A Project Report

*On*

Comparing Mandelbrot generation times

*By*

Mayank Baber (BC201)

Mayuresh Chougule (BC211)

Pranav Deshpande (BC216)

Rachna Ganjoo (BC219)

*Under the guidance of*

Prof. Snehal Kuche



**Department of Computer Engineering Marathwada Mitra Mandal’s College of Engineering**

**SAVITRIBAI PHULE PUNE UNIVERSITY 2023-2024**

Marathwada Mitra Mandal’s College of Engineering Department of Computer Engineering, Pune\_51



Date:

# CERTIFICATE

This is to certify that,

Mayank Baber (BC201)

Mayuresh Chougule (BC211)

Pranav Deshpande (BC216)

Rachna Ganjoo (BC219)

of class, B.E. Computer have successfully completed their project work on “Comparing Mandelbrot generation times” at MARATHWADA MITRA MANDAL'S COLLEGE OF ENGINEERING in the partial fulfillment of the Graduate Degree course in B.E. Deep Learning at the Department of Computer Engineering, in the academic Year 2023-2024 Semester – II as prescribed by the Savitribai Phule Pune University.

Prof. Snehal Kuche Dr. K.S. Thakre

Guide Head of the Department

(Department of Computer Engineering)

# Acknowledgement

I feel great pleasure in expressing my deepest sense of gratitude and sincere thanks to my guide Prof. Kalyani Ghuge for their valuable guidance during the Project work, without which it would have been a very difficult task. I have no words to express my sincere thanks for the valuable guidance, extreme assistance and cooperation extended to all the **Staff Members** of my Department.

This acknowledgement would be incomplete without expressing my special thanks to **Dr. K.S. Thakre**, Head of the Department (Computer Engineering) for their support during the work.

I would also like to extend my heartfelt gratitude to my **Principal, Dr. V.N. Gohokar** who provided a lot of valuable support, mostly being behind the veils of college bureaucracy.

Last but not least I would like to thank all the Teaching, Non- Teaching staff members of my department, my parent and my colleagues those who helped me directly or indirectly in completing this Project successfully.

Mayank Baber (BC201)

Mayuresh Chougule (BC211)

Pranav Deshpande (BC216)

Rachna Ganjoo (BC219)

**Contents**

1. **TITLE OF THE PROJECT**
2. **ABSTRACT**
3. **INTRODUCTION**
   * Problem Definition
4. **ALGORITHM (TYPE OF ALGORITHM)**
5. **PERFORMANCE METRICS**
6. **RESULTS**
7. **CONCLUSION**
8. **REFERENCES**

**TITLE OF THE PROJECT**

Comparing the Mandelbrot Fractals generation times

**ABSTRACT:**

This project investigates the efficiency of Mandelbrot set generation using CPU and GPU architectures. Through implementation and comparison of the Mandelbrot algorithm on both platforms, it evaluates execution times and visual outcomes. The study aims to quantify the performance disparity between CPU and GPU processing for complex mathematical computations like the Mandelbrot set. Results are presented through execution time metrics and visual representations, highlighting the computational advantages of GPU parallelism. The project contributes insights into optimizing computational tasks by leveraging GPU acceleration, particularly for iterative and compute-intensive algorithms like the Mandelbrot set. These findings have implications for scientific computing and high-performance computing applications, showcasing the potential for accelerated calculations in graphical and scientific domains.

**INTRODUCTION:**

The Mandelbrot set, a captivating fractal pattern in the complex plane, is generated through iterative calculations based on the Mandelbrot algorithm. This project delves into comparing the computational efficiency of Mandelbrot set generation on CPU and GPU architectures. The significance lies in understanding how parallel processing on GPUs can accelerate complex mathematical computations. By implementing the Mandelbrot algorithm on both platforms, this study aims to quantify the performance gains achievable through GPU parallelism. The introduction of GPU acceleration in scientific computing has revolutionized the speed and scalability of computations, making it a focal point for optimizing algorithms like the Mandelbrot set calculation. Through this exploration, we delve into the intersection of computational mathematics and hardware acceleration, shedding light on advancements in high-performance computing paradigms.

**PROBLEM DEFINITION:**

The problem addressed in this project is the comparative analysis of Mandelbrot set generation efficiency between CPU and GPU architectures. Specifically, it aims to quantify the speedup achieved by utilizing GPU parallelism for iterative computations like those in the Mandelbrot algorithm. This investigation is motivated by the increasing demand for faster and more scalable computational methods in scientific and graphical domains. Understanding the performance disparity between CPU and GPU implementations for the Mandelbrot set can guide optimal resource utilization and algorithm design. The problem also encompasses evaluating the trade-offs between computational power, memory access, and parallel processing capabilities inherent in CPU and GPU architectures. Ultimately, this study seeks to elucidate the advantages and challenges associated with leveraging GPU acceleration for complex mathematical computations.

**ALGORITHM:**

The algorithm for generating the Mandelbrot set involves iterating through a grid of complex numbers in the specified complex plane region. Initially, the image size and grid dimensions are defined, allocating memory on the GPU for the image. The Mandelbrot algorithm on the GPU is implemented using CUDA, with a kernel function mandel\_kernel that computes the Mandelbrot set for each pixel in parallel. The kernel function calculates the real and imaginary components of the complex numbers within the grid, iterating through each pixel and determining the escape time or iteration count required for each complex number to diverge. This iterative process is performed using a nested loop structure that calculates the Mandelbrot set's escape time for each pixel based on its coordinates in the complex plane.

Additionally, the algorithm utilizes CUDA's grid and block dimensions to organize parallel threads for efficient computation. The mandel\_kernel function is designed to exploit GPU parallelism, with each thread responsible for computing the Mandelbrot set for a specific pixel. This parallel processing approach significantly accelerates the computation of the Mandelbrot set compared to traditional CPU-based methods, leveraging the GPU's ability to handle multiple calculations simultaneously. The resulting image data is then copied back from the GPU to the host for visualization and analysis, providing insights into the performance benefits of GPU acceleration for complex mathematical computations like the Mandelbrot set generation.

Three approaches have been covered in the project for comparison:

1. **Normal Sequential Approach:** The normal sequential approach utilizes a straightforward implementation of the Mandelbrot set calculation.

* Mandelbrot Function (`mandel`): This function determines whether a given complex number is a candidate for membership in the Mandelbrot set. It iterates over a fixed number of iterations and checks whether the magnitude of the resulting complex number exceeds a threshold.
* Create Fractal Function (`create\_fractal`): This function generates the Mandelbrot set image by iterating over each pixel in the image and computing the corresponding Mandelbrot set value using the `mandel` function.

1. **Numba Approach:** In this approach, the Numba library is used to accelerate the computation

* JIT Decorator: Numba's JIT (Just-In-Time) compilation is applied to both the `mandel` and `create\_fractal` functions. This compilation optimizes the execution of these functions for better performance.

1. **CUDA Approach:** This approach utilizes CUDA (Compute Unified Device Architecture) to parallelize the Mandelbrot set computation.

* CUDA Kernel (`mandel\_kernel`): A CUDA kernel function is defined to be executed on the GPU. It is responsible for computing the Mandelbrot set values for a subset of the image pixels in parallel.
* CUDA Device Function (`mandel\_gpu`): This function is called within the CUDA kernel to compute the Mandelbrot set value for a single pixel. It's similar to the `mandel` function used in the sequential approach but is designed to run on the GPU.
* Grid and Block Dimensions: The CUDA kernel is executed with a grid of blocks, and each block contains multiple threads. This configuration allows for efficient parallel computation on the GPU.

**PERFORMANCE METRICS:**

1. Sequential Approach:
   1. In the sequential approach, the computation of the Mandelbrot set is performed sequentially, with each pixel's value calculated one after the other. This method has limited parallelism, as each iteration of the set depends on the previous one. Consequently, the execution time increases linearly with the size of the image and the number of iterations.
2. Numba Approach:
   1. Utilizing the Numba library, the sequential Python code is optimized using just-in-time compilation, enhancing performance. By compiling the code into machine code at runtime, Numba reduces overhead and improves execution speed. Although this approach doesn't fully exploit parallelism, it enhances performance by optimizing the code's execution.
3. CUDA Approach:
   1. In contrast, the CUDA approach achieves significant parallelism by harnessing the power of the GPU. CUDA kernels enable thousands of threads to execute in parallel on the GPU cores. Each thread computes a pixel value independently, exploiting massive parallelism to accelerate computation. This approach offers superior performance compared to the sequential and Numba approaches, especially for large images and high iteration counts.

**RESULTS AND VISUALISATION**

**RESULTS:**

**Summary of Fractals Generation Compute time:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Summary** | **Sequential** | **Numba** | **CUDA** |
| **Compute Time** | 6.59 | 0.33 | 0.24 |

The results achieved demonstrate the significant performance improvements achieved by the Numba and CUDA approaches compared to the sequential approach:

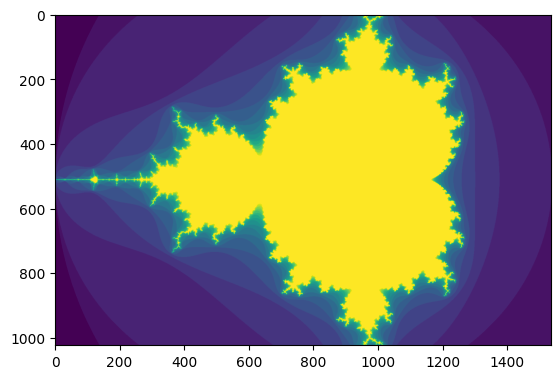
**1. Sequential Approach:** Mandelbrot set generation took approximately 6.59 seconds using the sequential approach. In this method, computation occurs sequentially, resulting in longer execution times, especially for large images or high iteration counts.

**2. Numba Approach**: With the Numba approach, the Mandelbrot set was generated in approximately 0.33 seconds, significantly faster than the sequential approach. Numba's just-in-time compilation optimizes the Python code, reducing overhead and enhancing execution speed. While not fully exploiting parallelism, this approach improves performance through code optimization.

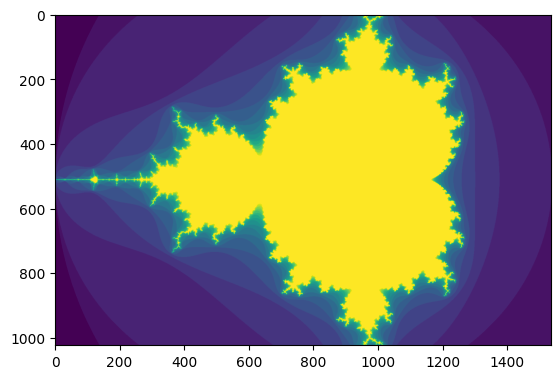
**3. CUDA Approach:** The CUDA approach achieved the fastest generation time, with the Mandelbrot set created on the GPU in only 0.24 seconds. By leveraging the massive parallelism offered by GPUs, CUDA kernels execute thousands of threads simultaneously, drastically reducing computation time. This approach is especially effective for large images and high iteration counts, as it harnesses the GPU's computational power to accelerate fractal generation.

**VISUALIZATIONS:**

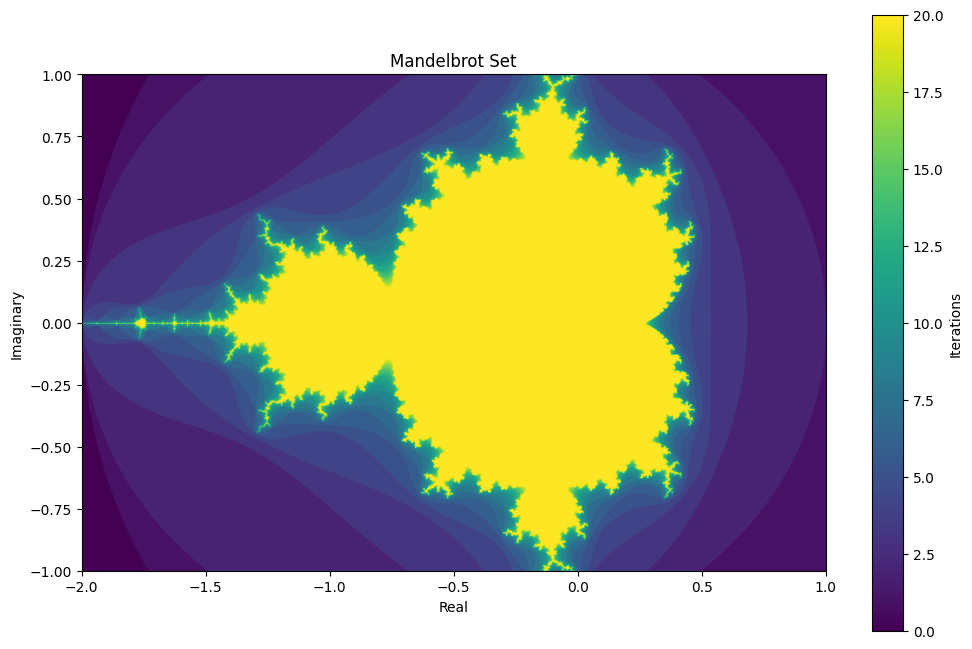
1. **Mandelbrot Fractals using the sequential approach:**



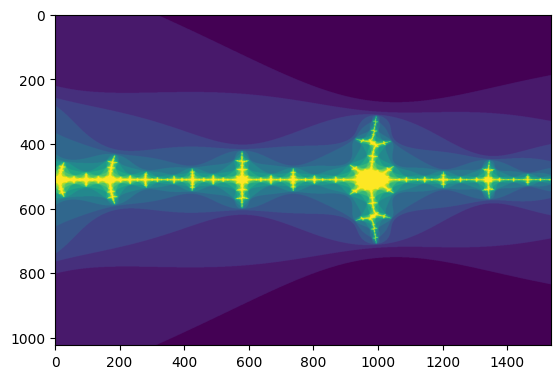
1. **Mandelbrot Fractals using the Numba approach:**



1. **Mandelbrot Fractals using the CUDA approach:**



1. **Mandelbrot Fractals archipelago representation:**



**CONCLUSION:**

The project aimed to optimize the generation of the Mandelbrot set using three different approaches: sequential, Numba, and CUDA. The sequential approach served as the baseline, where computation occurs one after the other, leading to longer execution times. On the contrary, the Numba approach utilized just-in-time compilation to optimize the Python code, resulting in faster execution by reducing overhead. However, the most significant performance gains were achieved with the CUDA approach, which leveraged the parallel processing capabilities of GPUs to execute multiple threads simultaneously.

The results demonstrated the effectiveness of parallelism in reducing computation time. The Numba approach provided noticeable improvements over the sequential method, showcasing the benefits of code optimization. Meanwhile, the CUDA approach achieved the most substantial performance enhancement, with Mandelbrot set generation times significantly reduced compared to both the sequential and Numba approaches. By harnessing the immense computational power of GPUs, the CUDA approach achieved unparalleled speed and scalability, making it ideal for generating Mandelbrot sets for large images and high iteration counts.

**REFERENCES:**

1. E. Guariglia, "Spectral analysis of the Weierstrass-Mandelbrot function," 2017 2nd International Multidisciplinary Conference on Computer and Energy Science (SpliTech), Split, Croatia, 2017, pp. 1-6.
2. Sui Tao, Tian Ming-hao and Zhang Ze-yang, "Research on high-periodic attracting points in Mandelbrot set," Proceedings of 2011 International Conference on Computer Science and Network Technology, Harbin, China, 2011, pp. 2038-2041, doi: 10.1109/ICCSNT.2011.6182371.
3. V. M. Siregar et al., "Efficient Computation of Mandelbrot Set Generation with Compute Unified Device Architecture (CUDA)," 2022 6th International Conference on Informatics and Computational Sciences (ICICoS), Semarang, Indonesia, 2022, pp. 1-5, doi: 10.1109/ICICoS56336.2022.9930601.
4. Koichi Ito, Ayako Suzuki, S. Nagashima and T. Aoki, "Performance evaluation using Mandelbrot images for image registration algorithms," 2009 16th IEEE International Conference on Image Processing (ICIP), Cairo, Egypt, 2009, pp. 4393-4396, doi: 10.1109/ICIP.2009.5413618.
5. P. Li, Z. Guan, J. Cao and F. Hang, "Research on Fractal Image Compression Algorithm Based on Generalized Mandelbrot Set," 2023 5th International Conference on Communications, Information System and Computer Engineering (CISCE), Guangzhou, China, 2023, pp. 337-340, doi: 10.1109/CISCE58541.2023.10142569.
6. A. Asaduzzaman, A. Trent, S. Osborne, C. Aldershof and F. N. Sibai, "Impact of CUDA and OpenCL on Parallel and Distributed Computing," 2021 8th International Conference on Electrical and Electronics Engineering (ICEEE), Antalya, Turkey, 2021, pp. 238-242, doi: 10.1109/ICEEE52452.2021.9415927.
7. W. F. Godoy et al., "Evaluating performance and portability of high-level programming models: Julia, Python/Numba, and Kokkos on exascale nodes," 2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), St. Petersburg, FL, USA, 2023, pp. 373-382, doi: 10.1109/IPDPSW59300.2023.00068.
8. N. Watkinson, P. Tai, A. Nicolau and A. Veidenbaum, "NumbaSummarizer: A Python Library for Simplified Vectorization Reports," 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), New Orleans, LA, USA, 2020, pp. 1-7, doi: 10.1109/IPDPSW50202.2020.00058.
9. M. Sourouri, J. Langguth, F. Spiga, S. B. Baden and X. Cai, "CPU+GPU Programming of Stencil Computations for Resource-Efficient Use of GPU Clusters," 2015 IEEE 18th International Conference on Computational Science and Engineering, Porto, Portugal, 2015, pp. 17-26, doi: 10.1109/CSE.2015.33.
10. H. Qi, M. Tanveer, M. S. Saleem and Y. Chu, "Anti Mandelbrot Sets via Jungck-M Iteration," in IEEE Access, vol. 8, pp. 194663-194675, 2020, doi: 10.1109/ACCESS.2020.3033733.