Київський національний університет імені Тараса Шевченка

**Лабораторна робота**

**З навчального курсу «Розподілене та паралельне програмування»**

**доц. Дерев’янченко О.В.**

Виконав:

студент 4 курсу

факультету комп’ютерних наук та кібернетики

ОНП «Інформатика»

групи ТТП-42

Ходаков Максим Олеговичи

**Київ 2025**

**Тема**

Реалізація програм для генерації множини Мандельброта за допомогою OpenMP та MPI.

**GitHub**

[**https://github.com/maksymkhodakov/RPPLabs**](https://github.com/maksymkhodakov/RPPLabs)

## 1. Вступ

Сучасні обчислювальні задачі вимагають ефективного використання апаратних ресурсів. Однією з таких задач є візуалізація множини Мандельброта, де для кожного пікселя зображення потрібно визначити, скільки ітерацій виконується, перш ніж значення послідовності "втікне" за межі заданого діапазону.  
У даній роботі розроблено дві версії програми:

1. **Версія з OpenMP:**  
   Програма працює в одному процесі, розподіляючи обчислення між потоками за допомогою OpenMP. Це дозволяє використовувати багатоядерність сучасних комп’ютерів.
2. **Версія з MPI:**  
   Програма працює у кількох процесах (можна запускати на кластері або на багатоядерному комп’ютері), де кожен процес обчислює свою частину зображення. Результати з усіх процесів збираються для фінального аналізу.

Основна ідея обох програм – для кожного пікселя виконати ітераційне обчислення за формулою:

  z₍ₙ₊₁₎ = zₙ² + c

де c – комплексне число, яке відповідає координатам пікселя, а z₀ = 0. Ітерації виконуються до тих пір, поки (zₙ.real)² + (zₙ.imag)² ≤ 4 або не досягнуто максимальне значення ітерацій (MAX\_ITER = 1000).

## 2. Опис реалізації

### 2.1. Версія з OpenMP

**Алгоритм і структура коду:**

* **Константи та параметри:**  
  Зображення має розміри 3000 × 3000 пікселів, а максимальна кількість ітерацій для кожного пікселя дорівнює 1000.  
  Значення MAX\_ITER використовується для обмеження кількості ітерацій, що дозволяє уникнути нескінченних циклів для точок, які належать множині Мандельброта.
* **Обчислення координат:**  
  Для кожного пікселя з індексами (i, j) обчислюється дійсна (real) та уявна (imaginary) частини. Для цього застосовується лінійна інтерполяція:  
    real = real\_min + j × (real\_max – real\_min) / (WIDTH – 1)  
    imag = imag\_max – i × (imag\_max – imag\_min) / (HEIGHT – 1)
* **Ітераційне обчислення:**  
  Для кожного пікселя виконується цикл, де розраховується послідовність за формулою  
    z₍ₙ₊₁₎ = zₙ² + c  
  цикл зупиняється, коли (z.real)² + (z.imag)² перевищує 4 або досягається MAX\_ITER. Результат (кількість ітерацій) зберігається у масиві.
* **Паралелізація:**  
  Основний цикл по рядках зображення розподіляється між потоками за допомогою директиви OpenMP «#pragma omp parallel for». Для підрахунку загальної суми ітерацій (контрольна сума), а також для знаходження мінімального та максимального значень, використовується механізм редукції.  
  Крім того, кожен потік інкрементує свій лічильник оброблених рядків (масив rows\_processed). Для того, щоб не перевантажувати консоль, повідомлення виводяться лише для кожного 500-го рядка.
* **Вивід результатів:**  
  Після завершення обчислень програма виводить:
  + Кількість рядків, оброблених кожним потоком.
  + Загальну контрольну суму (суму кількості ітерацій по всіх пікселях).
  + Глобально мінімальне та максимальне значення ітерацій.
  + Час виконання.
  + Значення пікселів у ключових точках: верхній лівий, верхній правий, нижній лівий, нижній правий та центральний піксель.

**Переваги:**  
OpenMP дозволяє легко масштабувати обчислення, змінюючи кількість потоків через змінну середовища OMP\_NUM\_THREADS. Це дає можливість швидко експериментувати з продуктивністю на багатоядерних процесорах.

### 2.2. Версія з MPI

**Алгоритм і структура коду:**

* **Розподіл даних між процесами:**  
  Глобальне зображення ділиться на блоки рядків. Якщо 3000 не ділиться рівномірно на кількість процесів, перші процеси отримують на один рядок більше. Змінні start\_row та end\_row визначають, які саме рядки обробляє кожен процес.
* **Локальні обчислення:**  
  Кожен процес виділяє пам’ять для свого блоку (масив local\_image) і виконує обчислення для кожного пікселя, використовуючи той же алгоритм, що і в версії OpenMP.  
  Локально обчислюється контрольна сума, мінімум та максимум значень ітерацій, а також вимірюється час обчислення за допомогою MPI\_Wtime().
* **Логування:**  
  Кожен процес виводить повідомлення для кожного 500-го глобального рядка, який він обробляє, що дозволяє відслідковувати прогрес обчислень. Також кожен процес виводить кількість рядків, які він обробив.
* **Об’єднання результатів:**  
  Після локальних обчислень за допомогою MPI\_Reduce збираються глобальні контрольна сума, мінімальне та максимальне значення ітерацій, а також час виконання (максимальний серед процесів).  
  Для того, щоб вивести значення пікселів у кутах та в центрі зображення, використовується MPI\_Gatherv, що дозволяє зібрати всі локальні блоки в один глобальний масив (global\_image) на процесі 0.
* **Вивід результатів:**  
  Процес з рангом 0 виводить:
  + Глобальну контрольну суму.
  + Глобально мінімальне та максимальне значення ітерацій.
  + Максимальний час виконання серед процесів.
  + Значення пікселів у ключових точках: верхній лівий, верхній правий, нижній лівий, нижній правий та центральний піксель.

**Переваги:**  
MPI дозволяє розподіляти обчислення між кількома процесами, що особливо корисно при роботі в кластері або при обчисленні дуже великих задач. Можливість збору статистичних даних з усіх процесів дає змогу провести детальний аналіз продуктивності.

## 3. Отримані результати та аналіз

При експериментальному запуску обох програм було отримано наступні результати:

* **Програма OpenMP:**  
  За допомогою OpenMP вдалося розподілити обчислення між потоками. Логування показало, які потоки обробляють кожен 500-й рядок, та яку кількість рядків обробив кожен потік. Загальна контрольна сума, мінімум та максимум значень ітерацій були правильно обчислені, а час виконання зменшувався при збільшенні кількості потоків (визначається змінною OMP\_NUM\_THREADS).
* **Програма MPI:**  
  Розподіл роботи між процесами забезпечив ефективне використання ресурсів. Кожен процес виводив повідомлення про обробку кожного 500-го глобального рядка, а в кінці процес 0 виводив загальну статистику. Збір даних за допомогою MPI\_Gatherv дозволив вивести значення пікселів у кутах та центрі зображення, що підтверджує коректність роботи алгоритму. Час виконання обчислень у MPI-версії також зменшується при збільшенні кількості процесів.

## 4. Демонстрація роботи програм

OpenMP (1 потік):

A screenshot of a computer

Description automatically generated

OpenMP (2 потоки):

A screenshot of a computer code

Description automatically generated

OpenMP (4 потоки):

A screenshot of a computer

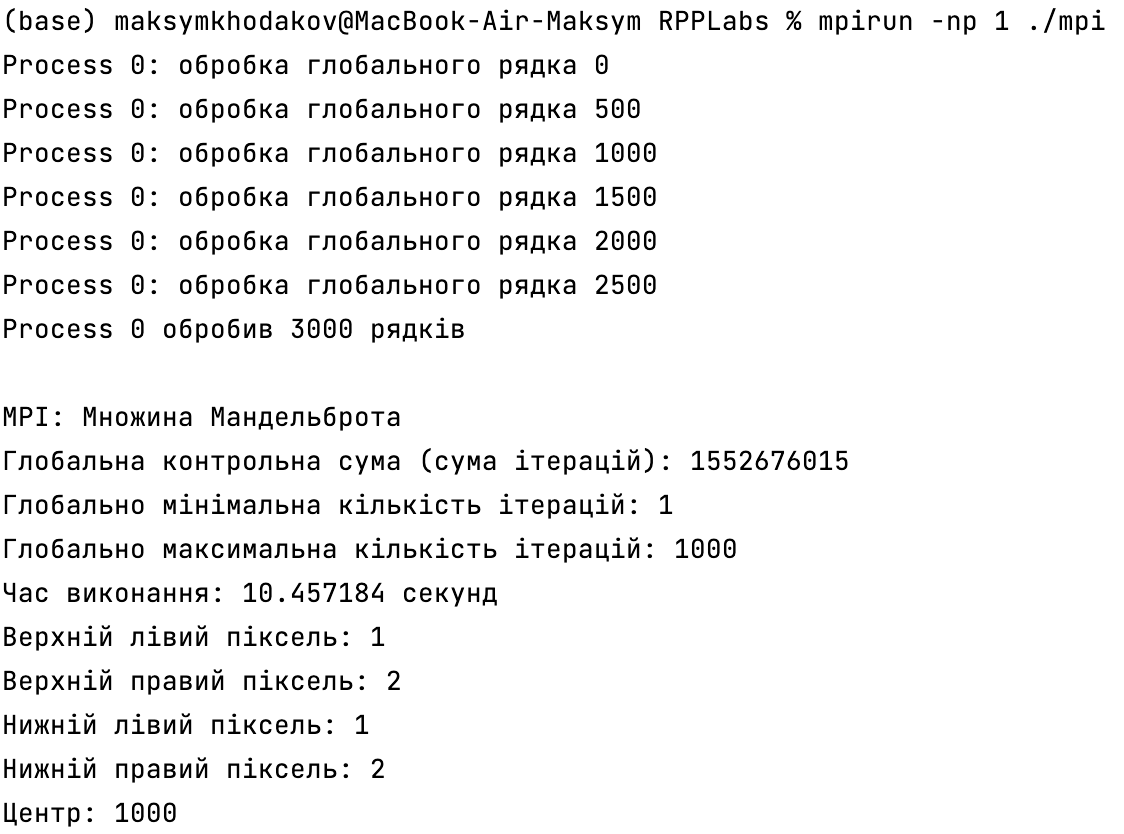
Description automatically generated

OpenMP (8 потоки):

A screenshot of a computer code

Description automatically generated

MPI (1 потік)



MPI (2 потоки)

A screenshot of a computer

Description automatically generated

MPI (4 потоки)

A screenshot of a computer

Description automatically generated

MPI (8 потоки)

A screenshot of a computer

Description automatically generated

OpenMP таблиця виконання:

|  |  |
| --- | --- |
| Кількість потоків (шт.) | Час виконання (с.) |
| 1 | 11.599971 |
| 2 | 6.058324 |
| 4 | 3.210927 |
| 8 | 1.904614 |

MPI таблиця виконання

|  |  |
| --- | --- |
| Кількість потоків (шт.) | Час виконання (с.) |
| 1 | 10.457184 |
| 2 | 5.465329 |
| 4 | 5.446247 |
| 8 | 4.772620 |

## 5. Висновки

Реалізовані програми демонструють ефективність використання паралельних технологій для вирішення обчислювально інтенсивної задачі, такої як обчислення множини Мандельброта. За допомогою детального логування вдалося відслідковувати розподіл навантаження між паралельними одиницями (потоками або процесами) та перевірити коректність обчислень. Отримані результати, такі як загальна контрольна сума, мінімум, максимум значень ітерацій та час виконання, дозволяють проводити подальший аналіз та оптимізацію алгоритму.