information technology and engineering (ic-ETITE)*, pages 1–5. IEEE, 2020.

[24] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 33(12):6999–7019, 2021.

[25] Xu Kang, Bin Song, and Fengyao Sun. A deep similarity metric method based on incomplete data for traffic anomaly detection in iot. *Applied Sciences*, 9(1):135, 2019.

[26] Leonid P Yaroslavsky. *Digital picture processing: an introduction*, volume 9. Springer Science & Business Media, 2012.

[27] Achmad Abdurrazzaq, Ahmad Kadri Junoh, Wan Zuki Azman Wan Muhamad, Zainab Yahya, and Ismail Mohd. An overview of multi-filters for eliminating impulse noise for digital images. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 18(1):385–393, 2020.

[28] Antoni Buades, Bartomeu Coll, and J-M Morel. A non-local algorithm for image denoising. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 2, pages 60–65. Ieee, 2005.

[29] Manju Mathews and Jisha P Abraham. Automatic code parallelization with openmp task constructs. In *2016 International Conference on Information Science (ICIS)*, pages 233–238, 2016. doi: 10.1109/INFOSCI.2016.7845333.

[30] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[31] Byeongrok Min, Junhyeong Ryu, Yongseok Son, Sungrae Cho, and Jeongyeup Paek. Parallel processing of large-scale point cloud data for connected and automated mobility–a survey. In *2024 15th International Conference on Information and Communication Technology Convergence (ICTC)*, pages 469–473. IEEE, 2024.

[32] T Kalaiselvi, P Sriramakrishnan, and K Somasundaram. Performance analysis of morphological operations in cpu and gpu for accelerating digital image applications. *International Journal of Computationa Science and Information Technology (IJCSITY)*, pages 15–27, 2016.

[33] Hyeonjin Kim and William J. Song. Las: Locality-aware scheduling for gemm-accelerated convolutions in gpus. *IEEE Transactions on Parallel and Distributed Systems*, 34(5):1479–1494, 2023. doi: 10.1109/TPDS.2023.3247808.

[34] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, 1998. doi: 10.1109/99.660313.

[35] Eduard Ayguade, Nawal Copty, Alejandro Duran, Jay Hoeflinger, Yuan Lin, Federico Massaioli, Xavier Teruel, Priya Unnikrishnan, and Guansong Zhang. The design of openmp tasks. *IEEE Transactions on Parallel and Distributed Systems*, 20(3):404–418, 2009. doi: 10.1109/TPDS.2008.105.

[36] Seonmyeong Bak, Colleen Bertoni, Swen Boehm, Reuben Budiardja, Barbara M. Chapman, Johannes Doerfert, Markus Eisenbach, Hal Finkel, Oscar Hernandez, Joseph Huber, Shintaro Iwasaki, Vivek Kale, Paul R.C. Kent, JaeHyuk Kwack, Meifeng Lin, Piotr Luszczek, Ye Luo, Buu Pham, Swaroop Pophale, Kiran Ravikumar, Vivek Sarkar, Thomas Scogland, Shilei Tian, and P.K. Yeung. Openmp application experiences: Porting to accelerated nodes. *Parallel Computing*, 109:102856, 2022. ISSN 0167-8191. doi: https://doi.org/10.1016/j.parco.2021.102856.

[37] Ruud Van der Pas, Eric Stotzer, and Christian Terboven. *Using OpenMP-The Next Step: Affinity, Accelerators, Tasking, and SIMD*. MIT press, 2017.

[38] akash2907. Bird species classification. Kaggle, 2024. URL https://www.kaggle.com/datasets/akash2907/bird-species-classification. Data set.