fractal and fractional

Article

Possibilities of Use for Fractal Techniques as Parameters of Graphic Analysis †

Bogdan Popa *, Dan Selișteanu and Alexandra Elisabeta Lorincz

Department of Automatic Control and Electronics, University of Craiova, 200585 Craiova, Romania
* Correspondence: bogdan.popa@edu.ucv.ro; Tel.: +40-749143135

† This paper is an extended version of our papers published in Proceedings of the 23rd International Carpathian Control Conference (ICCC), Sinaia, Romania, 29 May–1 June 2022; pp. 82–87. https://doi.org/10.1109/ICCC54292.2022.9805946. and Proceedings of the 20th International Carpathian Control Conference (ICCC), Krakow-Wieliczka, Poland, 26–29 May 2019; pp. 1–6. https://doi.org/10.1109/CarpathianCC.2019.8765938.

Abstract: Image processing remains an area that has impact on the software industry and is a field that is permanently developing in both IT and industrial contexts. Nowadays, the demand for fast computing times is becoming increasingly difficult to fulfill in the case of massive computing systems. This article proposes a particular case of efficiency for a specifically developed model for fractal generations. From the point of view of graphic analysis, the application can generate a series of fractal images. This process is analyzed and compared in this study from a programming perspective in terms of both the results at the processor level and the graphical generation possibilities. This paper presents the structure of the software and its implementation for generating fractal images using the Mandelbrot set. Starting from the complex mathematical set, the component iterations of the Mandelbrot algorithm lead to optimization variants for the calculation. The article consists of a presentation of an optimization variant based on applying parallel calculations for fractal generation. The method used in the study assumes a high grade of accuracy regarding the selected mathematical model for fractal generation and does not characterize a method specially built for a certain kind of image. A series of scenarios are analyzed, and details related to differences in terms of calculation times, starting from the more efficient proposed variant, are presented. The developed software implementation is parallelization-based and is optimized for generating a wide variety of fractal images while also providing a test package for the generated environment. The influence of parallel programming is highlighted in terms of its difference to sequential programming to, in turn, highlight recent methods of speeding up computing times. The purpose of the article is to combine the complexity of the mathematical calculation behind the fractal sets with programming techniques to provides an analysis of the graphic results from the point of view of the use of computing resources and working time.

Keywords: fractals; parallel processing; Mandelbrot set; image processing

check for updates

Citation: Popa, B.; Selișteanu, D.; Lorincz, A.E. Possibilities of Use for Fractal Techniques as Parameters of Graphic Analysis. *Fractal Fract.* **2022**, *6*, 686. https://doi.org/10.3390/fractalfract6110686

Academic Editor: Viorel-Puiu Paun

Received: 27 September 2022
Accepted: 14 November 2022
Published: 19 November 2022

1. Introduction

Physical phenomena, chaos, and nature create truly spectacular images in their natural environment. These images can easily be framed in typical shapes, such as circles or cubes, which are hard to describe. Beginning with Euclid, the research of these irregular shapes falls under the "*investigation of amorphous morphology*", and the generation of fractal images can lead to unique discoveries in various fields.

Mandelbrot provided a foundation from which to tackle Starting from the above challenge [1] and described a geometry of nature that can be used today as an encyclopedia for generating images that can fit into forms of natural phenomena. Using this encyclopedia, many unpredictable shapes called "*fractals*" can be presented in the context of computer generation with the help of current computer technology.

Nowadays, thanks to the exponential development of the computer industry, these types of sets can be generated with the help of personal computers. Factors such as processor parameters, relative to the considered calculation time and generation time, participate in sibling image generation.

This study develops the research of the authors of [2,3]. As in those articles, the focus here is also on fractal analysis from several perspectives.

Fractal simulations can model and predict the general statistical nature of a system, without specific behavior having to be provided at a particular time. Fractal geometry has recently entered the field of graphic research, due to its uses in various scientific and technological applications, especially in the field of computer image processing. Fractal generation is a method that is successfully used as an image processing tool.

However, an easier and more efficient method of generation is needed, which allows for the calculation of fractal images in a shorter time. The amount of time required to generate images is extensive, and in the case of video systems, it is extremely long. Currently, there are many techniques used to speed up this process and at the same time reduce redundant computation. It is obvious that, being an iterative process, fractal image generation can be approached in parallel. Furthermore, an image can be composed independently for each pixel, with its color being the result of the iteration of a different complex number.

Today, process execution time is one of the determining factors of software techniques. Taking into account information transfer and processing, it can be assumed that this calculation time implies a direct impact on the resources of the used system. The focus of the current article is to present a high-performance optimization option to generate fractal images using parallel programming techniques.

A series of comparative studies between the sequential and parallel generation methods for the generation of fractal images are also presented. The focus is on the execution time, which is the main improvement objective. The obtained results confirm that the proposed method generates a difference in the generation time for fractal images.

The domain is a vast one, stretching from the characteristics working environments in the context of the mechanization of fruits and vegetables, which can be analyzed using a method of segmenting apple fruits based on fractal characteristics, as in [4], to the computing systems that have been developed based on a number of methods for covering pixels to be studied in the classification and identification of plants [5].

Fractal geometry has recently come into the limelight due to its uses in various scientific and technological applications, especially in the field of computer image processing. It is successfully used as an image-processing tool. The fractal image coding technique is one of the oldest image coding techniques available today. Currently, the methods used to form fractal images are also used in medicine in X-ray analyses of bone marrow. There are several examples of images and video streams obtained from fractal generation. However, a simpler way of generating fractals is needed, which allows for them to be calculated in a shorter time, because the time required to generate images is extensive, and for video systems, it is even longer.

There are several examples such as fractal geometry applied to urban planning [6], dynamics in different neurodegenerative diseases [7], geographical models [8], fluctuation–dissipation theorem, and hidden symmetry [9], with fractal techniques being used in different fields.

The steps described in this study center on knowledge merging and cross-cutting components. The generation procedure starts from the mathematical basis of complex non-analytical connections. The following steps refer to the software components that involve parallel programming and image processing focused on allocating the escape time (i.e., the time to finish generating algorithm). They lead to the identification and approximation of the time parameters required for a complex generation.

The article deals with the following aspects:

- The development of sequential fractal generation software solutions for Mandelbrot and Julia sets.

- The proposing of distributed computing software solutions for the generation of fractal images.
- A comparative analysis of implemented software solutions and an analysis of the computing time and influence of the processor.

To achieve the proposed objectives, a systematic and graded study was important, starting with the basic elements necessary to describe the algorithms proposed for optimization. This is how the Mandelbrot and Julia fractal sets are described. Starting from the mathematical aspects and the need for complex calculation, the design model of the Mandelbrot set was approached.

The software solution implemented based on this algorithm can be found in examples of obtained images representing the result of selected calculation parameters. In this study, the element of novelty and the impact of these processes is highlighted at the CPU level. The parallelization variant at the pixel traversal loop level is presented. This solution is much faster than the sequential version, with this being highlighted by the obtained results. Furthermore, the evolution of multi-threading and the impact of the division of tasks is presented in detail.

The areas of applicability for the proposed study are as follows. In addition to the development of computer games, other areas include:

- Real-time processing.
- Saving computing resources and energy.
- The identification/association of fractal images with natural forms.
- The compression and decompression of images.

Fractal geometry has recently come into the limelight, due to its uses in various scientific and technological applications, especially in the field of computer image processing. It is successfully used as an image-processing tool. The fractal image coding technique is one of the oldest image coding techniques available today. Currently, the methods of forming fractal images are also used in medicine in X-ray analyses of bone marrow. There are several examples of images and video streams obtained from fractal generation. However, a simpler method of generating fractals is needed, which allows for them to be calculated in a shorter time, because the time required to generate images is extensive, and for video systems, it is even very longer. Currently, there are many techniques used to speed up this process and at the same time to reduce redundant computation. Being an iterative process, it can be approached in parallel.

Concerning this article, the main contributions of this study are in algorithm efficiency. These achievements are listed below:

- Improving the long times required to generate fractal images with the sequential algorithm via the proposed parallel algorithm.
- The implementation of the sequential generation solution for image generation using Mandelbrot and Julia sets.
- The implementation of the fractal generation software solution using the parallel algorithm.
- CPU evolution analysis for test scenarios and the benchmarking of sequential and parallel implementations in terms of time and CPU resource usage.

The present article is an extension of the articles [2,3,10–12] and concentrates on the most important achievements previously presented by authors in this field.

The authors of [13] proposed a novel multicore parallel processing algorithm which was applied to investigate the fractal dimension of the Indian border, whereas the authors of [14] used task parallelism and a multi-core system to increase the speed of serial implementation by approximately 15× using a multi-core CPU.

Research in the field of fractal generation and the use of algorithms and software systems is also used in image compression or decompression, as in [15], where a multithreading-based parallelism technique using multi-core processors which minimizes the compression duration was proposed. The need for the parallelization of compression techniques was highlighted in [16], where the emphasis was also on a parallel development model similar to the one proposed in the present article.

Software systems have played an increasingly essential role in the technical development of human society due to the continuous improvement of digital information technology and computer science [17]. Technology is a growing field, which means that software system functions have become more powerful, the scale of software has become larger, and internal logical relationships have become more sophisticated, meaning that they can process a series of increasingly complex algorithms [18].

In conclusion, the main purpose of the article is to highlight the importance of parallel programming as a means of optimizing algorithms in terms of efficiency to generate fractal images in optimal time. The innovations of this work are the analysis of computing times and the proposal of a method based on the computing power of modern processors which results to quickly generated fractal images.

2. The State of the Art in Applications of Fractals for Optimization Methods

In this chapter, the state-of-the-art parallel fractal algorithms currently used to speed up generation times are presented. An extensive description of parallelization methods with development applications for Mandelbrot and Julia sets can be found in [19]. Fractal generation can include techniques using different algorithms and deal with objects that do not have integer dimensions. Early and standard examples of deterministic (mathematical) fractals include the Cantor set, the Koch curve, the Sierpinski triangle, the Mandelbrot set, and the Julia sets [20].

When it comes to classic examples of parallelization methods in the context of the present article, [21,22] are worth mentioning. When it comes to different methods of parallel computing in the context of optimizing the computing time for the generation of fractal manifolds such as the Mandelbrot set, it is necessary to make a distinction between optimization methods at the CPU level or the GPU level, such as in [23].

Nowadays, the most popular methods for parallel programming are implemented using the message-passing interface and open multi-processing (Open MP), with an example of mixing of these two being Chapel, which uses a hybrid system in a distributed-memory environment [24]. A variant of the generation of the Mandelbrot set using the two technologies is presented in [25], with this involving the use of an AMD Ryzen™ 5 2500U Quad-Core system analyzed in detail at the processor level. Other articles that use the MPI technology to speed up the Mandelbrot set are [14,26,27].

With regard to the fractals-shapes, examples of applications for the optimization and testing of Mandelbrot set generation at the GPU level take the form of the methods proposed in [23], that make use of GPU processing, which takes a shorter amount of time compared to Numba JIT and pure Python, and in [28]. Additionally, in [29], the Fastflow developed framework is used. These processing methods differ according to the field, with one of the utilities expressed [30] being related to chaos, solitons, and fractals.

The evolution of the development of computing power has been exponential, whether we are talking about processors, microprocessors, DSP (digital signal processor) systems, or new FPGA (field-programmable gate array) models in full change. Languages and software systems must keep pace with the evolution of hardware and with real-time processing. Thus, a natural image identification system based on fractal techniques is useless without a very fast response time.

Professional computing systems, clusters, and powerful computers can achieve sufficient processing times in the case of difficult tasks, but the main aim is to bring such processing into viable forms for normal users and personal systems. Examples of applications using the C# language for the implementation of fractal generation solutions can be found in [31] and [32–34].

Depending on the type of work unit, the approach to not only the generation strategy but also the language and access to resources in fractal generation methods are vast and determine a multitude of perspectives on the means of optimization. In most of the techniques addressed today, the problem of computing time is mentioned as the main optimization factor, with approaches in various areas of computational generation (CPU, GPU) being presented. Regardless of the method, the generation of fractal sets remains a good means of testing for clusters and presenting applications for distributed computing.

3. Mathematical Background

3.1. The Mandelbrot Set

The Mandelbrot set is mathematically defined by a family of complex quadratic polynomials:

$$P_c : C \to C \tag{1}$$

provided by the iteration:

$$P_c : z \to z^2 + c \tag{2}$$

where c is a complex factor. For each c, one considers the behavior of the sequence:

$$\left(0, P_c(0), P_{c(P_{c(0)})}, P_{c(P_{c(P_c(0))})}, \dots\right) \tag{3}$$

obtained by iterating $P_c(z)$. The process starts from the main point, $z = 0$, and thus $P_c(z)$ tends towards infinity or there is the possibility of it being maintained within a certain limit [35]. Furthermore, the Mandelbrot set is defined as the representation of all points, c, that do not converge to infinity. Another definition is also given by the fact that this set is a compact one distributed near the disk with a radius of 2 around its origin. Specifically, a point belongs to the Mandelbrot set if and only if $|P_c^n(0)| \leq 2$ for $n \geq 2$. The set can leave the limit if $P_c^n(0)$ becomes greater than two, and the sequence will tend to infinity.

There are several programs utilized to generate the Mandelbrot set and other fractals, some of which are integrated into various complex software tools. These programs use a variety of algorithms to determine the selection value for each pixel's color and perform a cost- and time-efficient calculation. Aspects of the escape edges of fractal points are treated by marginal multifractal analysis techniques starting from the basic functional equation. In [36], J. Barral addresses this analysis and proves the absolute continuity of the distribution of Z under simple moment conditions, a problem that is given by the Poincare equation presented in [37]. On 1 March 1980, at IBM's Thomas J. Watson Research Center in Yorktown, Heights, New York, Benoit Mandelbrot first saw a visualization of the set. This conscientious evidence pointed to unexpected places, with the outcome resembling a handful of dust scattered on a sheet of paper under Mandelbrot's eyes [38].

3.2. The Julia Set

The Julia set represents a function, f, that denotes $J(f)$. The Fatou set is denoted as $F(f)$. These names are derived from the French mathematicians Gaston Julia and Pierre Fatou, who are known for their study of complex dynamics in the twentieth century.

Let $R(z)$ be considered a fractional function:

$$R(z) = (P(z))/(Q(z))\left(0, P_c(0), P_{c(P_{c(0)})}, P_{c(P_{c(P_c(0))})}, \dots\right) \tag{4}$$

where $z \in C^*$, C^* is considered the Riemann sphere, and P and Q are considered polynomials without common divisors. The "coloring" of the Julia set is represented by $J(z)$, which represents the set of points z. The coefficients of the points z do not tend to infinitely. After which, $J(z)$ is reiterated. The Julia square sets can be generated by the square represented by the mapping:

$$Z_{n+1} = Z_n^2 + c \tag{5}$$

for a fixed c. For each c, this transformation generates a fractal.

The Julia set is composed by completing the set by recalling periodic points [39]. This method provides a description of the Julia set, which assumes both a topological and a dynamic characterization of the set. The difference between the two sets lies in the form of the function; the Mandelbrot set can be said to be a special case of a Julia set, since the second set gives a fraction with two distinct polynomials.

3.3. Designing the Mandelbrot Set

Starting from the previously presented mathematical basis, it can be said that the simplest algorithm for computing a graphical representation of the Mandelbrot set is called the "*escape time algorithm*". A repeated iterative calculation is performed for each of the x and y coordinates of each point in the defined plane. Starting from the evolution of this calculation, a color is chosen for the responsive pixel defined by x and y (the locations of each point).

The obtained results represented by the complex parameters of each iteration are used as the starting values for the next iteration, with this cycle continuing until a critical threshold according to Equation (3). The values are checked during each iteration to see if they have reached an "*escape*" value or not (in which case the coordinates are saved). If this condition is reached, the calculation is stopped, the pixel is projected, and the x and y values for the next point are examined.

For different starting values, the calculation time is short, and after only a minor number of iterations can one of the cycle's stop conditions be met, an aspect generated by the chaotic nature of the squares of complex numbers. For remarkably close initial values, there is also the possibility that the process takes a long time due to hundreds or thousands of iterations necessary to fulfill one of the escape conditions (stop).

The algorithm is based on the choice of the "*depth*" you want to examine the fractal shapes, despite the computational cost, and this is given by the number of iterations allowed. Therefore, higher the number of iterations, the more beautiful details can be identified in the image. Furthermore, the computation time will have an exceedingly high value for the computation and design of a fractal image.

There are also more complex calculation methods, but these detect the possibility of stopping faster and include the possibility of calculating the distance from the origin to the point. Using the Pythagorean theorem, and in the context of the distance being greater than 2, the point has reached the end. Another method used to program this set faster is the Buddhabrot method [40], which is based on identifying the "*escape*" points and finding the areas where they are.

Black is usually used to illustrate values that do not exceed the set limit until at the set repetition limit, and gradually lighter colors are used for points that do not meet these conditions. These points provide a visual representation of how many cycles the point can be assigned a color or not. The definition of the Mandelbrot set, together with its basic properties, suggests a simple algorithm for drawing an image of its set.

The region in the selected complex plane is subdivided into a certain number of predetermined pixels relative to the dimensions of the image that is to be generated. To be able to color any area with such pixels, the procedure is as follows: first of all, it is iterated starting from the critical point 0 under $P(c)$. After this, a verification operation is performed at each step if the point at each moment of generation has a modulus greater than 2. When this happens, we can say that the value does not belong to the Mandelbrot set and color that pixel according to the number of iterations used.

Otherwise, it keeps iterating until a fixed number of steps. After which, it decides whether the parameter is, in the Mandelbrot set, the most likely scenario, or at least very close to it, and displays that pixel as black. The algorithm does not use complex numbers but simulates the transposition of complex number-type operations with the help of two real numbers (of float type).

The generation program can be shortened if the programming language contains default complex operations with complex data types. The usual sequential algorithm for developing the Mandelbrot sets displays images from the entire set of pixels in a rectangular area selected while checking all pixels (corresponding to the constant c factor) to choose whether it belongs to the Mandelbrot set or is outside the set.

Example of the algorithm in pseudocode:

```
for x parameter from 0 value to width of the image [2]for y parameter from 0 value to height of the image
    set the count number to 0
    z=c //c represents a complex number characterized by the real part with the value
    //similar to x and the imaginary part that is equal to y
    do z=z2+c //z also a complex number
    count=count+1
    while ($|z|<2.0$) and (count<Maxnumber of iterations)
    if $|z| \leq 2.0$ paint the Pixel with color/*The pixel belongs to the Mandelbrot Set*/
```

where z is the complex variable. This variable is preset based on a ratio between *Min* and *Max*, real and imaginary, correlated with the image size.

In the calculation formula of the color of each pixel based on a predefined color palette, the most important factor is the free complex variable, c, accordingly to the calculation of the Mandelbrot set [11].

Furthermore, important factors in terms of working time are the image size and the number of iterations, which can be one of the most important test factors in terms of efficiency. This variable is preset with a value with represents a ratio between the min. and max. values, according to the real and imaginary components correlated with the image size. Thus, the complex variable c will have been assigned a distinct, consecutive value for every iteration of color determination for each pixel in the plane.

Thus, the computation formula introduces various variables for each round of processing. This component increases the degree of diversity in terms of the colors that make up the images, with this being an aspect that also supports the theory of fractal components. To obtain colored images related to the set, the assignment of color for each value for the number of iterations performed can be achieved using a different functional model (linear, exponential, etc.). Furthermore, new algorithms have been devised to compute filled Julia sets. A number of beautiful and exciting figures of newly filled Julia sets are included to demonstrate the power of and fascination behind the composed images [41]. Another perspective involves using the Fibonacci literacy process to explore the Julia and Mandelbrot sets by establishing the escape criteria of a transcendental function, as in [42].

4. Software Implementations

Parallel computing is a current approach arising from the evolution of hardware components, in which many operations are performed simultaneously [43]. The basic principle of parallel programming is that large problems can be broken down into smaller subproblems, which are then solved simultaneously ("in parallel"), much the same as in the *divide et impera programming method*.

It can be said that for such iterative computing scenarios, for which the independent calculation of different parameters is repeated, there is no point in waiting for the completion of an operation, for they are practically independent. Thus, we can distribute this "*multiple activity*" time to several independent computational possibilities.

The specific code for a visualization of the Mandelbrot sequential set can be thought of as the following (easily programmable in almost any language):

```
Double type variable MinReal = −2.0 (limit);
Double type variable MaxReal = 1.0 (limit);
Double type variable MinImag = −1.2 (limit);;
Double type variable MaxImag = MinImag+(MaxRe-MinRe)*ImageHeight/ImageWidth;
Double type variable Re_factor = (MaxReal-MinReal)/(ImageWidth-1);
Double type variable Im_factor = (MaxIm-MinIm)/(ImageHeight-1);
for(unsigned y=0; y<ImageHeight; ++y)
{
```

```
double c_im = MaxImag − y*Im_factor;
for(unsigned x=0; x<ImageWidth; ++x)
{
double c_re = MinReal + x*Re_factor;
//Calculates whether c belongs to the Mandelbrot set or
//no and draw the pixel by the coordinates (x,y) accordingly
}
}
```

Put simply, in the formula that calculates the value assigned to the color of each pixel, based on a predefined color palette, the most important factor is the free complex variable, c, from the calculation of the Mandelbrot set.

This variable is preset based on a proportional calculation between *Min* and *Max*, which are real and imaginary values correlated with the requested image size. The complex variable c will have a distinct, consecutive value for each iteration of determining the value for setting the color for each pixel in the plane. Thus, the calculation formula will have as starting data a different variable for each round of processing and the variety of colors assigned to each pixel will automatically increase.

The code involves the iterative traversal for each pixel within the image for which the equivalent complex number is calculated for the components in *c_re* and *c_im* (parameters of the complex number C). The preceding code structure is the basic iterative sequential core of computing the value of each pixel in the Mandelbrot set based on a preset number of iterations or overruns for the modulus of $C < 2$.

The standard type is the most used method of coloring for the exterior of the set. It uses the value "n" after the inner loop has finished its iteration. This is done by using the number of iterations to achieve a value for the function with a modulus greater than 2 (value). After the inner loop has ended its cycle (with n being the number of iterations required), a value for n is obtained which is between 0 and the maximum number of preset iterations.

If n is the maximum total iterations, then the function is established to have a modulus greater than 2, the current C is (most likely) part of the set, and it can be colored, for example, black (or any color we want). On the other hand, if n is less than the maximum number of iterations, then it is known that c does not belong to the set, and then one can map the value of n to a color and set the pixel to that color.

In [44], following the presentation of this application in the context of a computing time performance evaluation, the comparative analysis of the algorithm complexity via a theoretical estimation and practical results was proposed. Thus, it was determined that the number of iterations of each cycle is difficult to approximate, with this being an aspect of the chaos component of the fractal theory.

After an analysis of the code core that generates such an image utilizing the Mandelbrot set, the following aspects must be considered to establish the output conditions:

- The order of complexity is $O(n^2)$.
- The Maxiteration parameter—a barrier that limits the number of iterations that represents the main factor in the working time analysis.
- The condition $|z| < 2.0$ offers the option of exiting the loop after 1 or Maxiteration steps.

The architecture presented in Figure 1 shows a simple explanation of the system for coloring an image using the Mandelbrot set rendering. The six steps are as follows:

(1) Setting the default parameters.
(2) The adaptation of image parameters according to size.
(3) The choice to operate in parallel or sequential mode.
(4) Part 1 of the Mandelbrot requirements—generating the values of the parameter c.
(5) Part 2: testing the exit conditions and applying the loop.
(6) Part 3: complete the final coloring.

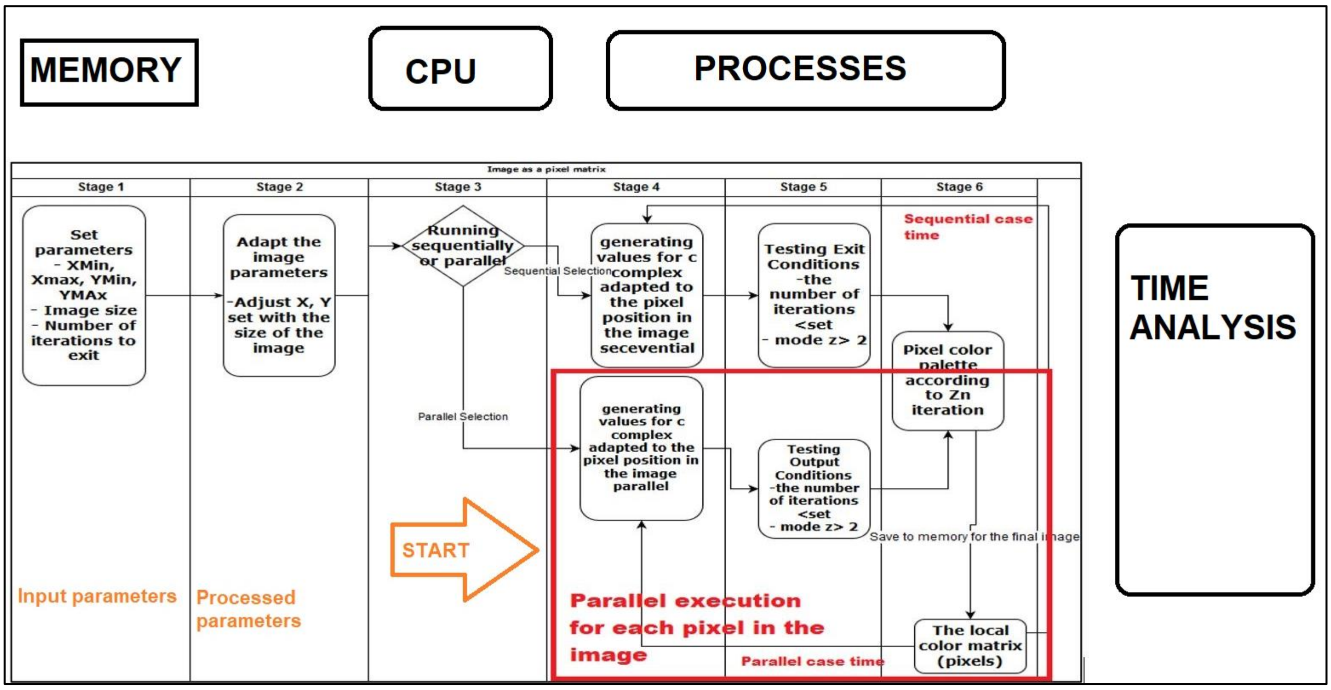

MEMORY
CPU
PROCESSES
TIME ANALYSIS
Image as a pixel matrix
Stage 1
Stage 2
Stage 3
Stage 4
Stage 5
Stage 6
Set parameters - XMin, Xmax, YMin, YMAx - Image size - Number of iterations to exit
Adapt the image parameters -Adjust X, Y set with the size of the image
Running sequentially or parallel
Sequential Selection
Parallel Selection
generating values for c complex adapted to the pixel position in the image seceventiall
Testing Exit Conditions -the number of iterations <set - mode z> 2
Sequential case time
Pixel color palette according to Zn iteration
generating values for c complex adapted to the pixel position in the image parallel
Testing Output Conditions -the number of iterations <set - mode z> 2
Save to memory for the final image
The local color matrix (pixels)
START
Input parameters
Processed parameters
Parallel execution for each pixel in the image
Parallel case time

Figure 1. Description of the functional architecture of the application. Highlighting for the Mandelbrot parallel generation variant.

The diagram presented in Figure 1 also shows the marking in a red perimeter of the routines utilized in parallel. The iterative system that allows independent computation is also presented. Being a similar redundant iterative process for each determination, the optimization solution is to approach the calculation for each independent pixel. The main idea proposed in this paper is that of parallelization, which assumes that for the determination of the fractal image, the computing for each pixel is independent of others. This can represent an iterative and parallel development platform as a test for the processor used.

Additionally, in Figure 1, the aspects of analysis, the problem of used memory, and the problem of processor capacity, with this having a direct impact on computing and processing power, are presented. The processing aspect, which is internal at the CPU level, but also the focus of this work at the level of time analysis, is also mentioned.

4.1. Sequential Implementation

Currently, image processing is a field of great interest and involves the pre-presentation, compression, reconstruction, classification, recognition, and analysis of digital images. One of the most important practical uses of fractals is in compressing and storing images with very small values in terms of space. Each file, based on seven float parameters, can store an image of over 10,000 pixels. Put simply, at the storage level, an image made up of two Mega pixels can be stored with 7×32 bits, i.e., 224 bits [12]. Thus, in the context of data storage systems for images, fractal generation methods are very effective.

If we approach the analysis aspect from the point of view of data storage systems, i.e., the memory used, it can be concluded that the systems based on fractal calculations are very efficient. Thus, complex images can be stored with the application of only a small number of starting parameters. Taking the input parameters and the previously presented algorithm as a starting point, a series of software applications were developed for generating fractal images.

Figure 2 shows the application's top-level architecture, presenting input information that is processed at the top level with respect to the Mandelbrot algorithm, Julia fractions, or other fractal generation methods. In the end, the decision related to each pixel of the image will be made, as will a decision related to the colors selected for coloring. The general design of the application is presented in Figure 2. The application uses seven items of input data, distributed as follows:

- Four details for the parameters used in the initial state variable of the Mandelbrot set (XMin, XMax, YMin, YMax), which are directly associated with the image dimension so that distinct values are allocated for the variable c from the set computation for the state related to the $P_{c(0)}$ of the Mandelbrot iteration.
- Two items of data associated with setting the size of the image to be produced.
- The maximum number of generations, for which it is decided if the value is or is not within the bounds of the set. Depending on the default values, the color associated with the current interaction is decided.

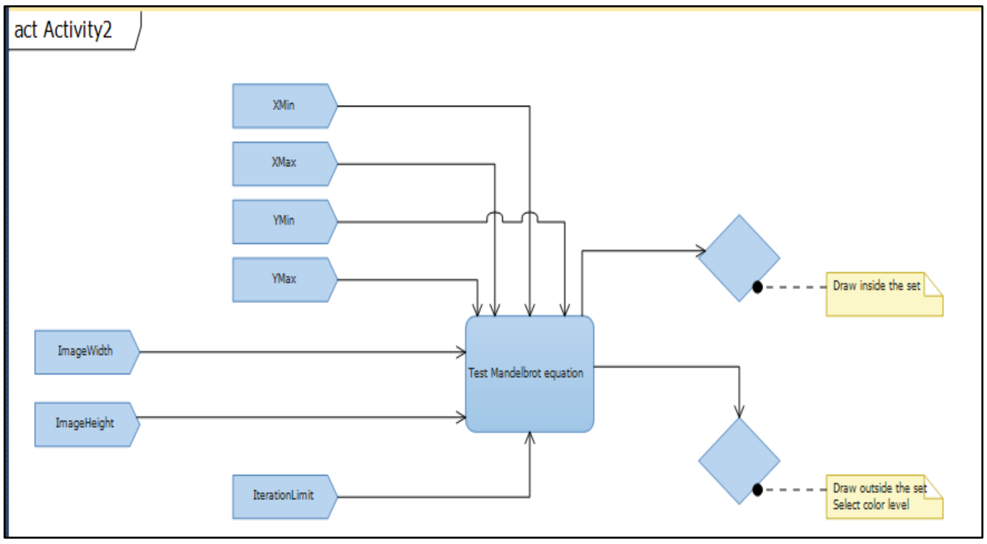


Figure 2. General application architecture of the sequential Mandelbrot algorithm.

The development project was implemented in the C# language [45], which is a multi-paradigm programming language that includes powerful, imperative, declarative, functional, generic, and object-oriented (class-based) calls and requires a component-oriented programming architecture. It was developed by Microsoft as part of its. NET initiative and was then approved as a standard by Ecma (ECMA-334) and ISO (ISO/IEC 23270: 2006). C# is one of the programming languages designed for popular language infrastructure. C# is intended to be an easy, modern, general-purpose object-oriented programming language [46].

Figure 3 shows the dependencies and relationships at the class level. These assume internal methods for the class called state. This class primarily refers to setting and retrieving the parameters (the seven previously presented) at a certain time during the processing.

Figure 4 shows, in detail, at the class level, the methods utilized in the description of the Mandel main class. This class involves the calculation of the parameters at the end of the algorithm and the determination of the output parameters. The main method, referred to as *CalculatePotentials*, consists of the implementation of the sequential *Escape-time algorithm*. This method uses, in turn, another method that performs the calculation only at the iteration level for complex numbers represented by real and imaginary parameters.

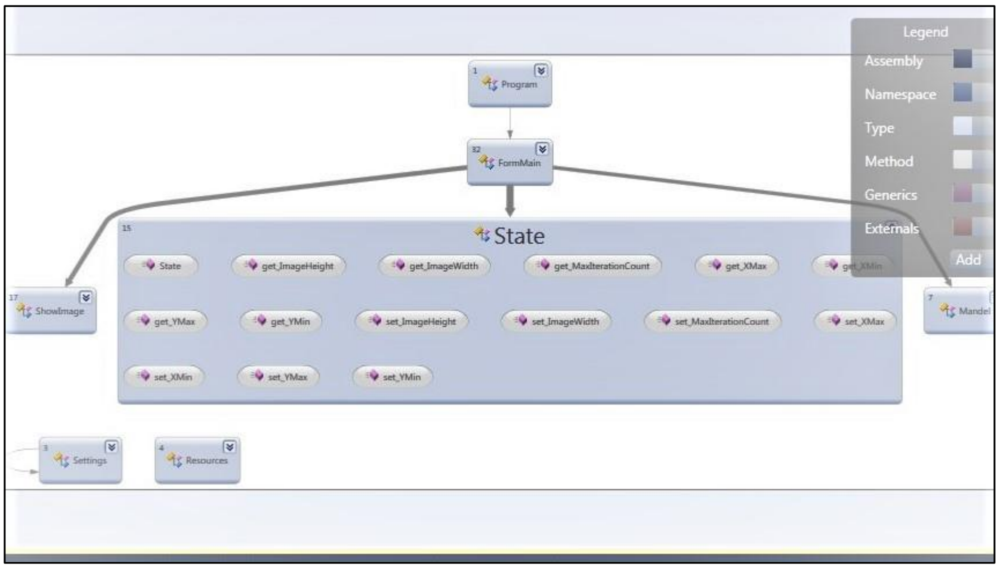


Figure 3. The class dependency's structure.

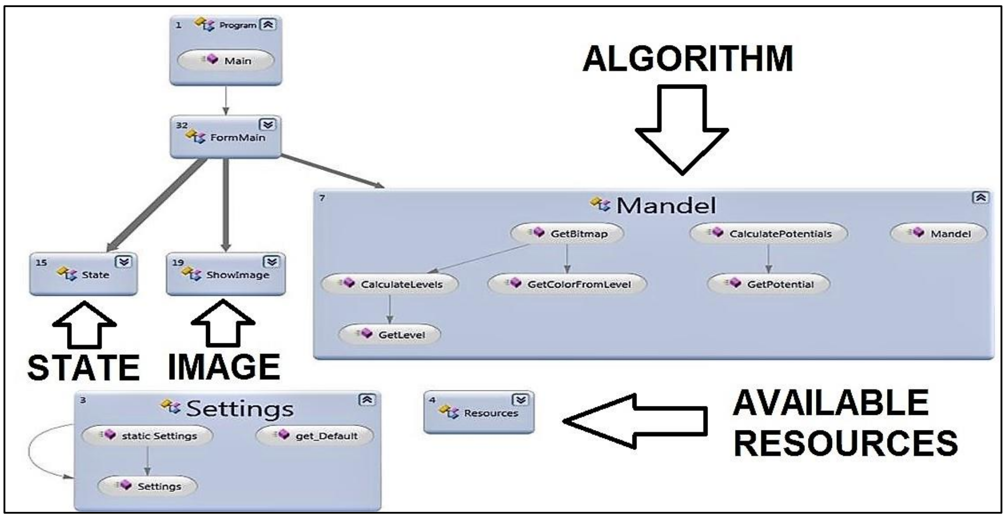


Figure 4. Mandelbrot Class Architecture—Main class.

The interface aspect of the project was also presented in [10]. This environment can generate large fractal images that can later be used, as an input environment, in the field of image compression, as presented in [15].

Following the testing of the developed sequential application, two factors were identified:

(1) The computation time for the sequential algorithm increases exponentially for larger images and parameters with larger values.

(2) During image generation, the set generation component may take longer than the computation of the iteration. It has been observed that, at the level of code metrics, the effective portion of the calculation can represent 34/800 effective lines of code or, in terms of cyclomatic complexity, a ratio of 7/160 to the entire program [12].

In the example of real-time systems, late detection decisions mainly arise as a result of the inability to of a processor to manage and can render the fractal model inappropriate. Given current processing capacities, a solution to this issue is imminently needed. This affects and increases their scope. If fractal images were used for detection systems, image storage, or large quantities of information, the need for high-speed processing would be the main factor in their use.

The proposed application was constructed in the C# language and allows the user to interactively explore the Mandelbrot set and save the generated images as well as the coordinates of the implemented viewing window. The parameters Xmin, Xmax, Ymin, and Ymax, shown in the application interface in Figure 5, define the watching window for the Mandelbrot set. The values can be modified manually, with the other parameters being revised automatically. One of the parameters on display is the width-to-height ratio for the existing viewport. The *maximum iteration count* establishes the number of iterations next to which a given point is represented as being out of the Mandelbrot set. The specific instructions available to the user are as follows:

(a) Calculated: restored the main image.
(b) Save: saves the main view to a file, as a Windows bitmap. It also stores the parameters in an XML file with a similar name.
(c) Reset: restores all parameters to initial values.

The application interface is presented in detail in [11], along with a series of images obtained with various parameters.

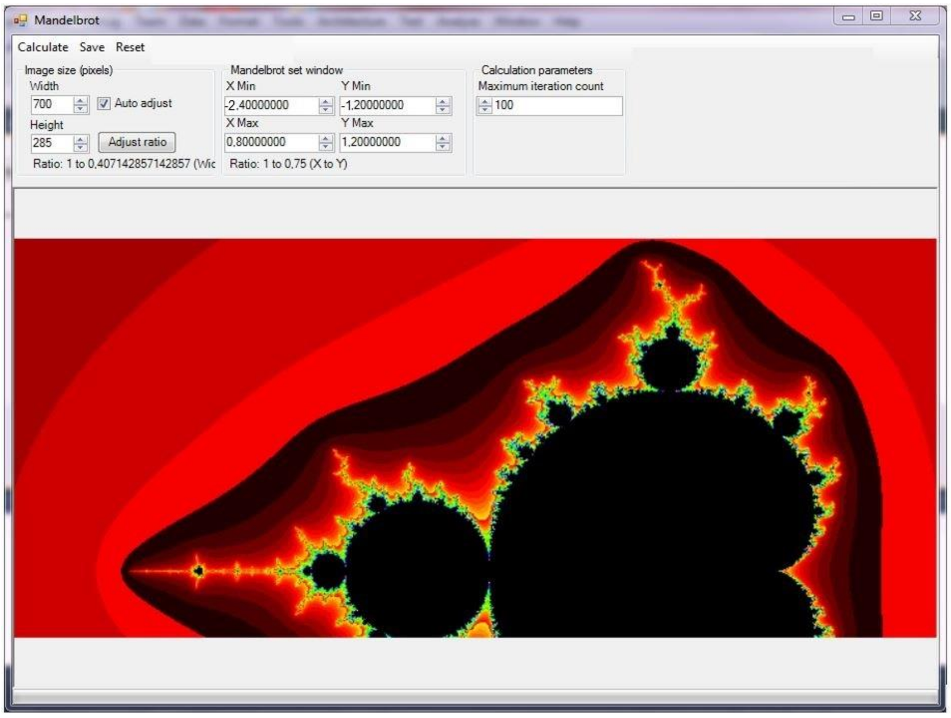


Image size (pixels)
Width 700 [✓ Auto adjust]
Height 285
Ratio: 1 to 0,407142857142857 (Wic

Mandelbrot set window
X Min -2,40000000 Y Min -1,20000000
X Max 0,80000000 Y Max 1,20000000
Ratio: 1 to 0,75 (X to Y)

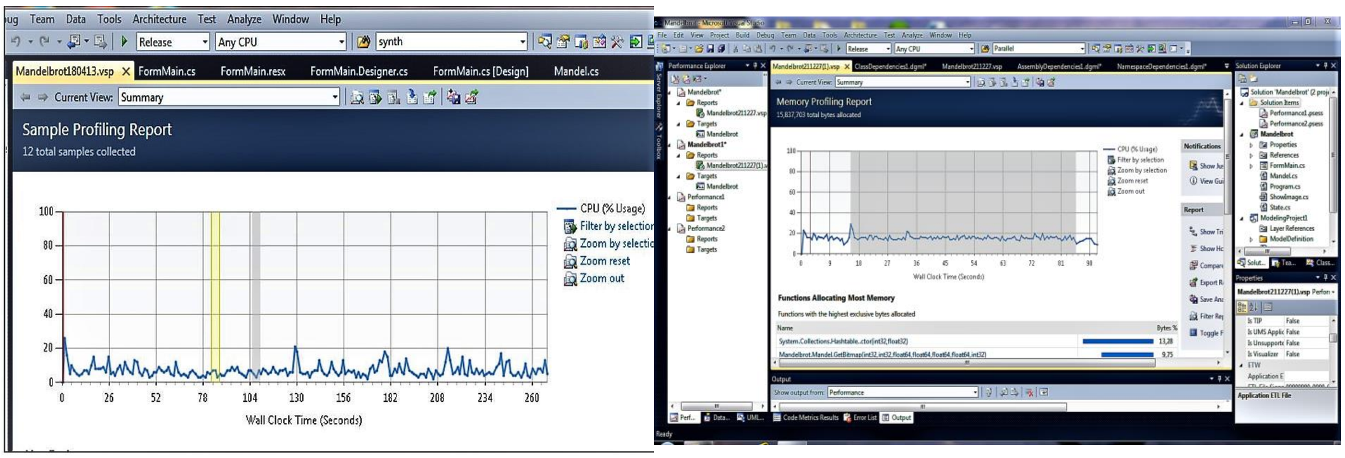
Calculation parameters
Maximum iteration count 100

Figure 5. An example of the use of the developed interface and the generation of a fractal image.

4.2. CPU Analysis for Sequential Tool

Fractal generation systems can be useful, for instance, in image storage systems based on fractal generation rules. Such systems are considered efficient in terms of space and storage/restoration rules. Even the giant Microsoft used a fractal-based storage system in the 2000s in the famous Encarta encyclopedia. The classes used to develop this application represented a mixture of functions and Windows libraries but also of new classes formed for calculating the Mandelbrot generation and for representing the image.

One of the most important analyses for the current application is shown in the graph in Figure 6a. This graph was made using this analysis support provided by the Microsoft Visual Studio 10 environment and provides an insight into the percentage of process utilization while the application is running.


Sample Profiling Report
12 total samples collected
Current View: Summary
CPU (% Usage)
Filter by selection
Zoom by selection
Zoom reset
Zoom out
Wall Clock Time (Seconds)
Memory Profiling Report
15,837,703 total bytes allocated
Functions Allocating Most Memory
Functions with the highest exclusive bytes allocated

(**a**) (**b**)

Figure 6. Evolution of CPU usage percentage over time. (**a**) During the running of a generation scenario, to support the interface, CPU resources are used at a rate of 20% over 260 s of test time. (**b**) Example of normal testing for a 90-s pattern. It is observed that the maximum level of use of the processor does not exceed 30% of the resources and that the average is approximately 20% in the random sequential generation test.

When running a build script, 20% of CPU resources are used to support the interface. After starting the application, several distinct builds can be performed depending on the set parameters. It can be seen in Figure 6a that, unlike when setting or changing parameters, resulting in a usage CPU below 5%, the only moments that significantly demanded computing power are around seconds 0, 26, 130,156, 182, 215, and 240, moments in which image generation was carried out. Considering this process was carried out for sequential iterative generations, it can be noted that the process can overload the computing power in the case of these applications.

Overcoming this involves another approach to streamlining which leads to C# application development and the transformation of the computation for sequential generation into parallel generation.

Figure 7 shows an example of an image generated starting from the preset parameters and using the developed application. Thus, images such as the one shown in Figure 7, can be generated in a faster time. The sequential process is redundant, does not provide reliability, and does not fully utilize the process resources.

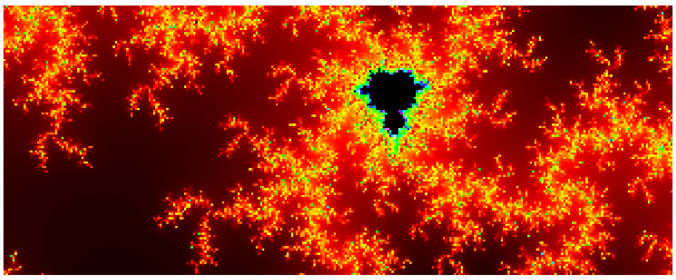


Figure 7. Mandelbrot image generated with XMin equal to 0.2035629459272944, XMax equal to 0.20243564914073117, YMin equal to 0.81859534365893782, YMax equal to 0.81814009408410127, a Width of 300, a Height of 121, and a limit of 678 iterations.

All these aspects lead to a extended generation time. The internal time calculation is set as the difference between the start of the calculation to determine the color of the first pixel and the completion of the entire image. Since this calculation is more difficult and requires more computing power to process sharper complex images, the following examples will focus on the execution cycle for setting the color of each pixel in parallel.

5. The Proposed Solution and Measurement Results

5.1. Proposed Solution

The proposed optimization variant involves several objectives:

- Communication between tasks should not complicate the calculation.
- The calculation should be applied in an appropriate time for the proposed sequential and parallel scenario without extra time being needed for other processes.
- Effective use at the maximum capabilities of the multithreading processor used.

In the case of the parallel calculation proposal for calculating the coloring value for each pixel at the level of the Mandelbrot sets, this is achieved with the help of *Parallel FOR* loops at the coordinate level with a matrix of states considered to be stored as shared memory.

Another parallelization strategy can be the classic model of the producer-consumer algorithm, but, from the point of view of efficiency, this involves direct access to resources and can generate much slower execution times.

Starting from the scheme proposed in Figure 1, it can be said that only part of the actual process of the application will be carried out in parallel, i.e., the part related to the execution of the Mandelbrot algorithm. To calculate this area, an example of the code is as follows:

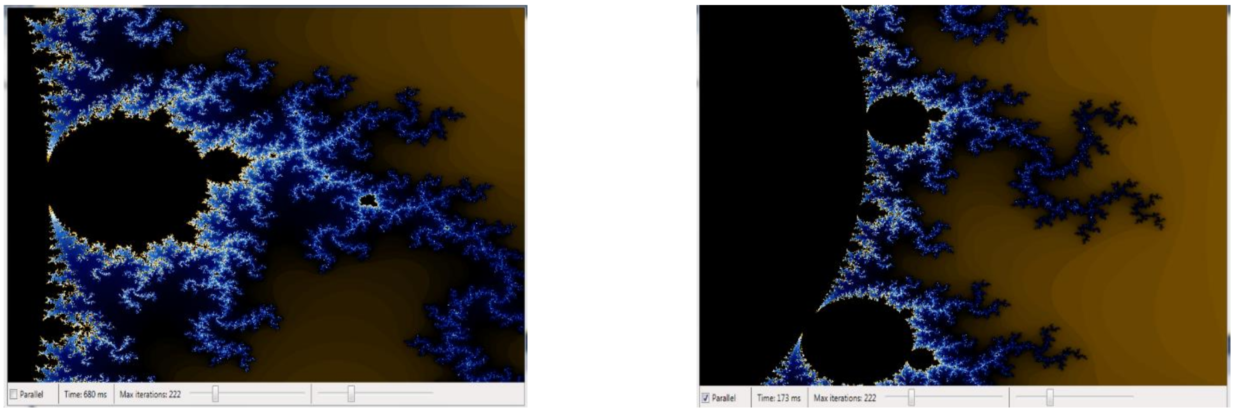
```
private void UpdateView()//Update image View [2]
{
frame.IterationsMax = (int) IterationsMax;
frame.Height = (int) Grid1. CurrentHeight;
frame.Width = (int) Grid1.CurrentWidth;
frame.Sol = UseParallel ? parallelSol: sequntialSol;//execution choice
Cursor = Cursors.Wait;
var y = new Stopwatch();//Stop time for count [47]
y.Start();//Start the processes
frame.Update();//Bring up to date with the Mandelbrot calculus
CalculationTime = y.ElapsedMilliseconds;//time counter
UpdateImage();//Update the image template
Cursor = Cursors.Arrow;
}
```

The interface is simple, being focused on two components, one of a graphical nature, the selected color palette, and another of a technical nature, which refers to the number of iterations that the fractal calculation involves. All these calculations contribute to the displayed execution time.

This method highlights the sequential computation time compared to the parallel one. The parallelization process refers to the basic condition for generating fractals, the structure made up of the two “for” loops. They are implemented in parallel mode; this process is elementary for the C# type language and is a standard function. The loop determines independently, and distributed, the final value for each value in the plan.

The internal time calculation is set from the time the calculation starts for the first pixel until the entire image is finished. Since this calculation is more difficult and requires more computing power to process sharper complex images, the following examples will focus on the execution cycle for setting the color of each pixel in parallel. The following approach analyzes the results obtained within the generation of the Mandelbrot set and considers the performance time factor for these difficult tasks.

There are two scenarios presented in Figure 8: the simple generation scenario, which is time-expensive, 680 ms, for the sequential Mandelbrot algorithm, and the parallel one, which is calculated in 173 ms, with these scenarios calculated at a 222 iterations limit.



(**a**) (**b**)

Figure 8. Evolution of time generation for sequential algorithm and parallel algorithm. (**a**) Image generated with sequential algorithm. (**b**) Image generated with a parallel algorithm.

A graph of the sequential and parallel implementations of Mandelbrot algorithms and a fractional case of the Julia algorithm were previously presented in [11]. Additionally, analysis of scenarios in terms of computing time using different processor models were presented in detail in [3].

The generation process has a series of repetitive iterations behind it to generate the color for each pixel in the image. This process can be considered a test case for CPU capacity and is a good test case for parallelism.

5.2. Evaluation Results

Figure 9 illustrates a test case with a duration of 55 s. The test was divided as follows:

- The first 9 s represented the evolution of sequential generations.
- For the next 15 s, a change was made for the parallel generations.
- In the period between 35–42 s, a surplus of resources was used in the case of parallel generation.

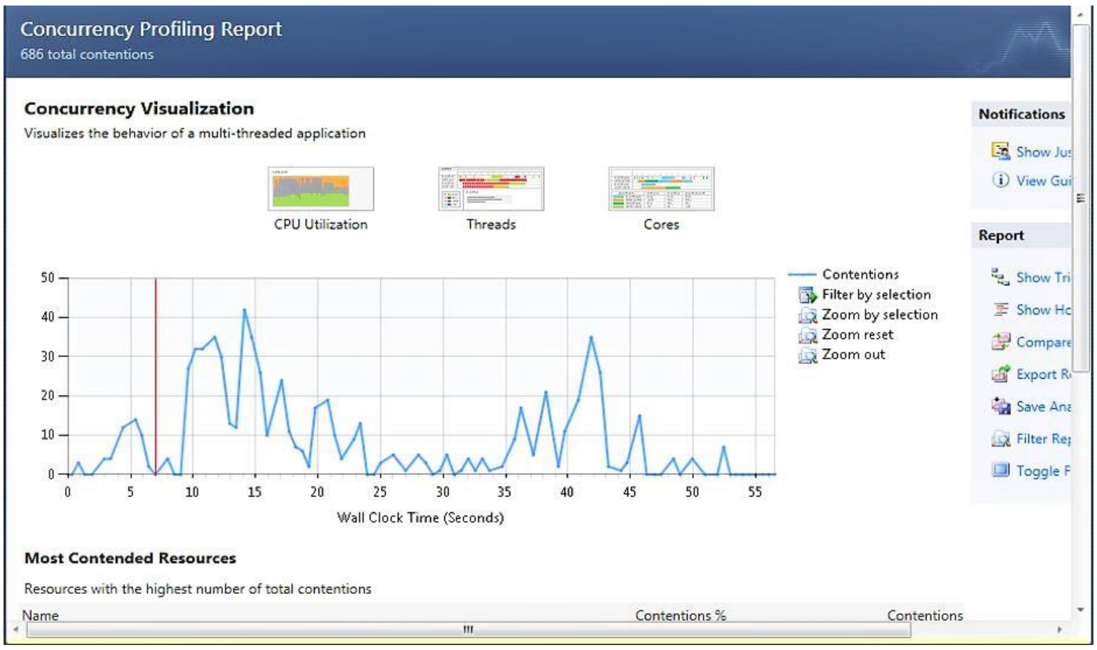


Figure 9. Visualization of resource conflicts (concurrency) at processor resources.

The processor-level study refers to the distribution of competitive resources for threads and their use percentages.

The concurrency viewer categorizes context-related events by examining the stack of thread settings for delaying APIs. If there is no stack match-up, the wait method provided by Windows should be used. Thus, this evaluation system presents the case of the moments of contention that occured during parallel execution in the scenario of blocking, decoupling, and thread synchronization.

Figure 10 shows a simple example of normal CPU usage for a generation test. This test took place consecutively for 50 s. The usage percentage of the processor is presented and, in comparison to Figure 6, a much higher load can be observed. The analysis was performed with the Visual Studio environment. For 25 s, different sequential generations were performed, and, after 25 s, the context was changed and parallel generations were performed.

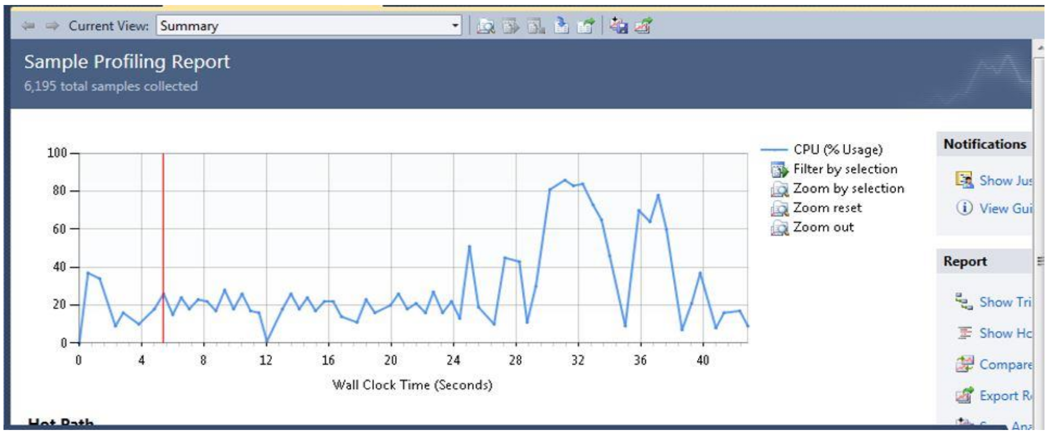
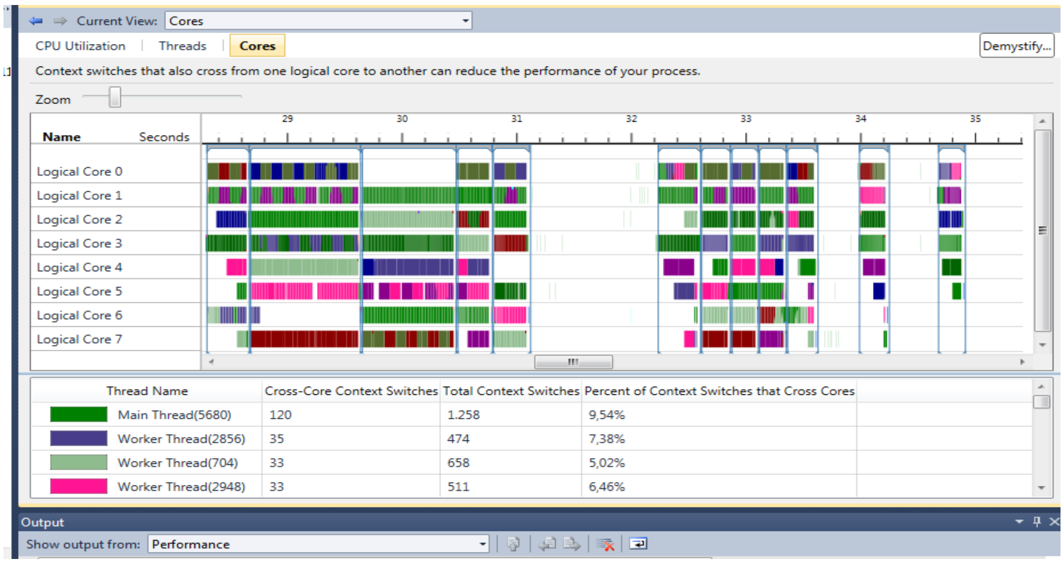


Figure 10. CPU-level analysis of two execution scenarios.

Figure 11 shows the usage of all eight processor-level resources when performing a parallel image generation. The developed application does not continuously generate fractal sets but can change the work scenario and select the calculation method at certain time points set by the user. Within the image in Figure 11 you can see a sample with no actions performed between 1 and 32 s. However, the total moments of work are also visible in the case of establishing the generations.



Context switches that also cross from one logical core to another can reduce the performance of your process.

	Thread Name	Cross-Core Context Switches	Total Context Switches	Percent of Context Switches that Cross Cores
🟩	Main Thread(5680)	120	1.258	9,54%
🟪	Worker Thread(2856)	35	474	7,38%
🟩	Worker Thread(704)	33	658	5,02%
🟥	Worker Thread(2948)	33	511	6,46%

Figure 11. Cores-level analysis.

Another conclusion to draw is that changing the working mode can lead to lower process performance, which is one of the deciding factors when applications use more processor cores. The application is well-optimized in terms of computing and memory resources used. In the proposed scenario, the allocation is carried out automatically by the system, with the context changes being implemented by default.

Performance evaluation for parallel computing can be measured concerning the speedup provided, with this increasing the efficiency in terms of the time required for both sequential and parallel processing [48]. The parameters used to calculate the speedup factor, or how much a parallel algorithm is faster than a corresponding sequential algorithm, are as follows:

$$Speedup = (sequential\ algorithm\ time)/(parallel\ algorithm\ time) \tag{6}$$

The main objective of the developed application is the optimal use of the available computing resources. Considering that the evaluation was performed on a system using an Intel I7 Q720 8-core processor, the maximum utilization of computing power was one of the main goals and can lead to lower computation times.

The test result for the parallel execution is shown in Figure 12. Arguably, one of the most complex aspects of the process is the synchronization of tasks, with this having an effect on resources.

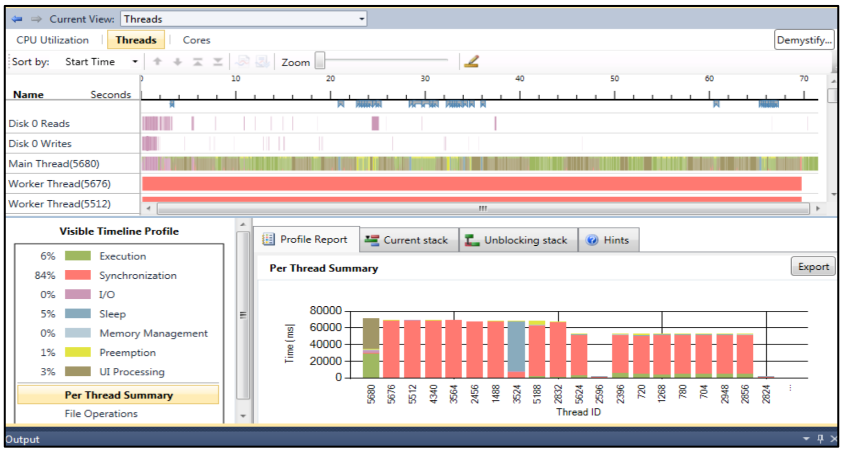


Figure 12. Allocation of the resources and the used threads for a usual work scenario.

The most important aspect of the process is to synchronize resources for the set computation algorithm and avoid resource conflicts. In the scenario shown in Figure 12, 84% of the time was allocated to synchronization, 6% to the actual execution, and only 5% to waiting times. The total run analyzed was 80 s, during which different parameters for the algorithm were applied, leading to different generated images.

It can be said that Figure 13 provides a clear illustration of the difference between sequential vs. parallel calculation. During the 76 s of analysis, one can quickly observe the change in calculation context, between a slow calculation variant, which for the sequential algorithm was between seconds 22 and 30 or 42 and 50, and the parallel one, which was between seconds 34–26 and 54–57.

The figure also shows aspects of other processes, system processes, rest, or generation due to the developed application. Application focus and goal achievement are prioritized by using eight working cores as opposed to using a single static core. The difference between the calculation times is very clear, and from the whole mix of uses or pauses, only 11% of the processor resources are used in this scenario, though it must be specified that the system does not generate automatically but on manual command.

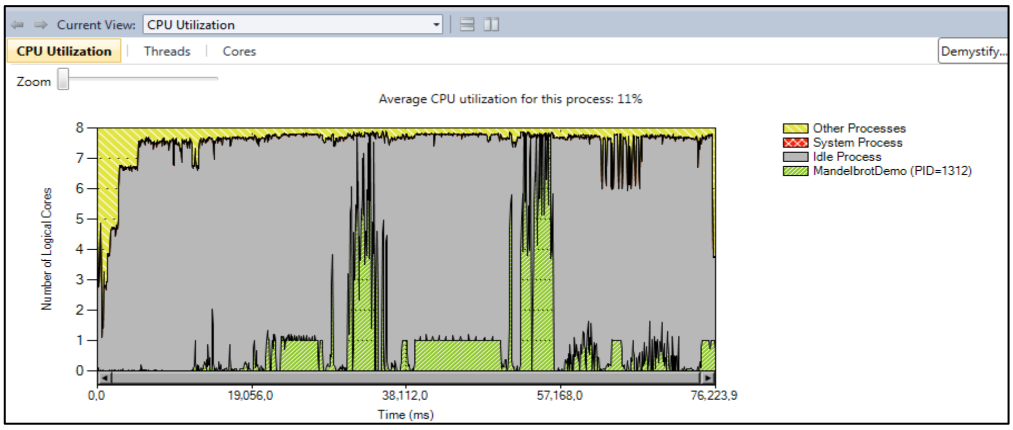
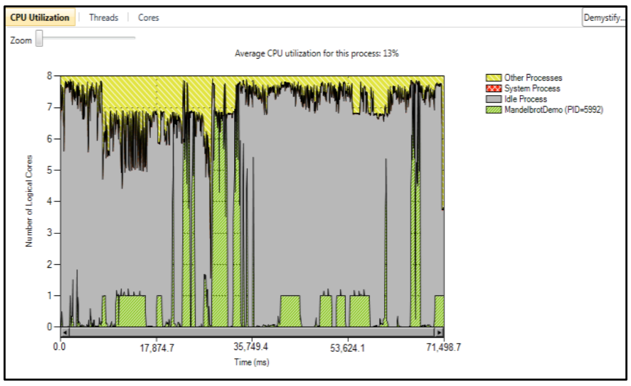
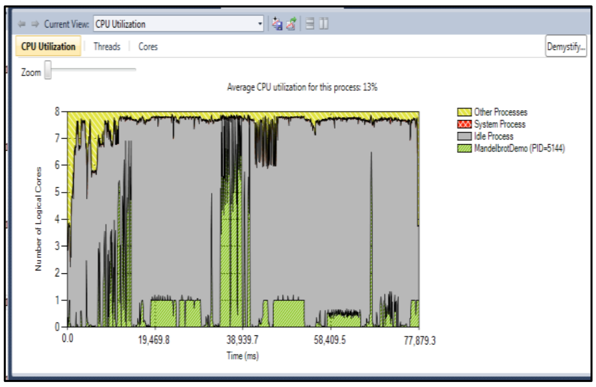


Figure 13. This is a figure. Schemes follow the same formatting.

Figure 14 is presented in the same context as Figure 13 to highlight the results obtained at the level of resource use and management. It should be noted that the time for performing the calculation is strongly influenced by factors such as the size and quality of the image. Thus, the result obtained is correlated with the number of pixels of the image, which indirectly represents the number of iterations required for processing.



(**a**)



(**b**)

Figure 14. Scenarios for CPU usage during the application: (**a**) The description over 71 s with a variety of changes in the sequential work context vs. parallel; (**b**) the description for 78 s with a variety of changes in the sequential work context vs. parallel cases scenarios.

Mandelbrot sets differ from the Julia set. If we apply the algorithm to generate Julia-type fractals, then the code sequences multiply proportionally to the shape of the chosen defining fraction. This will lead to an even greater number of lines of code multiplied for each iteration.

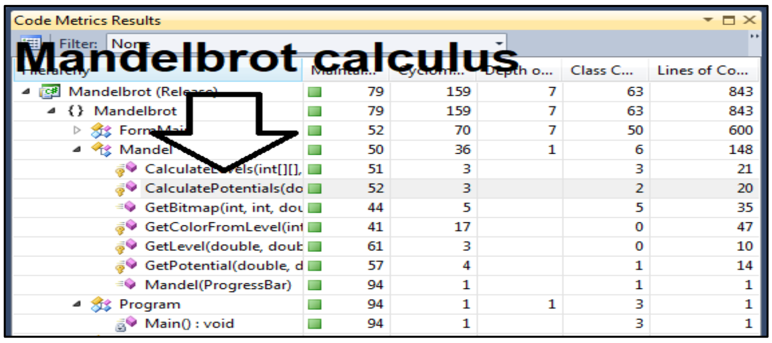
Older processing models with single-core architecture offer only one possibility in terms of sequential computation. In the context of the evolution of distributed systems, new architectural models offer modern approaches. Thus, many algorithms currently used in a multitude of fields must be readapted to modern computing means. Practically, at present, sequential fractal generation no longer has a purpose. In the proposed scenario, the two possible generation models are highlighted, as are the adaptability of certain mathematical models in the context of the new computing technologies. Even in industrial systems or those in the software industry, the computing time factor is currently the central point of concern for any effective application. Resource utilization is also an aspect to consider.

The current methods of reducing the effective computation time for fractal generation fall into two classes:

- The efficiency of the mathematical work package, the identification of repetitive models, and the means of stopping the iteration before the limit through complex calculations are all presented in [49]. Fractal tools combine programming with a basic set of mathematical experiments. The points of attraction were also identified within the analyzes performed on the topology of the image in [50]. This can lead to the identification of redundant rendering patterns, reducing the computation time.
- Another approach can be the parallel processing of the Mandelbrot set [18], which provides an insight into parallel implementation using message-passing interface (MPI) technology.

The complexity of the Julia mathematical type can produce an exponential evolution in terms of the running time. The diverse nature of fractal images, also outlined in chaos theory, leads to different results for scenarios that are mathematically similar.

Figure 15 presents the overall result of the implemented code metrics. It can be seen that the weight of the Mandelbrot algorithm at the point of code lies within a weight of roughly 4%, with most of the instructions suggesting the image generation section and the selection from the selected color palette.



Hierarchy	Maintai...	Cyclom...	Depth o...	Class C...	Lines of Co...
Mandelbrot (Release)	🟩 79	159	7	63	843
{} Mandelbrot	🟩 79	159	7	63	843
FormMai	🟩 52	70	7	50	600
Mandel	🟩 50	36	1	6	148
CalculateLevels(int[][],	🟩 51	3		3	21
CalculatePotentials(do	🟩 52	3		2	20
GetBitmap(int, int, dou	🟩 44	5		5	35
GetColorFromLevel(int	🟩 41	17		0	47
GetLevel(double, doub	🟩 61	3		0	10
GetPotential(double, d	🟩 57	4		1	14
Mandel(ProgressBar)	🟩 94	1		1	1
Program	🟩 94	1	1	3	1
Main() : void	🟩 94	1		3	1

Figure 15. Code metric results for the Mandelbrot calculus.

In Figure 16, two images obtained after consecutive renderings are presented, and it can be said that the generation of fractal images is infinite and generates distinct images no matter how refined the starting point is. Thus, a series of similar patterns can be observed that are repeated recurrently within the images. The detection of repetitive patterns can be a way of reducing operations and optimizing calculations.

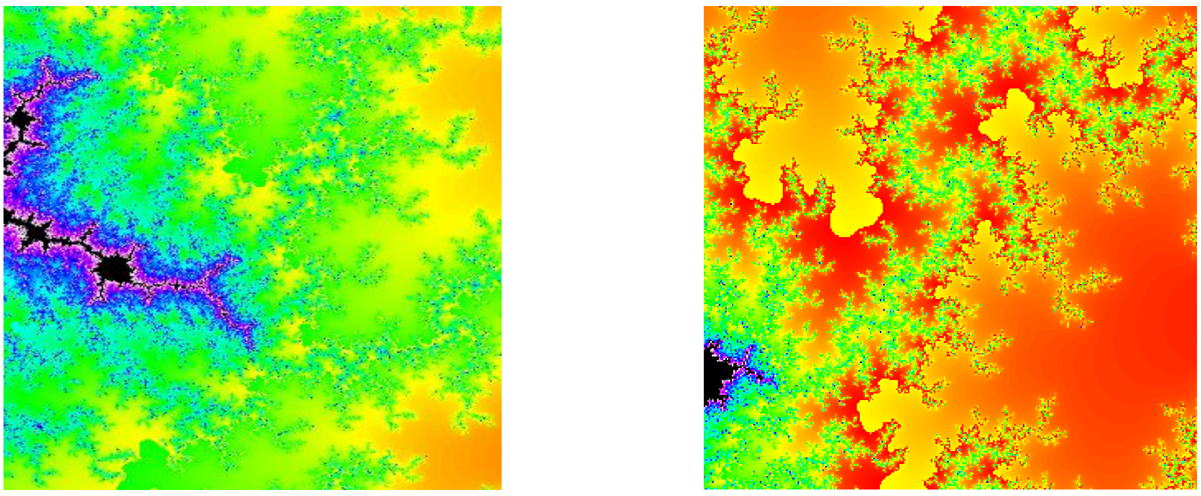

(**a**) (**b**)

Figure 16. Example of Mandelbrot images generated by the developed tool. (**a**) Primary image for Mandelbrot generation with out-of-set black areas. (**b**) Focused image for the image in a specific Mandelbrot model.

In the research activity, a PC was used that has the Intel I7 Q720 processor at 1.60 GHz with 4 Gb of RAM with the 64-bit system using Microsoft. Another possible research scenario at the extension level of the presented activity could consist of the comparative analysis of algorithms on two distinct models of computing systems, such as the one presented in [51] for another image processing algorithm.

Compared to the example in Table 1, the parallel computation time is significantly shorter. Practically, the proposed implementation is much more efficient, as the number of iterations in the loop increases. Loop-specific numbers of iterations are a factor, leading to higher computation times. By this, it can be said that it is imperative to use parallel computing in these cases. The number of iterations set to stop the loop in this test were 256, 500, and 1000, respectively, but this can be preset by the assigned parameter. The time difference becomes much more pronounced as the number of iterations or the size of the image increases. Thus, the application developed in parallel demonstrates its efficiency.

Table 1. Table with the difference in execution times between the classic sequential implementations and the parallel implementations proposed in the paper.

Results as Time Analysis	**Time**		
	256 Iterations	**500 Iterations**	**1000 Iterations**
Sequential	0.583 s	1.062 s	1.992 s
Static parallel	0.172 s	0.341 s	0.535 s

A better approach to consider involves a network made up of the many processes involved in the calculation process, which, at the same time, can be implemented through the MPI (message-passing interface) method used for this type of test. Additionally, the number of cores and the capacity of the processor or the synchronization system are important in this case. This approach to testing the algorithm, specifically in a cluster computing system, was proposed in [11].

There may also be exceptional cases with many processes and calculations that last for days; this requires a very advanced system in terms of computing power to solve these challenges. The following graph, presented in Figure 17, shows values for the calculation times obtained in the context of images with increasing dimensions. The form of the Julia-type function is also presented in [11].

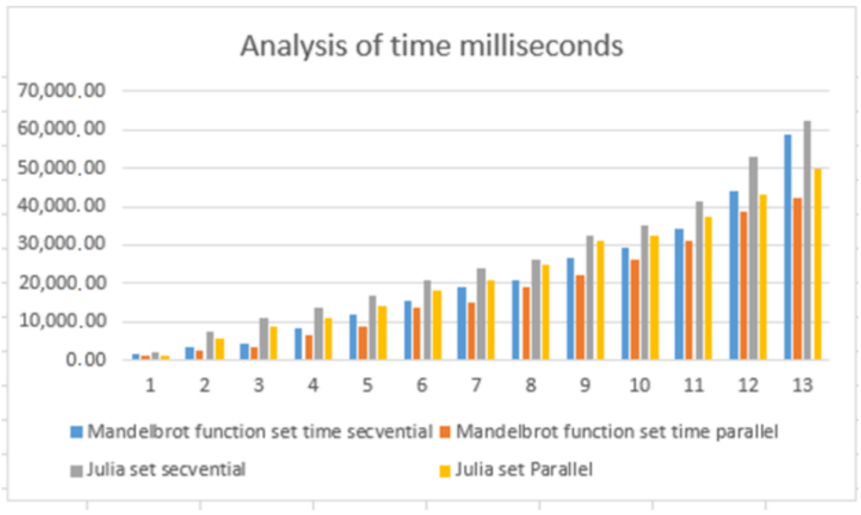


Figure 17. Calculation times were obtained for images of increasing size. Comparative study of Mandelbrot vs. Julia [11].

6. Discussion

In the presented article, additional contributions were made to the works [2,3,10–12,16]. Thus, several applications have been described, implemented, and tested, namely:

(1) Analyses and comparisons of sequential fractal generation algorithms for Mandelbrot and Julia sets.
(2) The parallelization algorithm was proposed in subchapter 4.1 and is valid for both variants of fractal sets.
(3) Applications were created for the generation and analysis of fractal images.
(4) A CPU-level report was presented in the subchapter, from which the contribution of optimization at the level of parallel execution was presented.
(5) Several execution time measurements were generated which illustrate the differences in terms of calculation speed.

The proposals are beneficial for the development of applications that use visualization systems for generating fractals. These results will be used in future research studies. They can be associated with a set of complex images as input data for other applications or for different systems for framing these images and classifying them. Moreover, these proposals can represent a test base for different processors and more complex images, which is required in practical cases in the field of engineering.

It should be mentioned that the developed application uses a portable programming language that can be installed and used on any platform available today (C#) in various environments for image processing. Whether we are talking about image generation, identification, or processing, this application has good results in terms of computing time and is acceptable for real-time systems in this field. The application utilizes the processing capabilities of the medium on which it runs. The application can also be used by inexperienced users and can easily generate complex images.

From the point of view of future work, a series of distinct directions could be taken, such as:

- The further development of options for algorithms that generate fractal images may contain parallel programming techniques such as semaphores, dynamic priorities, monitors, barriers, message queues, and pipes. There are also modern means which exemplify parallel computing on fractals, such as the Momentics tool and the Software Development Platform for QNX Neutrino. One of the more popular reply-driven programs is a fractal graphics program distributed over a network [52].
- It should also be mentioned that the combination of mathematical fractal generation with natural images is a field of research with multidisciplinary applicability. Considering the MOSCBIOS project [53], a series of dimensions of certain natural factors on various types of fruits were discovered. The method developed and described in [10] was constructed in the C# language and is easily combined with the fractal generation tools presented in the current article. Taking into account the continuous evolution of software in the field of image and signal processing, the methods described in this article could be developed further. Such proposals could be:
    - An online library for fractal visualization and the identification of natural images using fractal means based on MOSCBIOS [53] project results.
    - The development of an image compression tool for fractal generation, such as in the context of input images.
    - A real-time natural image detection system centered on fractal analysis.
- Another variant of the subsequent approach is the use of images generated with the means described in this paper as input data for image compression or decompression applications.

Fractal images are effective test tools because they have an unpredictability component and model-based compression techniques cannot be applied to them. An example can be dyadic compression or decompression, as shown in Figure 18.

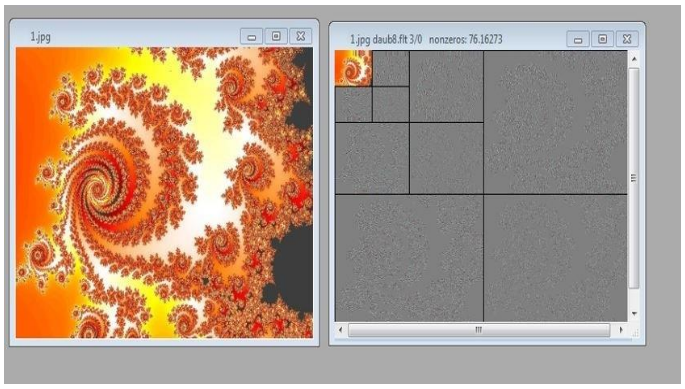


Figure 18. Using fractal results in applications for the compression or decompression of images.

One comparative approach that can be taken is between the programming languages chosen for the development of fractal sets. From the point of view of the proposed parallelization strategy, there are fundamental differences between the chosen language, C#, and Python. Python is a popular language in the context of image processing applications, but it has drawbacks from the point of view of real-time parallelization applications.

First, the Python language uses an interpreter that converts the script into C language. Furthermore, the interpreter can access only one processor resource at a time, making it redundant. When running Python scripts, a single processor core can be accessed through the interpreter, in the case of distributing the proposed calculation, the optimization, execution time, and efficiency can not reach the standards that the C# language and the project provide.

On the other hand, C# provides support regarding distributed processing at the resource level, in addition to providing several mechanisms implemented and optimized to access Windows functions. It also provides the possibility of distributed access at the base level. For parallelization applications, the Python language cannot yet provide support and involves many drawbacks.

7. Conclusions

Fractal theory represents a vast field of analysis. The results of this article can be described in terms of testing and making comparisons between applications that generate fractals sets, such as the Julia and Mandelbrot sets, and the presentation of parallel computing modalities.

The past four decades have seen significant progress in understanding how to analyze irregular shapes and structures in the physical and biological sciences, starting with the exploration of fractals and their modeling. Different types of images have been discovered that cannot obtained by other methods and can be compared, in terms of complexity, only with natural formations.

The work was founded on a question of mathematical competence and has gone on to explore to sequential development, the parallel approach, and efficient methods.

This paper has presented an novel way to design different sets of fractals. Different methods were considered, starting with the contribution of parallel programming in contrast to sequential programming. The aim was to highlight modern methods of speeding up the computation time. The developed applications have applicability in various fields.

Contributions were made considering the developed fractal generation application, testing, and comparing sequential and parallel algorithms for Mandelbrot generation. Report presentations were made in the context of processor evolution and to illustrate the differences in the type of computation and resources used. All this led to the creation of an efficient model for generating fractal images in short times using regular means and the resources available to us today.

In conclusion, the present paper has presented the necessary optimization of the Mandelbrot algorithm using parallel programming techniques in addition to an analysis of the implemented system.

Author Contributions: Conceptualization, B.P. and D.S.; formal analysis, B.S, D.S., and A.E.L.; investigation, B.P.; methodology, B.P.; project administration, D.S.; software, B.P and A.E.L.; supervision, D.S.; validation, B.P. and A.E.L.; visualization, B.P.; writing—original draft, B.P.; writing—review and editing, B.P. and D.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the grant POCU/380/6/13/123990, co-financed by the European Social Fund within the Sectorial Operational Program Human Capital 2014–2020.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mandelbrot, B. *The Fractal Geometry of Nature*; WH Freeman: New York, NY, USA, 1982; Volume 1.
2. Popa, B.; Ionete, C.; Lorincz, A.E.; Bădulescu, L.A. Possibilities for fast Generation of Fractal Images and Various Fields of Applicability. In Proceedings of the 23rd International Carpathian Control Conference (ICCC), Sinaia, Romania, 29 May–1 June 2022; pp. 82–87. [CrossRef]
3. Popa, B.; Popescu, D.; Roman, M.; Constantinescu, R.L. Optimizing Algorithms for low CPU Usage in Different Scenarios. In Proceedings of the 20th International Carpathian Control Conference (ICCC), Krakow-Wieliczka, Poland, 26–29 May 2019; pp. 1–6. [CrossRef]
4. Song, X.; Yang, L. The Study of Adaptive Multi Threshold Segmentation Method for Apple Fruit Based on the Fractal Characteristics. In Proceedings of the 2015 8th International Symposium on Computational Intelligence and Design (ISCID), Hangzhou, China, 12–13 December 2015; pp. 168–171. [CrossRef]
5. Jiang, W.; Ji, C.; Zhu, H. Fractal Study on Plant Classification and Identification. In Proceedings of the International Workshop on Chaos-Fractals Theories and Applications, Shenyang, China, 6–8 November 2009; pp. 434–438. [CrossRef]
6. Jahanmiri, F.; Parker, D.C. An Overview of Fractal Geometry Applied to Urban Planning. *Land* **2022**, *11*, 475. [CrossRef]
7. Tahmineh, A. On the fractal geometry of gait dynamics in different neuro-degenerative diseases. *Phys. Med.* **2022**, *14*, 100050.
8. Frankhauser, P.; Pumain, D. Fractals and geography. In *Machine Learning and the City: Applications in Architecture and Urban Design*; Wiley: Hoboken, NJ, USA, 2022; pp. 31–55.
9. dos Anjos, P.H.; Gomes-Filho, M.S.; Alves, W.S.; Azevedo, D.L.; Oliveira, F.A. The fractal geometry of growth: Fluctuation-dissipation theorem and hidden symmetry. *arXiv* **2022**, arXiv:2203.04461. [CrossRef]
10. Bogdan, P. Iterative function systems for natural image processing. In Proceedings of the 2015 16th International Carpathian Control Conference (ICCC), Szilvasvarad, Hungary, 27–30 May 2015; pp. 46–49. [CrossRef]
11. Popa, B. Visual Study About the Fractals and New Means of Viewing. *Int. J. Sci. Res.* **2016**, *5*, 383–386.
12. Popa, B.; Roman, M.; Petre, E.; Cosmulescu, S.; Stoenescu, A.-M. Software Tools to Manage and Simulate Information from the Natural Environment. In Proceedings of the 21st International Carpathian Control Conference (ICCC), High Tatras, Slovakia, 27–29 October 2020; pp. 1–6. [CrossRef]
13. Husain, A.; Reddy, J.; Bisht, D.; Sajid, M. Fractal dimension of India using multicore parallel processing. *Comput. Geosci.* **2022**, *159*, 104989. [CrossRef]
14. Francisco, H.L.J.; Pérez, O.M. Parallel fractal image compression using quadtree partition with task and dynamic parallelism. *J. Real-Time Image Process.* **2022**, *19*, 391–402.
15. Ranjita, A.; Raghuwanshi, M.M.; Singh, K.R. Fractal Image Coding-Based Image Compression Using Multithreaded Parallelization. In Proceedings of the Information and Communication Technology for Competitive Strategies (ICTCS 2021), Jaipur, India, 17–18 December 2021; Springer: Singapore, 2023; pp. 559–569.
16. Popa, B.; Popescu, D. Lossless Compression in Image Processing Technologies and Applications. *Ann. Univ. Craiova* **2014**, *11*, 13–18.
17. Zhang, Z.-W.; Jing, X.-Y.; Wang, T.-J. Label propagation based semi-supervised learning for software defect prediction. *Autom. Softw. Eng.* **2016**, *24*, 47–69. [CrossRef]
18. Nam, J.; Fu, W.; Kim, S. Heterogeneous defect prediction. *IEEE Trans. Softw. Eng.* **2018**, *44*, 874–896. [CrossRef]
19. Matloff, N. *Programming on Parallel Machines*; University of California: Davis, CA, USA, 2011; p. 39319.
20. Husain, A.; Nanda, M.N.; Chowdary, M.S.; Sajid, M. Fractals: An Eclectic Survey, Part-I. *Fractal Fract.* **2022**, *6*, 89. [CrossRef]
21. Rückert, J.; Dortmund, F.H. Artificial Art: Image Generation using Evolutionary Algorithms. *Schr. Fachbereichs Inform. Fachhochsch. Dortm.* **2013**, *2*, 175–190.
22. Tregubova, I.A.; Sobko, K.O.; Gokhman, R.O. Fractal Graphics as the Modern Technology of Science. Цифрові Технології **2018**, *24*, 105–111.

23. Asaduzzaman, A.; Trent, A.; Osborne, S.; Aldershof, C.; Sibai, F.N. Impact of CUDA and OpenCL on Parallel and Distributed Computing. In Proceedings of the 2021 8th International Conference on Electrical and Electronics Engineering (ICEEE), Antalya, Turkey, 9–11 April 2021; pp. 238–242.
24. Helbecque, G.; Gmys, J.; Carneiro, T.; Melab, N.; Bouvry, P. A performance-oriented comparative study of the Chapel high-productivity language to conventional programming environments. In Proceedings of the Thirteenth International Workshop on Programming Models and Applications for Multicores and Manycores, Seoul, Korea, 2–6 April 2022; pp. 21–29.
25. Gómez, E.S. MPI vs. OpenMP: A case study on parallel generation of Mandelbrot set. *Innovación Softw.* **2020**, *1*, 12–26.
26. Tracolli, M. Parallel Generation of a Mandelbrot Set. VIRT&L-COMM. Available online: http://services.chm.unipg.it/ojs/index.php/virtlcomm/article/view/112 (accessed on 1 November 2022).
27. Hungilo, G.G.; Emmanuel, G.; Pranow. Performance comparison in simulation of Mandelbrot set fractals using Numba. *AIP Conf. Proc.* **2020**, *2217*, 030007.
28. Sallow, A.B. Implementation and Analysis of Fractals Shapes using GPU-CUDA Model. *Acad. J. Nawroz Univ.* **2021**, *10*, 1–10. [CrossRef]
29. Rockenbach, D.A.; Stein, C.M.; Griebler, D.; Mencagli, G.; Torquati, M.; Danelutto, M.; Fernandes, L.G. Stream processing on multi-cores with GPUs: Parallel programming models' challenges. In Proceedings of the 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Rio de Janeiro, Brazil, 20–24 May 2019; pp. 834–841.
30. Kaboudian, A.; Cherry, E.M.; Fenton, F.H. Large-scale interactive numerical experiments of chaos, solitons and fractals in real time via GPU in a web browser. *Chaos Solitons Fractals* **2019**, *121*, 6–29. [CrossRef]
31. Guzev, V.; Serdyuk, Y. Asynchronous parallel programming language based on the Microsoft. NET platform. In Proceedings of the International Conference on Parallel Computing Technologies, Nizhni Novgorod, Russia, 15–19 September 2003; Springer: Berlin/Heidelberg, Germany, 2003; pp. 236–243.
32. Bassil, Y. Implementation of Computational Algorithms using Parallel Programming. *Int. J. Trend Sci. Res. Dev.* **2019**, *3*, 704–710. [CrossRef]
33. Guzev, V.; Serdyuk, Y.; Chudinov, A.; Strategy, L.L.C. MC#: Asynchronous parallel programming language for cluster-and GRID-architectures. In Proceedings of the International Workshop on C# and .NET Technologies on Algorithms, Computer Graphics, Visualization, Computer Vision and Distributed Computing, Plzen, Czech Republic, 6–8 February 2003.
34. Heffner, K.; Tarditi, D.; Smith, M.D. Extending object-oriented optimizations for concurrent programs. In Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007), Brasov, Romania, 15–19 September 2007; pp. 119–129.
35. Douady, A.; Hubbard, J.H. Etude dynamique des polynômes complexes. *Prépublications Mathémathiques d'Orsay* **1985**, *2*, 1984.
36. Barral, J. Moments, continuité, et analyse multifractale des martingales de Mandelbrot. *Probab. Theory Relat. Fields* **1999**, *113*, 535–569. [CrossRef]
37. Grabner, P.J. Poincaré Functional equations, harmonic measures on Julia sets, and fractal zeta functions. In *Fractal Geometry and Stochastics V*; Birkhäuser: Cham, Switzerland, 2015; pp. 157–174.
38. Taylor, A.R.P.; Sprott, J.C. Biophilic Fractals and the Visual Journey of Organic Screen-savers. *Nonlinear Dyn. Psychol. Life Sci.* **2008**, *12*, 117–129.
39. Devaney, R.L.; Linda, K. *Chaos and Fractals, Proceedings of Symposia in Applied Mathematics*; American Mathematical Society: Providence, RI, USA, 1989; Volume 39, p. 64.
40. Jovanovic, P.; Tuba, M.; Simian, D.; Romania, S.S. A new visualization algorithm for the Mandelbrot set. In Proceedings of the 10th WSEAS International Conference on Mathematics and Computers in Biology and Chemistry, Prague, Czech Republic, 23–25 March 2009; World Scientific and Engineering Academy and Society (WSEAS): Stevens Point, WI, USA, 2009.
41. Mamta, R.; Kumar, V. Superior Julia set. *Res. Math. Educ.* **2004**, *8*, 261–277.
42. Özgür, N.; Swati, A.; Tomar, A. Julia and Mandelbrot Sets of Transcendental Function via Fibonacci-Mann Iteration. *J. Funct. Spaces* **2022**, *2022*, 2592573. [CrossRef]
43. Gottlieb, A.; Almasi, G.S. *Highly Parallel Computing*; Benjamin/Cummings: Redwood City, CA, USA, 1989; ISBN 0-8053-0177-1.
44. Systems with Propagation Project. Available online: http://dae.ucv.ro/cercetare/proiecte.php (accessed on 1 October 2022).
45. Hejlsberg, A.; Torgersen, M.; Wiltamuth, S.; Golde, P. *C# Programming Language (Covering C# 4.0)*, 4th ed.; McGraw Hill Professional: Boston, MA, USA, 2010.
46. *ECMA-334*; C# Language Specification. 4th ed. ECMA International: Geneva, Switzerland, 2006; Retrieved 26 January 2012. Available online: https://www.ecma-international.org/wp-content/uploads/ECMA-334_4th_edition_june_2006.pdf (accessed on 1 October 2022).
47. Microsoft Docs. Available online: https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.stopwatch?view=net-6.0 (accessed on 1 October 2022).
48. Haron, N.; Ami, R.; Aziz, I.A.; Jung, L.T.; Shukri, S.R. Parallelization of Edge Detection Algorithm using MPI on Beowulf Cluster. In *Innovations in Computing Sciences and Software Engineering*; Springer: Dordrecht, The Netherlands, 2010; pp. 477–482.
49. Hosseini, S.A.; Shookooh, B.R.; Shahhosseini, S.; Beizaee, S. Speeding up fractal image de-compression. In Proceedings of the International Conference on Computer Applications and Industrial Electronics, Kuala Lumpur, Malaysia, 5–8 December 2010; pp. 521–526. [CrossRef]

50. Tao, S.; Hao, T.M.; Yang, Z.Z. Research on high-periodic attracting points in Mandelbrot set. In Proceedings of the 2011 International Conference on Computer Science and Network Technology, Harbin, China, 24–26 December 2011; pp. 2038–2041. [CrossRef]
51. Popa, B. Study about the edge detection algorithm and its applications. In Proceedings of the 18th International Carpathian Control Conference (ICCC), Sinaia, Romania, 28–31 May 2017.
52. Krten, R. *Getting Started with QNX Neutrino 2: A Guide for Realtime Programmers*; PARSE Software Devices: Ottawa, ON, Canada, 1999.
53. MOSCBIOS Project. Available online: http://dae.ucv.ro/cercetare/proiecte.php (accessed on 1 October 2022).