ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

UDC 681.004.89:164.053 DOI https://doi.org/10.35546/kntu2078-4481.2024.3.26

V. G. VASENKO
Student at the Department of Computer Science
Kherson National Technical University
ORCID: 0009-0003-2558-2588

N. V. KORNILOVSKA
Associate Professor at the Department of Computer Science
Kherson National Technical University
ORCID: 0000-0002-8331-8027

S. V. VYSHEMYRSKA
Associate Professor at the Department of Computer Science
Kherson National Technical University
ORCID: 0000-0002-6343-7512

M. V. KARAMUSHKA
Associate Professor at the Department of Computer Science
Kherson National Technical University
ORCID: 0000-0001-5982-4598

ACCELERATING IMAGE PROCESSING USING PARALLEL COMPUTATIONS IN OPENMP: DEVELOPMENT AND PERFORMANCE ANALYSIS OF A GRAPHICS EDITOR

In today's digital world, image processing plays a crucial role in various aspects of human life. From creating visual content for social media to analyzing medical images, powerful tools for image manipulation are in constant demand. The growing requirements for image quality and processing speed make the search for efficient methods to accelerate these processes highly relevant.

This paper investigates the development and implementation of a graphics editor that utilizes OpenMP parallel computing to accelerate data processing. This technology enables the efficient distribution of computational tasks across multiple processor cores, significantly improving system performance. The developed software offers a set of popular graphic effects, including negative, grayscale, sepia, blur, sharpening, and posterization. Each effect is implemented in both sequential and parallel modes using OpenMP, allowing for the comparison of different computational approaches.

To evaluate the effectiveness of the proposed solution, a series of experiments were conducted on images of various sizes. These experiments involved applying each effect to images ranging from small icons to high-quality photographs. A comparative analysis of the efficiency of sequential and parallel computation methods demonstrated the significant advantages of the latter. The results of the study show a substantial acceleration of image processing when using OpenMP technology. This acceleration is particularly noticeable for computationally intensive effects such as blurring or sharpening, and when working with large images. In some cases, it was possible to achieve a significant increase in processing speed, opening up new possibilities for working with large volumes of graphic data.

This research has significant practical value for software developers working on optimizing the performance of graphics editors and other image processing applications. It demonstrates how the application of modern parallel computing technologies can significantly improve the efficiency of working with graphic data, paving the way for the creation of more powerful and faster image processing tools.

Key words: *graphics editor, parallel computing, OpenMP, Qt, image processing, performance optimization, C++.*

В. Г. ВАСЕНКО
студент кафедри інформатики і комп'ютерних наук
Херсонський національний технічний університет
ORCID: 0009-0003-2558-2588

Н. В. КОРНІЛОВСЬКА
доцент кафедри інформатики і комп'ютерних наук
Херсонський національний технічний університет
ORCID: 0000-0002-8331-8027

С. В. ВИШЕМИРСЬКА
доцент кафедри інформатики і комп'ютерних наук
Херсонський національний технічний університет
ORCID: 0000-0002-6343-7512

М. В. КАРАМУШКА
доцент кафедри інформатики і комп'ютерних наук
Херсонський національний технічний університет
ORCID: 0000-0001-5982-4598

ПРИСКОРЕННЯ ОБРОБКИ ГРАФІЧНИХ ДАНИХ ЗА ДОПОМОГОЮ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ В OPENMP: РОЗРОБКА ТА АНАЛІЗ ПРОДУКТИВНОСТІ ГРАФІЧНОГО РЕДАКТОРА ЗОБРАЖЕНЬ

У сучасному світі цифрових технологій обробка графічних даних відіграє ключову роль у багатьох сферах життєдіяльності людини. Від створення візуального контенту для соціальних мереж до аналізу медичних знімків – усюди потрібні потужні інструменти для маніпуляції зображеннями. Зростаючі вимоги до якості та швидкості обробки зображень роблять актуальним пошук ефективних методів прискорення цих процесів.

У цій роботі досліджується розробка та впровадження графічного редактора зображень з використанням паралельних обчислень на основі технології OpenMP для прискорення обробки даних. Ця технологія дозволяє ефективно розподіляти обчислювальні завдання між декількома ядрами процесора, що значно підвищує продуктивність системи. Розроблене програмне забезпечення пропонує набір популярних графічних ефектів, включаючи негатив, відтінки сірого, сепію, розмиття, підвищення різкості та пастеризацію. Кожен з цих ефектів реалізовано у двох режимах: послідовному та паралельному з використанням OpenMP. Це дозволяє не тільки обробляти зображення, але й порівнювати ефективність різних підходів до обчислень.

Для оцінки ефективності запропонованого рішення проведено серію експериментів з обробки зображень різної розмірності. Ці експерименти включали застосування кожного ефекту до зображень різного розміру – від маленьких іконок до високоякісних фотографій. Порівняльний аналіз ефективності послідовного та паралельного методів обчислення показав значні переваги останнього. Результати дослідження демонструють суттєве прискорення обробки зображень при застосуванні технології OpenMP. Особливо помітним це прискорення стає для ресурсоємних ефектів, таких як розмиття чи підвищення різкості, та при роботі з зображеннями великої розмірності. У деяких випадках вдалося досягти багатократного збільшення швидкості обробки, що відкриває нові можливості для роботи з великими обсягами графічних даних.

Дане дослідження має значну практичну цінність для розробників програмного забезпечення, що працюють над оптимізацією продуктивності графічних редакторів та інших застосунків для обробки зображень. Воно демонструє, як застосування сучасних технологій паралельних обчислень може суттєво підвищити ефективність роботи з графічними даними, відкриваючи шлях до створення більш потужних та швидких інструментів обробки зображень.

***Ключові слова:** графічний редактор, паралельні обчислення, OpenMP, Qt, обробка зображень, оптимізація продуктивності, C++.*

Problem statement

The increasing complexity of image processing tasks, coupled with the proliferation of multi-core processors, has necessitated the exploration of parallel computing techniques to enhance performance. This research focuses on developing a graphics editor that effectively utilizes parallel processing to accelerate image manipulation operations.

By leveraging the computational power of multi-core processors and employing parallel programming techniques, we aim to create a high-performance graphics editor that can handle large images and complex image processing pipelines efficiently. Traditional sequential image processing algorithms often struggle to meet the demands of modern applications, which require real-time processing and high-quality results. Parallel computing offers a promising solution to these challenges by distributing the computational workload across multiple cores, leading to significant performance improvements.

The proposed graphics editor will incorporate a wide range of image processing functionalities, including image enhancement, filtering, segmentation, and analysis. These operations are often computationally intensive, requiring significant processing power to achieve satisfactory results. By parallelizing these tasks, we can reduce processing times and enable users to manipulate images more efficiently.

Furthermore, the graphics editor will be designed to handle large images, which are becoming increasingly common in fields such as medical imaging, scientific visualization, and digital photography. Traditional image processing algorithms can struggle to process large images in a timely manner, leading to bottlenecks and reduced user productivity. By leveraging parallel computing, we can overcome these limitations and enable users to work with large images seamlessly.

In addition to performance improvements, the graphics editor will also prioritize user experience. A well-designed interface and intuitive tools will make it easy for users to apply various image processing techniques and achieve desired results. The editor will also provide features for saving and sharing processed images, facilitating collaboration and workflow efficiency.

Overall, this research aims to develop a graphics editor that represents a significant advancement in the field of image processing. By effectively utilizing parallel computing techniques, the editor will provide a powerful and efficient tool for users to manipulate images, leading to improved productivity, enhanced creativity, and new possibilities for applications across various domains.

Research publications

The development of a graphics editor that effectively utilizes parallel computing to accelerate image manipulation operations is based on a significant body of previous research. Contemporary research in image processing highlights the growing need for efficient algorithms capable of handling large datasets and performing complex operations in real-time.

Key research areas that underpin this study include: studies by [1] and [2] have introduced new parallel algorithms for operations such as filtering, segmentation, and object recognition. These studies have demonstrated significant computational speedups through the parallelization of computational tasks.

Studies by [3, 4] have explored the use of Graphics Processing Units (GPUs) for accelerating various image processing operations. The results of these studies have shown the high efficiency of GPUs for highly parallel computations. User interfaces for graphics editors [5] have focused on developing intuitive user interfaces for graphics editors, allowing users to effectively perform various image processing tasks.

Specifically, studies by [6] have been dedicated to the development of parallel algorithms for Gaussian blurring, one of the most common operations in image processing. These studies have shown that the parallel implementation of this algorithm can significantly reduce computation time, especially for large images.

The main material

The aim of this study is to develop a graphics editor using the Qt library [7, 12] for image processing employing OpenMP technology [10, 11] and to compare its performance in sequential and parallel modes.

To achieve this goal, the following tasks were completed:

– Development of software with a graphical user interface based on C++ [8] and Qt, supporting a set of image processing effects in both sequential and parallel modes.

– Implementation of the following graphic effects: negative, grayscale, sepia, blur, sharpening, and posterization [9].

– Testing the execution speed of all effects in parallel and sequential modes based on measurements of the execution time of all graphic effects.

The architecture of the developed software is based on an object-oriented approach and generally consists of a MainWindow window, which also includes other component elements.

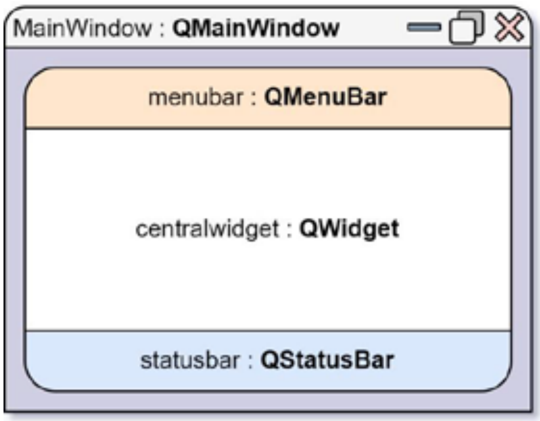


Fig. 1. Object diagram of the MainWindow window

Let's take a closer look at the components of the program's interface that provide its functionality and ease of use. QMenuBar, located at the top of the window, is an important navigation element. This menu bar provides the user with access to the extended functionality of the program. It includes key options such as opening an image from disk for further editing, saving the processed image to disk, allowing you to save the results of your work. For the user's convenience, there are informative windows with data about the program itself and its author, which allows you to better understand the capabilities and origin of the software.

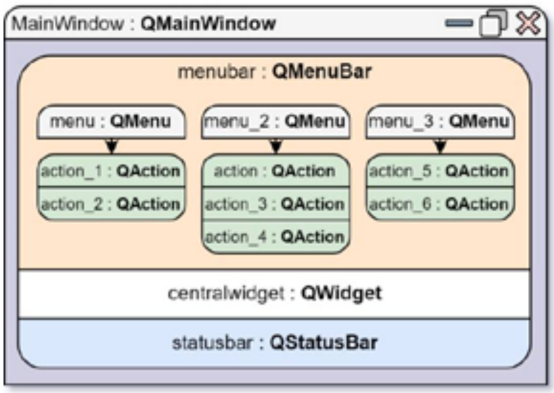


Fig. 2. Object diagram of the QMenuBar element

QcentralWidget is the heart of the program – this is the central area of the window where the main user interaction with the image takes place. It is here that uploaded files are displayed and further edited, making this component key to the functionality of the program. QstatusBar, located at the bottom of the interface, performs an important information function. This status bar is used to display various messages that inform the user about current processes and results of actions. In addition, it displays the execution time of operations, which allows you to evaluate the performance of the program and optimize image processing processes.

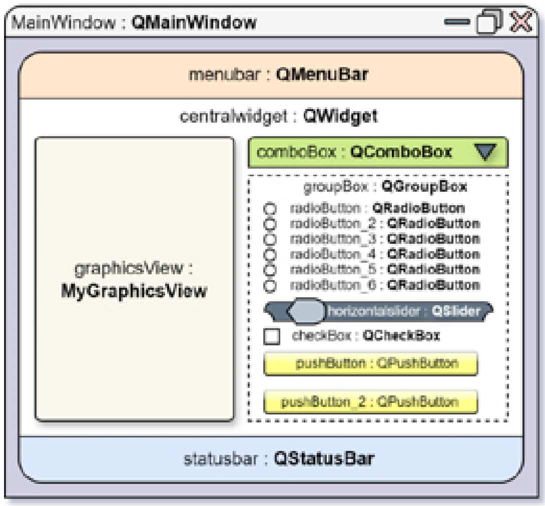

MainWindow : QMainWindow
menubar : QMenuBar
centralwidget : QWidget
graphicsView : MyGraphicsView
comboBox : QComboBox
groupBox : QGroupBox
radioButton : QRadioButton
radioButton_2 : QRadioButton
radioButton_3 : QRadioButton
radioButton_4 : QRadioButton
radioButton_5 : QRadioButton
radioButton_6 : QRadioButton
horizontalslider : QSlider
checkBox : QCheckBox
pushButton : QPushButton
pushButton_2 : QPushButton
statusbar : QStatusBar

Fig. 3. Object diagram of the centralWidget element

The centralWidget area, which serves as the main workspace of the program, can be conditionally divided into two functional parts:

The first and largest part is QGraphicsView, which occupies most of the window. This component is controlled by a specially developed class MyGraphicsView, created to ensure correct image scaling when the size of the main MainWindow window changes. QGraphicsView is the key visualization element where the image loaded by the user is displayed. Here, the user can observe the changes occurring to the image in real time when applying various graphic effects, allowing for instant evaluation of the processing results.

The second part is QComboBox, which provides the ability to change the interface theme, switching between dark and light modes, improving user comfort in different lighting conditions [9]. There is also a QGroupBox, which combines the following elements, from top to bottom:

– QRadioButton – allows you to select a specific graphic effect to apply. It's important to note that within a single QGroupBox, only one QRadioButton can be active at a time, preventing conflicts when selecting effects.

– QSlider – provides the ability to fine-tune the intensity of certain effects, such as Blur, Sharpen, and Posterize. This allows the user to achieve the desired level of image processing.

– QCheckBox – is responsible for activating the OpenMP technology, which allows for parallelizing computations, potentially accelerating image processing on multi-core systems.

– QPushButton – buttons used to apply effects to the image. Details:

– pushButton – the "Apply" button, which initiates the application of the selected graphic effect to the image, allowing you to see the processing result.

– pushButton_2 – the "Reset" button, which allows you to undo the applied effect, returning the original image to QGraphicsView. This is useful for comparing processing results with the original image or if you need to start editing again.

The QStatusBar, located at the bottom of the window, completes the interface composition. This element performs an important informational function, displaying various messages about the program's status and outputting the time spent on calculations and applying graphic effects. Such information helps the user evaluate the efficiency of different operations and optimize their workflow.

Evaluation of parallelization efficiency: To demonstrate and analyze the effectiveness of parallelization in the developed program, we will focus on the blur effect. This effect was chosen as a showcase example due to its high computational complexity, making it an ideal candidate for evaluating the benefits of parallel computing. The OpenMP method [10, 11] was used to implement parallelization, and its effectiveness was evaluated by comparing the execution time of the sequential and parallel versions of the algorithm.

Before delving into the details of the evaluation, it is worth considering the blur algorithm itself and its implementation. The blur effect is a widely used graphic technique that smoothes an image by blending the colors of neighboring pixels. This effect is often used to soften sharp edges, reduce noise in an image, or create a depth-of-field effect. Let's consider a step-by-step algorithm for implementing this effect:

1. The process begins with loading the source image into the program. This can be an image of any common format (JPEG, PNG, BMP, etc.).
2. For the sake of uniform processing, the loaded image is converted to a single format – ARGB32. This format allocates 32 bits per pixel, with 8 bits for each of the channels: alpha (transparency), red, green, and blue. Such standardization simplifies further calculations and ensures predictable results.
3. At this stage, a weight is calculated for each pixel and its neighborhood using a Gaussian function. This function creates a "kernel" – a weight matrix that determines how strongly each neighboring pixel will affect the final result. The sum of all calculated weights is stored in the variable sumTotal. Then comes an important normalization step: each weight is divided by sumTotal to ensure that the sum of all weights is equal to 1. This guarantees that the overall brightness of the image remains unchanged after applying the effect.
4. Iterating over pixels and calculating new values. This phase is the most computationally intensive. The program sequentially processes each pixel of the image. For each pixel, the following actions are performed:

– New values are calculated for each color channel (red, green, blue) based on the weights of the Gaussian matrix and the values of neighboring pixels.

– This involves multiplying the color values of each neighboring pixel by the corresponding weight from the Gaussian matrix and then adding the results.

– The resulting values are rounded and limited to the range of 0-255 to ensure correct color.

– Finally, the new pixel with the calculated values is set in place of the original one.

5. After processing all pixels, the resulting blurred image is saved. It can be displayed to the user for viewing or saved to disk in the desired format.

This algorithm, especially its fourth step, is an ideal candidate for parallelization, as the processing of each pixel is an independent operation. That is why using OpenMP to parallelize these calculations can significantly improve the performance of the program, especially when processing large images or applying intensive blurring. Let's move on to a direct consideration of the results of the comparisons.

For a thorough evaluation of the effectiveness of parallel image processing, a comprehensive study was conducted that included a series of carefully planned experiments. These experiments covered a wide range of conditions to ensure a comprehensive analysis of the performance of the developed software.

Methodology:

1. For testing, 24 images were selected, varying in size from the smallest (100x100 pixels) to extremely large (4000x4000 pixels). This range allowed us to evaluate the effectiveness of parallelization for different use cases.
2. Each graphic effect was applied to each image three times, both sequentially and in parallel mode. This ensured the statistical reliability of the results, minimizing the impact of random fluctuations in system performance.
3. For each experiment, the average execution time was calculated, which allowed us to obtain reliable performance indicators.

Experimental results: Analysis of the obtained data revealed a significant increase in performance when using OpenMP parallelization [10, 11], especially when processing large images. For example:

1. For the blur effect on a 4000x4000 pixel image, the parallel mode demonstrated an impressive speedup, being 3.5 times faster than the sequential one.

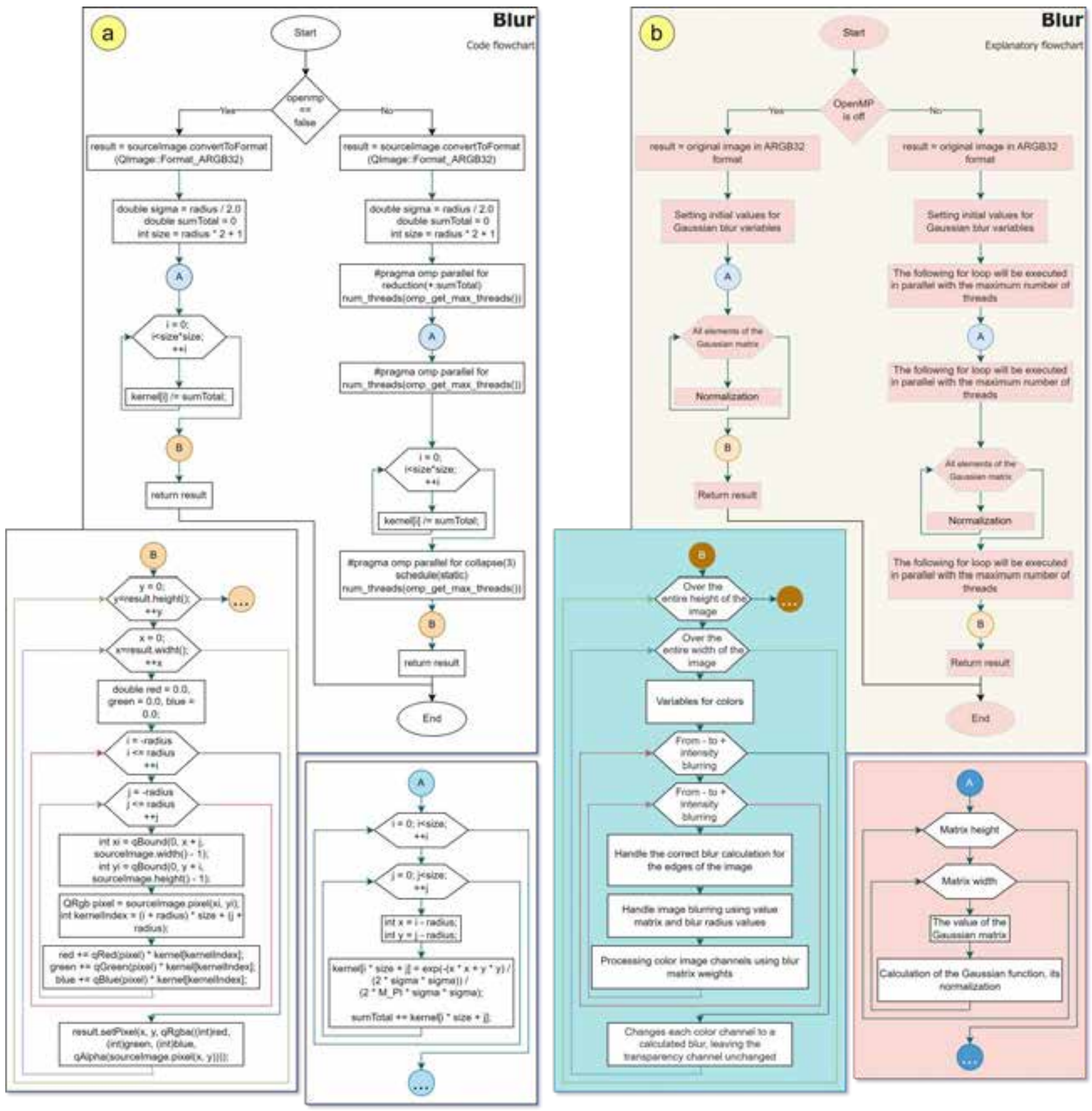
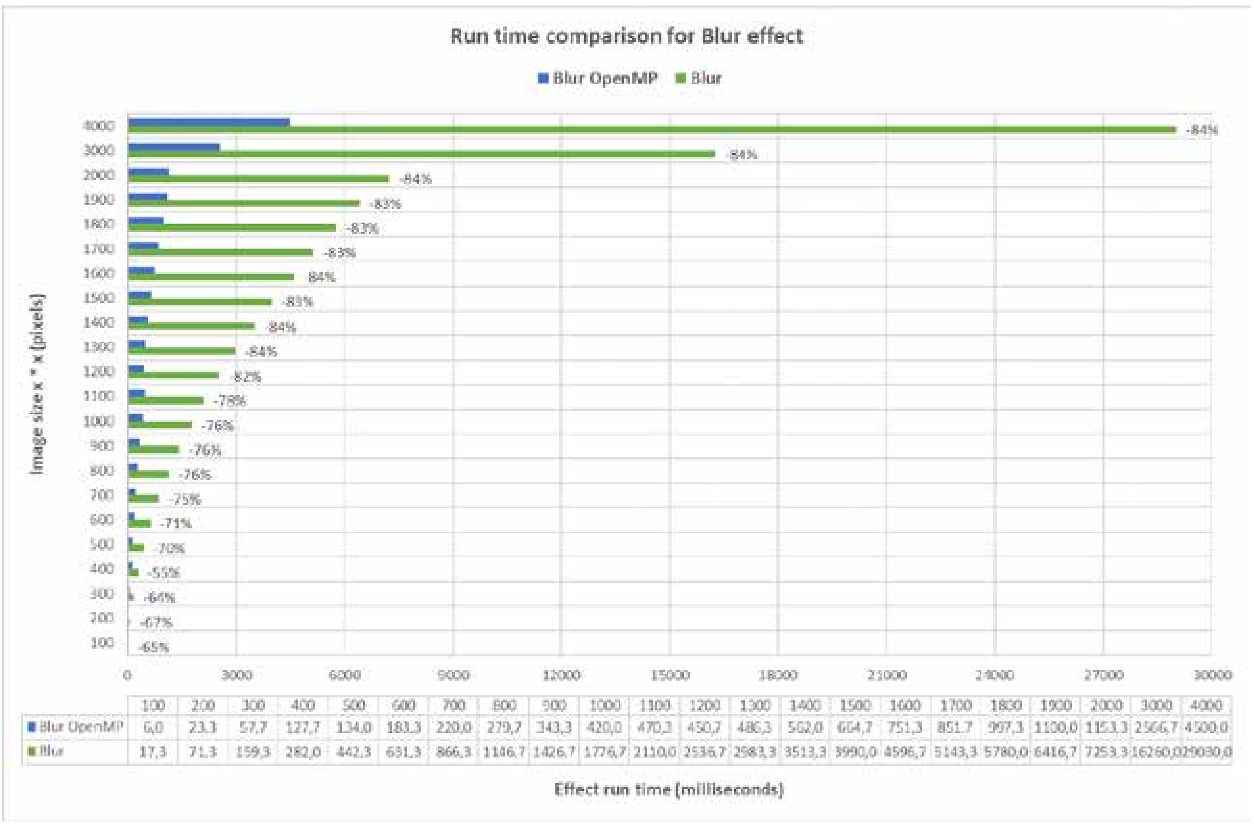

a
Blur
Code flowchart
Start
openmp == false
Yes
No
result = sourceImage.convertToFormat (QImage::Format_ARGB32)
double sigma = radius / 2.0 double sumTotal = 0 int size = radius * 2 + 1
#pragma omp parallel for reduction(+:sumTotal) num_threads(omp_get_max_threads())
#pragma omp parallel for num_threads(omp_get_max_threads())
i = 0; i<size*size; ++i
kernel[i] /= sumTotal;
return result
#pragma omp parallel for collapse(3) schedule(static) num_threads(omp_get_max_threads())
End
y = 0; y<result.height(); ++y
x = 0; x<result.width(); ++x
double red = 0.0, green = 0.0, blue = 0.0;
i = -radius i <= radius ++i
j = -radius j <= radius ++j
int xi = qBound(0, x + j, sourceImage.width() - 1); int yi = qBound(0, y + i, sourceImage.height() - 1);
QRgb pixel = sourceImage.pixel(xi, yi); int kernelIndex = (i + radius) * size + (j + radius);
red += qRed(pixel) * kernel[kernelIndex]; green += qGreen(pixel) * kernel[kernelIndex]; blue += qBlue(pixel) * kernel[kernelIndex];
result.setPixel(x, y, qRgba((int)red, (int)green, (int)blue, qAlpha(sourceImage.pixel(x, y))));
i = 0; i<size; ++i
j = 0; j<size; ++j
int x = i - radius; int y = j - radius;
kernel[i * size + j] = exp(-(x * x + y * y) / (2 * sigma * sigma)) / (2 * M_PI * sigma * sigma);
sumTotal += kernel[i * size + j];
b
Blur
Explanatory flowchart
Start
OpenMP is off
Yes
No
result = original image in ARGB32 format
Setting initial values for Gaussian blur variables
The following for loop will be executed in parallel with the maximum number of threads
All elements of the Gaussian matrix
Normalization
Return result
End
Over the entire height of the image
Over the entire width of the image
Variables for colors
From - to + intensity blurring
Handle the correct blur calculation for the edges of the image
Handle image blurring using value matrix and blur radius values
Processing color image channels using blur matrix weights
Changes each color channel to a calculated blur, leaving the transparency channel unchanged
Matrix height
Matrix width
The value of the Gaussian matrix
Calculation of the Gaussian function, its normalization
A
B
...

Fig. 4. Block diagram of the Blur effect execution (a); explanatory block diagram of the Blur effect execution (b)

2. Other effects also showed a significant improvement in performance, with a speedup of 2 to 4 times, depending on the algorithm complexity and image size.

Detailed analysis of the blur effect: This effect proved to be particularly indicative of the advantages of parallelization:

1. Even for the smallest images (100x100 pixels), an instant decrease in execution time was observed when using parallel mode.
2. The decrease in execution time ranged from 75% to 85% for most image sizes.
3. The maximum speedup (84%) was observed for images of size 1300x1300, 1400x1400, 1600x1600, 2000x2000, 3000x3000, and 4000x4000 pixels.
4. The lowest acceleration (55%) was recorded for images of size 400x400 pixels.
5. It is worth noting that the blur effect proved to be significantly more computationally intensive compared to simpler effects such as negative (Negative), grayscale (Grayscale), and sepia (Sepia). If for simple effects the execution time was measured in milliseconds, for Blur it was more appropriate to use seconds as a unit of measurement.



	100	200	300	400	500	600	700	800	900	1000	1100
🟦 Blur OpenMP	6,0	23,3	57,7	127,7	134,0	183,3	220,0	279,7	343,3	420,0	470,3
🟩 Blur	17,3	71,3	159,3	282,0	442,3	631,3	866,3	1146,7	1426,7	1776,7	2110,0

	1200	1300	1400	1500	1600	1700	1800	1900	2000	3000	4000
🟦 Blur OpenMP	450,7	486,3	562,0	664,7	751,3	851,7	997,3	1100,0	1153,3	2566,7	4500,0
🟩 Blur	2536,7	2983,3	3513,3	3990,0	4596,7	5143,3	5780,0	6416,7	7253,3	16260,0	29030,0

Fig. 5. Comparative diagram of execution time for the blur effect for sequential and parallel computation methods

Conclusion

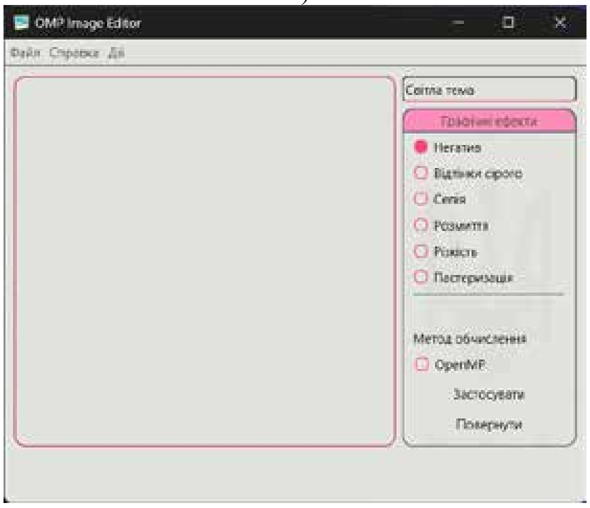
1. The analysis of the obtained results allows us to draw several important conclusions regarding the effectiveness of applying parallel computations [11, 12] in image processing.
2. The effectiveness of parallelization demonstrates a clear positive correlation with the size of the processed image. This is explained by the fact that larger images provide more opportunities for distributing the computational load among threads.
3. For small images (up to 500x500 pixels), the difference between sequential and parallel modes may be less noticeable. This is due to the fact that the overhead of creating and synchronizing threads may exceed the gain from parallel processing for a relatively small amount of data.
4. Starting from a size of 1000x1000 pixels and above, the parallel mode demonstrates a significant acceleration. Depending on the specific effect and system characteristics, this acceleration can reach 2-4 times compared to the sequential mode.
5. More complex effects, such as blur, show the greatest gain from parallelization. This is explained by the fact that they provide more opportunities for distributing the computational load among threads.
6. The results demonstrate that parallel processing provides better scalability of the solution. With increasing image size and effect complexity, the relative gain from parallelization increases, making this approach particularly valuable for processing large amounts of data or complex graphic effects.
7. The use of parallel computations allows for more efficient use of available computing resources, especially on modern multi-core processors.

Thus, the conducted study convincingly demonstrates the significant advantages of applying parallel computations in image processing, especially for resource-intensive operations and large formats. This opens the way for the creation of more efficient and productive graphics applications capable of processing large volumes of visual data in less time.

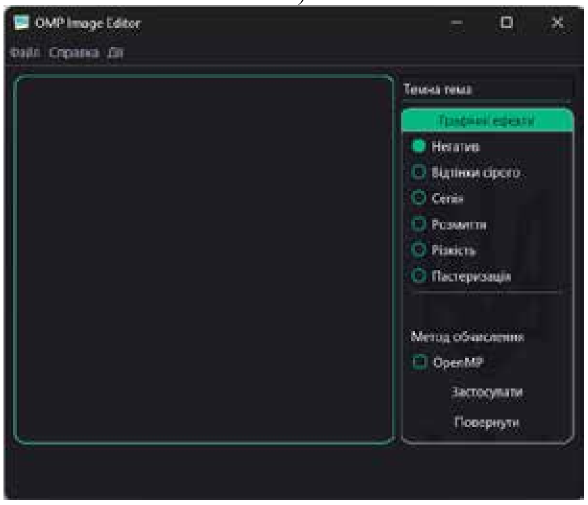
The developed graphics editor with OpenMP support demonstrates the effectiveness of using parallel computations to accelerate the processing of digital images. The results of the experimental study confirm that using OpenMP can significantly improve the performance of software, especially when working with large images and resource-intensive graphic effects.

The obtained results have practical significance for the development of high-performance image processing software and can be used as a basis for further research in the field of optimizing graphics editors and other applications that work with digital images. Further research can be directed towards optimizing algorithms for individual graphic effects, improving methods for balancing the load between threads, and adapting the developed approach to other platforms and architectures.

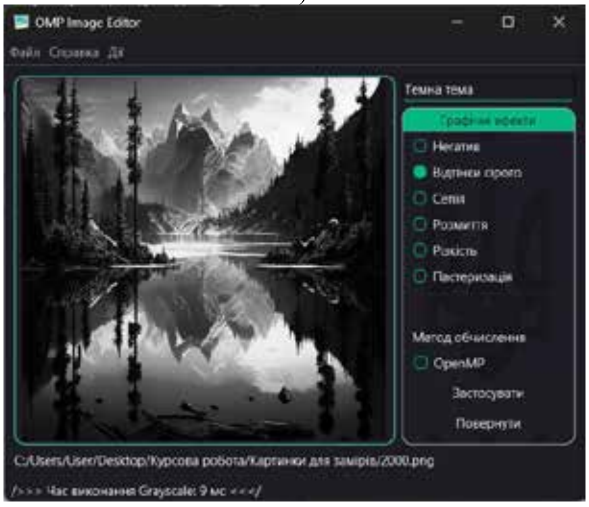
А)



Б)

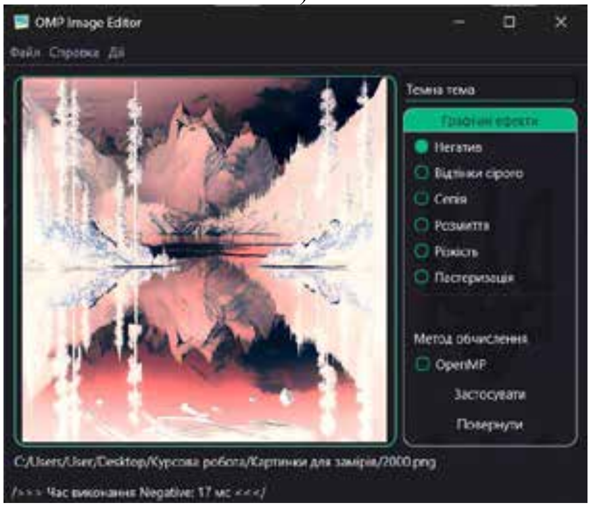


В)



```
C:/Users/User/Desktop/Курсова робота/Картинки для замірів/2000.png
/>>> Час виконання Grayscale: 9 мс <<</
```

Г)



```
C:/Users/User/Desktop/Курсова робота/Картинки для замірів/2000.png
/>>> Час виконання Negative: 17 мс <<</
```

Fig. 6. Interface of the developed program: A) light theme of the interface; B) dark theme of the interface; C) result of applying the grayscale effect; D) result of applying the negative effect.

References

1. Ladkat, A. S., Date, A. A., & Inamdar, S. S. (2016, August). Development and comparison of serial and parallel image processing algorithms. In *2016 International Conference on Inventive Computation Technologies* (ICICT). Vol. 2, pp. 1-4. IEEE. DOI: 10.1109/INVENTIVE.2016.7824894.

2. S. W. Song (2002) Models for Parallel and Distributed Computation. *Applied Optimization*. Vol.67, pp. 147-178. DOI: 10.1007/978-1-4757-3609-0_6.

3. Baumker, A., & Dittrich, W. (1996, April). Parallel algorithms for image processing: Practical algorithms with experiments. In *Proceedings of International Conference on Parallel Processing*. pp. 429-433. IEEE. DOI: 10.1109/IPPS.1996.508091

4. Haase, R., Royer, L. A., Steinbach, P., Schmidt, D., Dibrov, A., Schmidt, U., … & Myers, E. W. (2020). CLIJ: GPU-accelerated image processing for everyone. *Nature methods*, 17(1), pp. 5-6. DOI: 10.1038/s41592-019-0650-1

5. Myers, B. A., McDaniel, R. G., & Kosbie, D. S. (1993, May). Marquise: Creating complete user interfaces by demonstration. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems*. pp. 293-300. DOI: 10.1145/169059.16922

6. Del Turco, R. R. (2012). After the editing is done: Designing a Graphic User Interface for digital editions. *Digital Medievalist*, 7. DOI: DOI:10.16995/DM.30

7. Blanchette, J., & Summerfield, M. (2019). C++ GUI Programming with Qt 5: Create Amazing Applications with Qt. 2nd., Publishing House of Electronics Industry. 464 p.

8. Stroustrup, B. (2018). A Tour of C++ (2nd Edition). Addison-Wesley. 180 p. ISBN 978-0-13-499783-4.

9. Gonzalez, R. C., & Woods, R. E. (2018). Digital Image Processing (4th Edition). Pearson. 1168 p. ISBN 9780133356724.

10. Chapman, B., Jost, G., & Van Der Pas, R. (2008). Using OpenMP: Portable Shared Memory Parallel Programming. MIT Press. 384 p. ISBN: 9780262255905

11. Quinn, M. J. (2004). Parallel Programming in C with MPI and OpenMP. Dubuque, Iowa : McGraw-Hill Education, New York. 529 p.

12. Piccolino, M. (2018). " Qt 5 Projects: Develop cross-platform applications with modern UIs using the powerful Qt framework. Packt Publishing. 360 p. ISBN 178829551X.