Multicore Software Development: Fractal Generation Assignment

1. Problem Statement

A computer graphics application is used to generate high-resolution images of the Mandelbrot set, a famous mathematical fractal. The current implementation is purely sequential, calculating one pixel at a time, which results in long processing times for large images.

Your objective is to improve the performance of this application by creating a parallel version. You will modify the provided source code to leverage the capabilities of modern multi-core processors using the **OpenMP** parallel programming framework.

---

2. Provided Files

The assignment package contains the following files:

- **`Makefile`**: A script to compile both the sequential and parallel versions of the application.
- **`main.c`**: The main driver program. It handles command-line arguments and user input for selecting different fractal views.
- **`mandelbrot.h`**: The header file containing the function declarations.
- **`mandelbrot_seq.c`**: The complete, working sequential implementation. **Do not modify this file.** It serves as the baseline for your performance comparison.
- **`mandelbrot_par.c`**: The source file you must modify. **Initially, it is identical to the sequential version.**
- **`stb_image_write.h`**: An external, single-file library used for saving images in the PNG format.

---

3. Assignment Requirements

You are required to perform the following tasks:

1. **Compile and Run:** Use the provided `Makefile` to compile the entire project. Run the sequential version (`mandelbrot_seq`) to understand its operation and measure its baseline performance on a high-resolution image.

2. **Code Analysis:** Analyze the source code in `mandelbrot_seq.c` to identify the most computationally expensive part of the program (the "hotspot") that is suitable for parallelization.
3. **Parallel Implementation:** Modify the `mandelbrot_par.c` file to implement a parallel version of the Mandelbrot generation algorithm using OpenMP directives. You should not change the core logic of the algorithm itself.
    - Experiment around with OpenMP functionalities to try and achieve the maximum possible speedup. (Document all of this in your report.)
4. **Validation:** Ensure that the images generated by your parallel version (`mandelbrot_par`) are pixel-for-pixel identical to the images generated by the sequential version (`mandelbrot_seq`).
5. **Performance Measurement:** Measure the execution time of both the sequential and parallel versions using a consistent, high-resolution test case (e.g., 3840x2160). Calculate the speedup achieved by your parallel implementation.

** You can experiment around with varying image sizes as well to learn how size affects performance and execution timings.

---

4. Deliverables

You must submit the following items:

1. **Modified Source Code:**
    - Your modified `mandelbrot_par.c` file containing the OpenMP implementation.
2. **Performance Report:**
    - A brief report in a pdf file (`report.pdf`) or markdown file (`report.md`) that includes:
        - The final speedup value you calculated.
        - Testing statistics of the parallel implementation with varying number of threads and different OpenMP functionalities.
        - A detailed justification for the observed speedup, explaining how your parallel implementation achieves this performance gain.
        - The specifications of the hardware you used for testing (CPU model and number of cores).
    - Graphical representations of the statistics that you obtain, in the report.

---

5. How to Compile and Run 💻

Compilation

To compile the source code, open a terminal in the project directory and run the `make` command. It is good practice to clean any previous builds first.

```
make clean && make
```

This command will create two executable files: **`mandelbrot_seq`** and **`mandelbrot_par`**.

Running the Executables

The program is interactive. You provide the image dimensions and filename as command-line arguments, and it will then prompt you to choose which fractal view to generate.

Example: Running the Sequential Version

1. Run the executable with your desired settings:

```
./mandelbrot_seq 1920 1080 mandelbrot_sequential.png
```

2. The program will display a menu and wait for your input. Type a number and press Enter.

```
Please select an image type to generate:
  1: Full View
  2: Seahorse Valley
  3: Elephant Valley
Enter your choice: 2
```

3. The program will then generate the image and report the time taken.

Example: Running the Parallel Version

The process is identical, just use the parallel executable:

1. Run the executable with your desired settings:

```
./mandelbrot_par 1920 1080 mandelbrot_parallel.png
```

2. Enter your choice when prompted:

```
Please select an image type to generate:
  1: Full View
  2: Seahorse Valley
  3: Elephant Valley
Enter your choice: 2
```

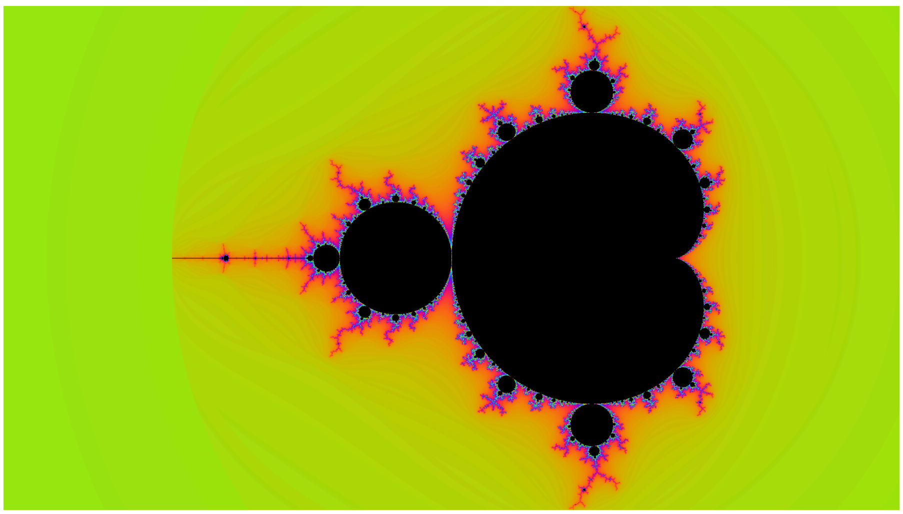
3. Compare the execution time of this run with the sequential version to calculate your speedup.

6. Expected Output Images

This section shows the expected output for each of the interactive choices. Your program's output should look identical to these reference images.

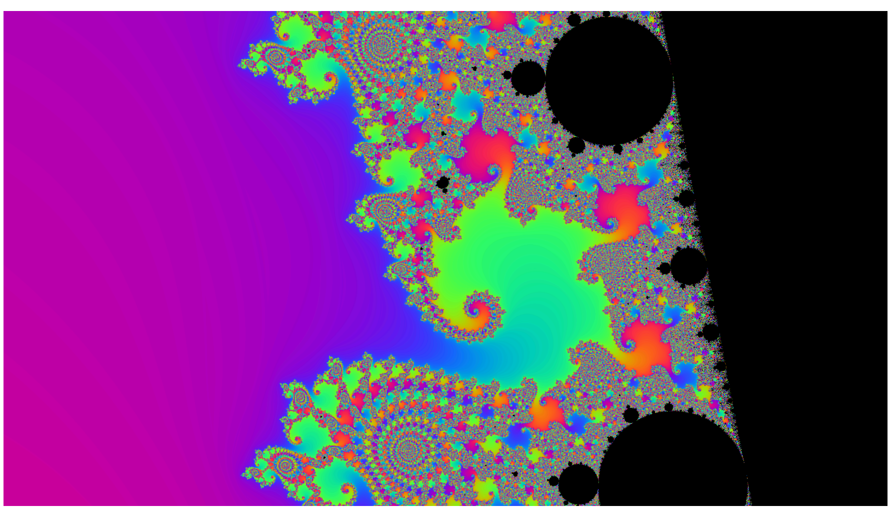
Choice 1: Full View

This is the default, zoomed-out view of the entire Mandelbrot set.



Choice 2: Seahorse Valley

This is a zoomed-in view of a highly detailed, spiral region.



Choice 3: Elephant Valley

This is a zoomed-in view of a different region, known for its large, circular shape.