DOI 10.15276/imms.v15.no1.15
UDC 004.932+004.4’2+004.272.2

Informatics and Mathematical Methods in Simulation
Vol.15 (2025), No. 1, pp. 15-23

ACCELERATION OF IMAGE PROCESSING ALGORITHMS USING SIMD TECHNOLOGY

O.O. Zhulkovskyi[1], H.Ya. Vokhmianin[1],
I.I. Zhulkovska[2], Yu.V. Ulianovska[2], E.A. Riabovolenko[2]

[1]Dniprovsky State Technical University
Dniprobudivska str., 2, Kamianske city, 51918, Ukraine
[2]University of Customs and Finance
2/4, Volodymyr Vernadskyi str., Dnipro, 49000, Ukraine
Email: olalzh@ukr.net

A rational choice of computing platform and software optimization that considers the specifics of processor architecture can significantly reduce the time of complex computations, improve overall system performance, and ensure efficient scalability when processing large amounts of data. This study addresses the problem of accelerating computational image processing algorithms, specifically the Gaussian smoothing algorithm, using SIMD data-level parallelism technology. The Gaussian smoothing algorithm, which is commonly used to reduce noise and remove small image details, is characterized by high computational complexity due to the need to perform numerous arithmetic operations on each pixel. In this regard, the optimization of such algorithms is a relevant task, and SIMD technology offers the potential for parallel data processing using extended processor instructions, thereby significantly enhancing computational performance. The aim of this study is to enhance the efficiency of the Gaussian smoothing algorithm through SIMD-based computation optimization. Two variants of the software implementation of the studied algorithm have been developed: one is scalar and the other has been optimized through the use of AVX-256 instructions. The optimized version employs computation vectorization, processing eight pixels simultaneously using 256-bit registers, which theoretically allows up to an eightfold speedup. Computational experiments were conducted using images with resolutions of 1920x1080 and 2560x1440 pixels, across various kernel radius and standard deviation values. The results demonstrated that implementing SIMD instructions resulted in a speed enhancement ranging from 6.9 to 7.3 times when compared to the scalar approach. When the kernel radius increased, the acceleration remained consistently high, confirming the approach’s effectiveness for more complex computations. It was confirmed that execution time is primarily influenced by the kernel radius, while the standard deviation has a lesser effect, since the radius defines the filter’s area of influence. The speedup achieved in the experiments is close to the theoretical maximum, demonstrating the advantages of the optimized implementation. Future research prospects include combining SIMD optimization with multithreaded processing and studying the potential of more powerful instruction sets such as AVX-512.
Keywords: image processing, Gaussian smoothing, SIMD, AVX-256, parallel data processing.

Introduction. In today’s world of high-performance computing, efficient utilization of PC hardware capabilities is pivotal for optimizing data processing speeds. This implies not only a rational choice of computing platform, but also the optimization of software with regard to the processor architecture's particular characteristics such as vector instructions (SIMD – Single Instruction Multiple Data), multi-core processing, cache memory, and so forth. The application of such approaches enables a significant reduction in the execution time of complex computations, increases overall system performance, and ensures efficient scalability when processing large amounts of data [1]. Processing large datasets, particularly images, remains one of the most computationally intensive tasks. A typical example of such a task is the Gaussian smoothing algorithm, which is widely used for noise reduction and removal of details in digital images by performing a large number of similar arithmetic operations on

each image pixel [2]. In the context of growing data volumes and increasing software performance requirements, the optimization of such algorithms is becoming particularly relevant.

SIMD technology provides the necessary tools for parallel data processing at the processor instruction level. The core principle of SIMD is the execution of a single operation simultaneously on multiple data elements, which significantly enhances computational efficiency [3].

Related works. Image processing, particularly in artificial intelligence and computer vision systems, demands faster and more efficient data processing, making hardware acceleration with specialized equipment essential, as general-purpose CPUs face performance limitations when processing high-resolution data in real-time [4].

Unlike scalar computations, where each processor instruction processes a single data element, SIMD instructions operate on data vectors, enabling a significant increase in computational performance for algorithms with a high degree of data-level parallelism. Architecturally, SIMD is implemented by introducing specialized vector registers and corresponding instruction sets into processors. In modern CPUs, the width of SIMD registers has evolved from 64 bits in early implementations to 128, 256, and 512 bits in the latest architectures, allowing for the simultaneous processing of 4, 8, or 16 single-precision floating-point elements (32-bit floats), respectively [5, 6]. This architectural feature provides a theoretical speedup of calculations proportional to the number of elements that fit within a SIMD register. Consequently, SIMD is advisable to use in algorithms with iterative blocks containing uniform arithmetic operations, where the computation of each subsequent element is independent of the previous one.

A modified algorithm for solving the classical problem of multiplying ultra-large square data matrices using SIMD technology demonstrates a speedup in the range of 2.53–4.78x compared to traditional data processing methods and is independent of the amount of processed data [7]. In graph algorithms, the use of SIMD increases efficiency on the central processor. The evaluation results show that on an 8-core machine, enabling SIMD in a naïve multi-core implementation provides an additional speedup of 7.48x, averaged over ten benchmarks and three input datasets [8]. The application of custom SIMD-oriented optimizations improves the basic SIMD implementation by 1.67x and outperforms the scalar version by 12.46x.

SIMD instructions can enhance prediction performance through compression/recovery (CR) by up to 25% and reduce dynamic power consumption by up to 43% on real, unmodified applications using predictive execution [9]. CR can also execute unmodified legacy code with short vector instructions (AVX-2) on newer architectures with wider vectors (AVX-512), achieving up to a 56% increase in performance.

In image processing workflows, SIMD is suitable for hardware acceleration using a vector processor matrix to improve the performance of deblurring techniques for CT and MRI images affected by artifacts [10], for lookup tables (LUT) aimed at efficient image transformation on x86/64 processors by processing complex mathematical functions [11], and in other algorithms such as Gaussian smoothing [2].

Gaussian blur is a fundamental operation in image processing based on the convolution of an image with a Gaussian kernel. The basic principle of the algorithm involves replacing the value of each pixel with a weighted sum of the values of its neighboring pixels, where the weights are defined by the Gaussian kernel [12]. When implementing Gaussian blur in software, the computational complexity of the algorithm is $O(r^2 \times n^2)$ for an image of size $n \times n$ and a kernel of size $r \times r$. In the context of large images or real-time applications, this level of complexity can lead to unacceptable delays, making algorithm optimization essential.

A significant acceleration can be achieved by splitting the two-dimensional convolution into two consecutive one-dimensional operations [13]. This fundamentally

reduces the complexity to $O(r \times n^2)$. The splitting principle is based on the property of the Gaussian function, which allows the two-dimensional filter to be represented as the product of two one-dimensional filters. To further accelerate the Gaussian blur algorithm, parallel computing can be applied at various levels: for example, thread-level parallelism using OpenMP [2] and instruction-level vectorization using SIMD [2, 14].

Parallelizing the modified 1D convolution across processing cores increases performance by approximately 1.3x and reduces power consumption by 6.9% and 3.2% for 1D and 2D convolutions, respectively [2]. In [14], to improve the performance of one-dimensional convolution operations, both data-level and thread-level parallelism were employed using parallel programming models such as intraprocedural programming, automatic compiler vectorization, and open multiprocessing. The experimental results demonstrated that the performance of the obtained implementations significantly surpassed that of other approaches. The performance of the multithreaded versions of all implementations was greatly improved compared to single-core implementations, achieving a speedup of 52.33x over the optimal scalar version.

Research Objective. The objective of the present study is to efficiently utilize SIMD technology to accelerate image processing algorithms, with a focus on Gaussian smoothing as a case study. The work is focused on analyzing the performance of the scalar implementation of the algorithm and its optimized version using SIMD instructions, as well as demonstrating how parallel data processing can improve computational efficiency in tasks with a high level of computational complexity.

Main Part. Gaussian smoothing is implemented via the convolution operation of an image with a two-dimensional kernel, the values of which are determined by the Gaussian function [15]:

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}},$$

where (x, y) – are the coordinates of a pixel relative to the center of the kernel; σ – is the standard deviation of the Gaussian distribution, which defines the degree of blurring.

In the software implementation of the algorithm, the two-dimensional kernel is decomposed into two one-dimensional filters to optimize computational efficiency:

$$G(x,y)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{\frac{-x^2}{2\sigma^2}}\times\frac{1}{\sqrt{2\pi\sigma^2}}e^{\frac{-y^2}{2\sigma^2}}$$

Thus, the smoothing operation is represented as a sequential application of horizontal and vertical one-dimensional filters:

$$I'(x,y)=\sum_i\sum_j I(x-i,y-j)\times G(i,j),$$

where I – is the input image, I' – is the result of the smoothing process.

The C++ code developed within the framework of this study implements the Gaussian blur algorithm in two versions: scalar and optimized using SIMD technology based on the AVX-256 instruction set, which is part of the x86-64 processor architecture extensions. The code has been designed to process images represented as two-dimensional arrays of pixels. The implementation comprises the generation of a Gaussian kernel, the application of convolution in two directions (horizontal and vertical), and the comparison of the performance of both methods. The process begins with the creation of a Gaussian kernel, where the kernel size specifies the number of elements, and the standard deviation σ controls the width of the Gaussian curve, thereby determining the degree of image blurring. The kernel

is formed based according to the mathematical formula for the Gaussian distribution. Following the calculation of the kernel values, a process of normalization is then applied.

Image processing is carried out in two stages. The first stage involves horizontal convolution. For each pixel at coordinates (x, y), a weighted sum of neighboring pixel values in the horizontal row is calculated. The result is stored in an intermediate buffer of the float data type. The second stage – vertical convolution – is performed in a similar manner, but on the intermediate buffer: for each pixel at coordinates (x, y), a weighted sum of pixel values in the vertical column is computed. The results are stored in an output array of the float data type, which represents the final blurred image. Edge pixel processing (accesses beyond array boundaries) at both stages is managed using a mirroring method, implemented via the std::max and std::min functions applied to the corresponding indices.

The optimized version of the algorithm is implemented using AVX-256 instructions from the <immintrin.h> header. This version also performs processing in two stages but accelerates computation by vectorizing, processing data in blocks of eight pixels at a time. The primary data type used for vector operations is __m256, which represents a 256-bit register and contains eight values of the float data type [16]. During the horizontal convolution stage, image rows are processed. For each row, the outer loop iterates over pixels in steps of 8, matching the size of the __m256 vector. Within the loop, data is loaded from the image array into the register using the _mm256_loadu_ps instruction. The core convolution operation is executed using the _mm256_fmadd_ps (fused multiply-add) instruction, which performs element-wise multiplication of the first two vectors and adds the result to a third, returning a new __m256 vector [16, 17]. Once convolution over all kernel elements is complete, the result is stored in an intermediate buffer using _mm256_storeu_ps. If the image width is not divisible by eight, the remaining pixels are processed sequentially using the scalar method described earlier. Vertical convolution in the SIMD version is carried out similarly, with adjustments for accessing columnar data.

The execution time of the algorithms is measured using std::chrono::high_resolution_clock from the <chrono> library [18].

In order to verify the accuracy of the results obtained from both versions, a comparative analysis is conducted between them. Within a loop iterating over all pixels, the absolute difference between corresponding float values is computed. If the difference exceeds a threshold of 10^{-5}, a discrepancy in the results is recorded.

Computational experiments were conducted using the following test environment: CPU Intel Core i7-12700H, 14 cores, 20 threads; RAM 32 GB (2 Goodram DDR4: 16 GB, 3200 MHz); Microsoft Visual Studio C++ 2022 IDE; Microsoft Windows 10 OS, <immintrin.h> library, SIMD AVX 256 bit, x64.

Results and Discussion. The experiments were conducted on a range of images with resolutions of 1920x1080 and 2560 x1440 pixels. Each image underwent ten experimental runs with varying blur radii and standard deviations. This experimental setup enables the investigation of the impact of the blur radius and standard deviation on computation time.

The results obtained from the experiments conducted with various configurations and image resolutions (Table 1 and Table 2) confirm the theoretical advantages of using SIMD technology. For an image with a resolution of 1920x1080 pixels and a kernel radius of $r = 5$, the scalar implementation may take approximately 275 ms, whereas the optimized version utilizing AVX instructions reduces this time to 40 ms, achieving a speedup of nearly 7x. For higher resolutions such as 2560x1440, performance gains are also substantial: the scalar version executes in approximately 480 ms, while the SIMD version completes in about 70 ms, indicating consistent acceleration. Minor deviations from the theoretical maximum can be attributed to overhead associated with data vectorization, result de-vectorization, and processing of edge cases, which cannot be fully vectorized.

An increase in the kernel radius up to 10 leads to an increase in computational complexity; however, the relative acceleration achieved through SIMD remains substantial. For an image with a resolution of 1920x1080, the scalar processing time increases to 475 ms, while the SIMD-optimized version maintains a stable execution time of approximately 68 ms. This scalability highlights the effectiveness of SIMD optimization for more complex computations.

Table 1.

Experimental results for an image with a resolution of 1920x1080 pixels

№	Kernel radius *r*	Standard deviation *σ*	Computation time, ms		Acceleration
			No-AVX	AVX-256	
1	5	1	275	40	6.88
2		2	271	39	6.95
3		3	279	40	6.98
4		4	279	40	6.98
5		5	280	40	7.00
6	10	1	470	68	6.91
7		2	475	68	6.99
8		3	472	68	6.94
9		4	473	68	6.96
10		5	475	68	6.99

Table 2.

Experimental results for an image with a resolution of 2560x1440 pixels

№	Kernel radius *r*	Standard deviation *σ*	Computation time, ms		Acceleration
			No-AVX	AVX-256	
1	5	1	480	69	6.96
2		2	474	68	6.97
3		3	483	70	6.90
4		4	476	69	6.90
5		5	489	70	6.99
6	10	1	837	115	7.28
7		2	837	114	7.34
8		3	841	115	7.31
9		4	833	115	7.24
10		5	839	115	7.30

The obtained results demonstrate that the computation time is influenced by the kernel radius *r*, whereas increasing the standard deviation *σ* results in relatively stable execution time (Fig. 1).

Figures 2 and 3 present the outcomes of image processing using the Gaussian smoothing algorithm with various blurring parameters. As illustrated in Fig. 2, increasing the standard deviation *σ* while maintaining the same radius *r* yields a noticeable effect, but it is less significant compared to increasing the radius itself. In Fig. 3, more blurred images are produced due to the increased radius. It can thus be concluded that the radius defines the effective area of influence of the filter. It determines how many pixels surrounding each processed pixel are taken into account for the blurring operation. When the radius is small, even a large *σ* does not result in significant blurring, as pixels outside the radius are not included in the computation.

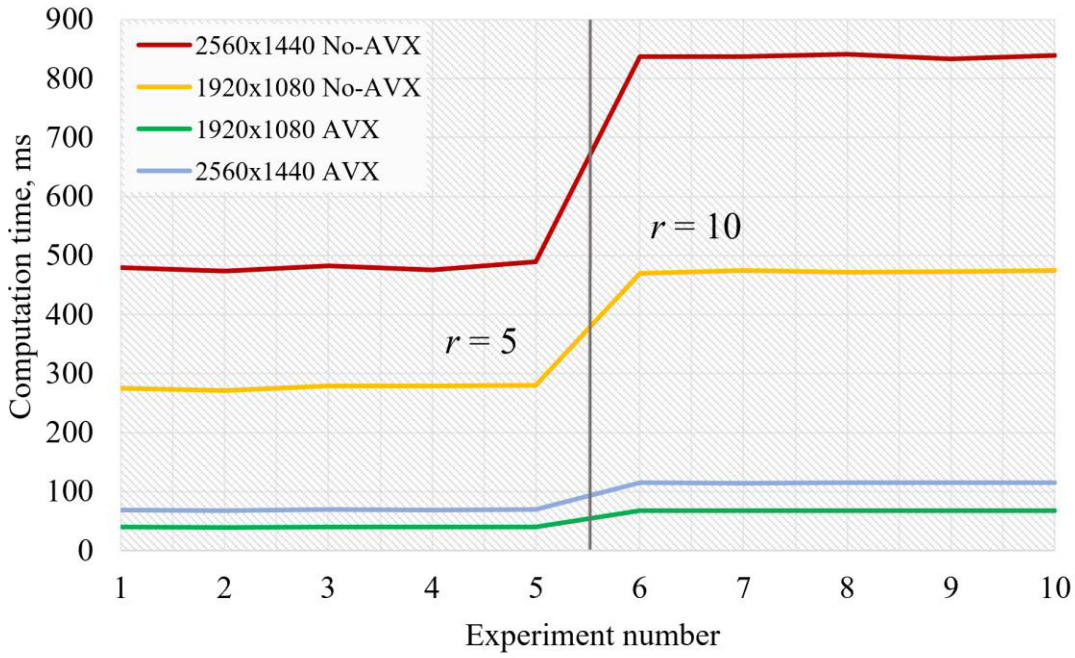

Fig. 1. Execution time results of computational algorithms for images of different resolutions

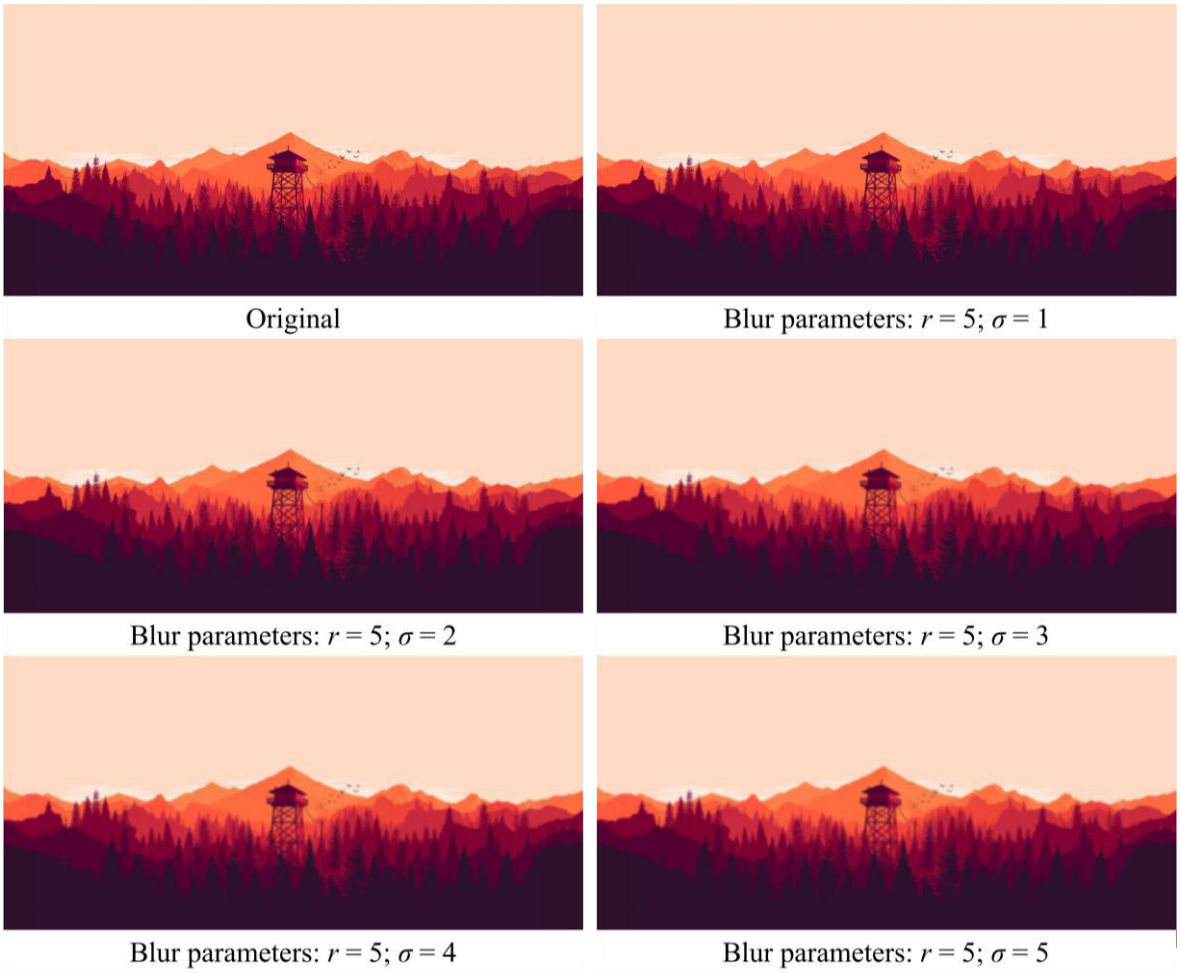

Fig. 2. Image processing results with kernel radius $r = 5$

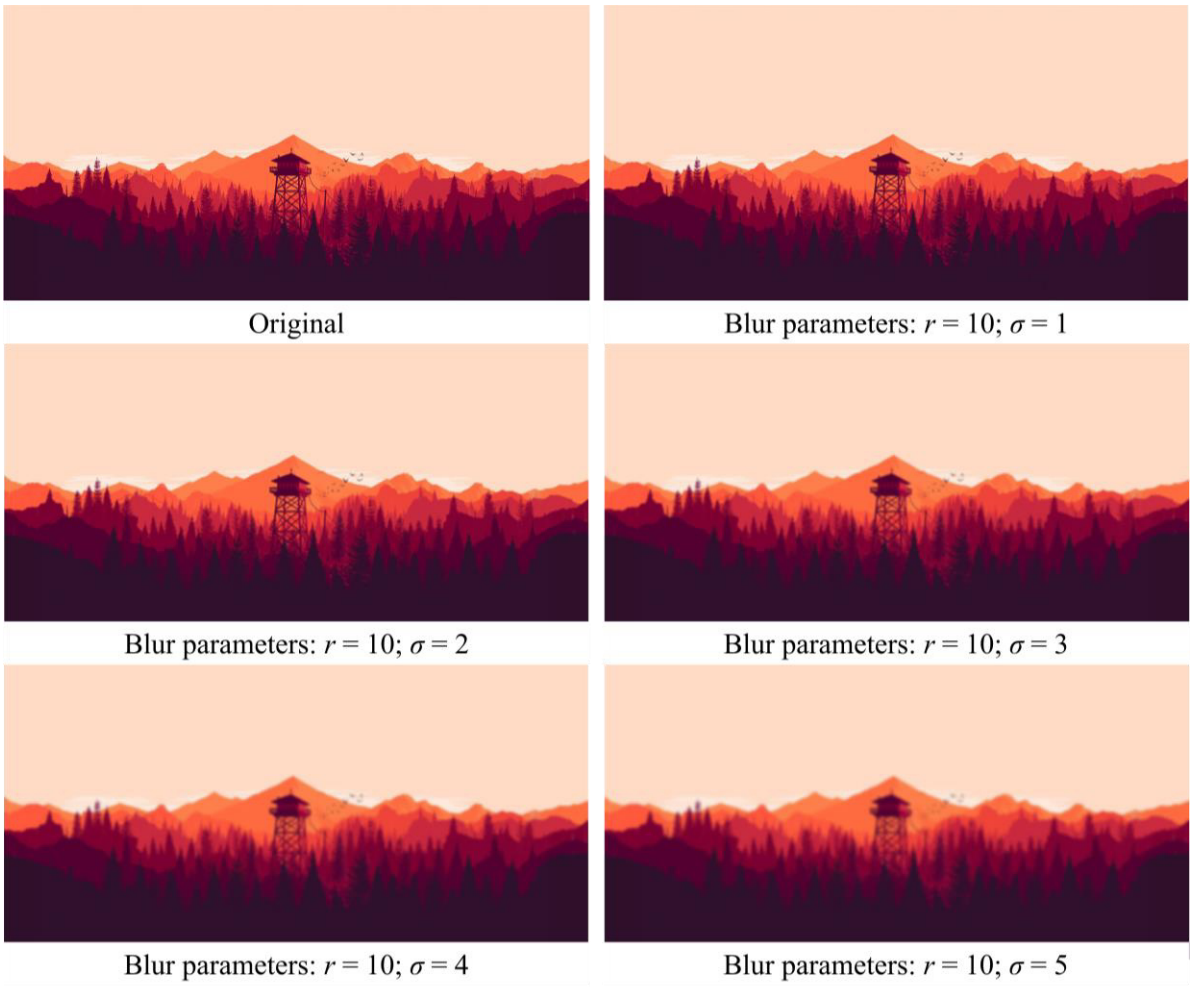

Fig. 3. Image processing results with kernel radius $r = 10$

Conclusions. The conducted research demonstrates the significant potential of utilising SIMD instructions to optimize image processing algorithms, particularly those involving Gaussian smoothing. The achieved speedup of 6.9–7.3x while maintaining identical output quality confirms the effectiveness of this approach for practical applications. Theoretically, the maximum increase in processing speed when using AVX-256 instructions for single-precision data type operations is 8x, since a single register can accommodate eight values of the float data type. The actual speedup obtained is close to the theoretical maximum, which indicates the effectiveness of the implementation. The slight deviation from the theoretical maximum can be attributed to overheads of data vectorization, result devectorization, and the processing of edge cases that cannot be fully vectorized.

Future research directions include combining SIMD optimization with multithreaded processing to achieve additional acceleration on multi-core processors, as well as exploring the potential of other instruction sets, such as AVX-512.

References

1. Masciari E., Napolitano E.V. Sustainability and High Performance Computing. *Lecture Notes in Computer Science*. 2024. P. 237–242. URL: https://doi.org/10.1007/978-3-031-78093-6_21
2. Zin N. Oo. The Improvement of 1D Gaussian Blur Filter using AVX and OpenMP. *2022 22nd Int. Conf. Control, Automat. Syst. (ICCAS)*, Nov. 27–Dec. 1 2022. Jeju, 2022. P. 1493–1496. URL: https://doi.org/10.23919/iccas55662.2022.10003739
3. Kusswurm D. SIMD Fundamentals. *Modern Parallel Programming with C++ and Assembly Language*. 2022. P. 1–16. URL: https://doi.org/10.1007/978-1-4842-7918-2_1

4. Vasile C.-E., Ulmămei A.-A., Bîră C. Image Processing Hardware Acceleration—A Review of Operations Involved and Current Hardware Approaches. 2024. Vol. 10, №12. P. 298. URL: https://doi.org/10.3390/jimaging10120298
5. Zhulkovskyi O., Zhulkovska I., Kurliak P., Sadovoi O., Ulianovska Y., Vokhmianin H. Using asynchronous programming to improve computer simulation performance in energy systems. *Energetika*. 2025. Vol. 71, №1. P. 23–33. URL: https://doi.org/10.6001/energetika.2025.71.1.2
6. Zhulkovskyi O., Zhulkovska I., Vokhmianin H., Firsov A., Tykhonenko I. Application of SIMD-instructions to increase the efficiency of numerical methods for solving SLAE. *Comput. Syst. Inf. Technol*. 2024. №4, P. 126–133. URL: https://doi.org/10.31891/csit-2024-4-15
7. Zhulkovskyi O.O., Zhulkovska I.I., Vokhmianin H.Y., Firsov O.D., Riabovolenko V.A. Research of progressive tools of parallel computations with the use of SIMD architecture. *Inform. math. methods simul.* 2023. Vol. 13, №3–4. P. 228–235. URL: https://doi.org/10.15276/imms.v13.no3-4.228
8. Zheng R., Pai S. Efficient Execution of Graph Algorithms on CPU with SIMD Extensions. *2021 IEEE/ACM Int. Symp. Code Gener. Optim. (CGO)*, Feb. 27–Mar. 3 2021. Seoul, 2021. P. 262–276. URL: https://doi.org/10.1109/cgo51591.2021.9370326
9. Barredo A., Cebrian J.M., Moreto M., Casas M., Valero M. Improving Predication Efficiency through Compaction/Restoration of SIMD Instructions. *2020 IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 22–26 2020. San Diego, 2020. P. 717–728. URL: https://doi.org/10.1109/hpca47549.2020.00064
10. Sharma P., Dubey A.K., Goyal A. Hardware Acceleration Using SIMD Based Vector Processor Array to Enhance Performance of Deblurring Methods for CT and MRI Images Having Motion Blur Artifacts. *2023 Int. Conf. Smart Syst. appl. Elect. Sci. (ICSSES)*, Jul. 7–8 2023. Tumakuru, 2023. P. 1–6. URL: https://doi.org/10.1109/icsses58299.2023.10199266
11. Kamei H., Honda S., Hayashi K., Maeda Y., Fukushima N. Lookup Register-Tables with Interpolation for Effective Image Transformation on x86/64 CPUs. *2024 IEEE Int. Conf. Vis. Commun. Image Process. (VCIP)*, Dec. 8–11 2024. Tokyo, 2024. P. 1–5. URL: https://doi.org/10.1109/vcip63160.2024.10849896
12. Tai L., Zhang L., Zhou X., Zhang S. Research on Image Restoration Processing Based on Gaussian Blur Algorithm. *2023 3rd Int. Signal Process., Commun. Eng. Manage. Conf. (ISPCEM)*, Nov. 25–27 2023. Montreal, 2023. P. 384–389. URL: https://doi.org/10.1109/ispcem60569.2023.00075
13. Kelefouras V., Keramidas G. Design and Implementation of 2D Convolution on x86/x64 Processors. *IEEE Trans. Parallel Distrib. Syst.* 2022. Vol. 33, №12. P. 3800–3815. URL: https://doi.org/10.1109/tpds.2022.3171471
14. Moradifar M., Shahbahrami A. Performance Improvement of Gaussian Filter using SIMD Technology. *2020 Int. Conf. Mach. Vis. Image Process. (MVIP)*, Feb. 18–20, 2020. IEEE, 2020. URL: https://doi.org/10.1109/mvip49855.2020.9116883
15. Nixon M.S., Aguado A.S. Basic image processing operations. *Feature Extraction & Image Processing for Computer Vision*. 2012. P. 83–136. URL: https://doi.org/10.1016/b978-0-12-396549-3.00003-3
16. Intel® Intrinsics Guide. URL: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html
17. Intel® Architecture Instruction Set Extensions Programming Reference. URL: https://www.intel.com/content/dam/develop/external/us/en/documents/319433-024-697869.pdf
18. Standard library <chrono>. URL: https://learn.microsoft.com/en-us/cpp/standard-library/chrono

ПРИСКОРЕННЯ ОБЧИСЛЮВАЛЬНИХ АЛГОРИТМІВ ОБРОБКИ ЗОБРАЖЕНЬ ІЗ ВИКОРИСТАННЯМ SIMD

О.О. Жульковський, Г.Я. Вохмянін, І.І. Жульковська,
Ю.В. Ульяновська, Е.А. Рябоволенко

Дніпровський державний технічний університет
2, Дніпробудівська вул., м.Кам'янське, 51918, Україна
Університет митної справи та фінансів
2/4, Володимира Вернадського вул., м.Дніпро, 49000, Україна
Email: olalzh@ukr.net

Раціональний вибір обчислювальної платформи, оптимізація програмного забезпечення з урахуванням особливостей архітектури процесора дозволяє значно зменшити час виконання складних обчислень, підвищити продуктивність системи в цілому та забезпечити ефективну масштабованість при обробці великих обсягів даних. В роботі досліджується проблема прискорення обчислювальних алгоритмів обробки зображень, зокрема алгоритму Гауссового згладжування, із використанням технології паралелізму на рівні даних SIMD. Алгоритм Гауссового згладжування, який застосовується для зменшення шуму та видалення дрібних деталей із зображень, характеризується високою обчислювальною складністю через необхідність виконання численних арифметичних операцій над кожним пікселем. У зв'язку з цим оптимізація таких алгоритмів є актуальною задачею, а технологія SIMD відкриває можливості для паралельної обробки даних з використанням розширених інструкцій процесора, що дозволяє суттєво підвищити продуктивність обчислень. Метою дослідження є підвищення ефективності алгоритму Гауссового згладжування за рахунок SIMD-оптимізації обчислень. Розроблено два варіанти програмної реалізації досліджуваного алгоритму – скалярний та оптимізований з використанням інструкцій AVX-256. Оптимізована версія алгоритму застосовує векторизацію обчислень, обробляючи одночасно вісім пікселів у 256-бітних регістрах, що теоретично може забезпечити прискорення до восьми разів. Обчислювальні експерименти проводилися із зображеннями розмірами 1920x1080 та 2560x1440 пікселів із різними значеннями радіуса ядра та стандартного відхилення. Результати показали, що використання SIMD-інструкцій забезпечує прискорення в межах 6.9–7.3x порівняно зі скалярною реалізацією. При збільшенні радіуса ядра прискорення залишалося стабільно високим, що підтверджує ефективність підходу для більш складних обчислень. Підтверджено, що час виконання залежить переважно від радіуса ядра, тоді як зміна відхилення має менший вплив, оскільки радіус визначає зону дії фільтра. Отримане в результаті експериментів прискорення наближається до теоретичного максимуму, демонструючи переваги оптимізованої реалізації. Перспективи подальших досліджень передбачають поєднання SIMD-оптимізації з багатопоточною обробкою та вивчення можливостей використання більш продуктивних інструкцій типу AVX-512.

Ключові слова: обробка зображень, Гауссове згладжування, SIMD, AVX-256, паралельна обробка даних.