HPC

MANDELBROT SET IMPLEMENTATION HYBRID MPI+OPENMP

NIGAM VISHAL [SM3800014]

University of Trieste

1. Introduction: Exploring the Mandelbrot Set with Parallel Computing

The objective of this project is to compute and visualize the Mandelbrot set, a well-known fractal that is both mathematically intriguing and computationally intensive. Given its properties, the Mandelbrot set is an excellent candidate for parallel computing. Each point in the complex plane can be computed independently, making it ideal for a hybrid approach that utilizes both the Message Passing Interface (MPI) and Open Multi-Processing (OpenMP).

In our implementation, we will explore both strong and weak scaling to evaluate performance. Strong scaling tests how the solution time varies with a constant problem size as the number of processors increases, while weak scaling examines how the solution time changes when both the number of processors and the problem size grow proportionally.

To conduct this evaluation, we will consider two main scenarios:

1. **MPI Scaling:** In this scenario, we will fix the number of OpenMP threads to one per MPI process and increase the number of MPI processes. This setup allows us to evaluate the scalability of distributing tasks across multiple processes, potentially running on different processors or nodes.

2. **OpenMP Scaling:** Here, we will fix the number of MPI processes to one and increase the number of OpenMP threads. This scenario will help us understand how the computation scales within a single process using multiple threads, leveraging multi-core processor capabilities.

These experiments will be carried out on the ORFEO cluster, utilizing its high-performance computing resources to assess the efficiency and performance of our hybrid MPI and OpenMP implementation. By conducting these scaling studies, we aim to optimize the computation of the Mandelbrot set and gather valuable insights into effectively utilizing parallel computing resources for similar tasks. This analysis will enhance our understanding of parallel algorithms and their practical application in high-performance computing environments.

2.Computational Architecture

In this project, we will implement a hybrid MPI and OpenMP solution to compute the Mandelbrot set. By leveraging the combined strengths of MPI and OpenMP, we can efficiently distribute the computational workload across multiple nodes and cores within the ORFEO cluster, specifically utilizing the THIN partition. This partition consists of several nodes, each equipped with two Intel Xeon Gold CPUs. Each CPU contains 12 cores, providing significant computational power. The nodes are interconnected via a high-speed network, enabling efficient data transfer between nodes. Each processor is organized into multiple Core Complexes (CCXs), each containing several cores with substantial L3 cache, ensuring efficient handling of intensive computations. This setup allows us to perform both strong and weak scaling experiments effectively on the THIN partition.

3. Understanding of Mandelbrot Set

The Mandelbrot set is a collection of complex points that form a fractal, renowned for its intricate and self-repeating patterns. This set is defined by iterating the function

$$fc(z)=z^2+c$$

on the complex plane, where zzz and ccc are complex numbers.A point belongs to the Mandelbrot set if the sequence generated by repeatedly applying the function remains bounded and does not tend toward infinity.

To visualize the Mandelbrot set, each point in the complex plane is mapped to a pixel in a two-dimensional grid, and its membership in the set is determined. For efficient storage and processing, this two-dimensional image is often stored as a one-dimensional array of pixels. Each pixel is represented as an unsigned char, with values ranging from 0 to 255.

Grayscale is used to color the image, providing a visual representation of the set. If a point is part of the Mandelbrot set, the corresponding pixel is assigned a value of 0 and is colored black. If a point is not part of the set, the pixel value corresponds to the iteration count at which the point exceeds a certain threshold, with values between 1 and 255 representing different shades of gray. This grayscale representation helps to highlight the complex boundary between the inside and outside of the Mandelbrot set.

4. Implementation

4.1 Implementation of mandelbrot_pgm.c

The `mandelbrot_pgm.c` file serves as the main implementation for computing and visualizing the Mandelbrot set using a hybrid MPI and OpenMP approach. This implementation leverages parallel processing to efficiently compute the Mandelbrot set over a specified region of the complex plane and generates a Portable Gray Map (PGM) image of the set.

The core of the file includes several key components. Firstly, the `mandelbrot` function determines if a point belongs to the Mandelbrot set. It iterates over the complex function $z \rightarrow z^2+c$ for a given point ‘c’ and checks whether the magnitude of the result remains bounded. If at any point during the iterations the magnitude exceeds 4, the function concludes that the point does not belong to the set and returns the current iteration count. Otherwise, it returns the maximum iteration count, indicating the point is within the Mandelbrot set.

The `write_pgm_image` function handles the generation of the output image. This function takes the computed matrix MMM, representing the Mandelbrot set, and writes it to a PGM file. Each pixel's value in the image corresponds to the iteration count at which the point exited the radius-2 circle, normalized to a 0-255 grayscale. Points within the Mandelbrot set are represented in black (0), while points outside are scaled accordingly to produce shades of gray.

The `main` function orchestrates the overall computation. It starts by parsing command-line arguments to set the grid dimensions, complex plane boundaries, and maximum iterations. The function then initializes MPI, distributing the computation across multiple processes. Each

process computes its assigned portion of the Mandelbrot set in parallel, utilizing OpenMP to further distribute the workload among available cores.

The computation involves dividing the complex plane into a grid and calculating the Mandelbrot set for each grid point. This task is parallelized using OpenMP's dynamic scheduling to balance the load, particularly given the varying computational demands of points near the set's boundary.

Once each process completes its computation, the results are gathered at the root process. MPI communication routines facilitate this data aggregation, ensuring all computed sections of the Mandelbrot set are collected. Finally, the root process invokes the `write_pgm_image` function to save the complete image in the specified directory.

Overall, the `mandelbrot_pgm.c` file effectively demonstrates a hybrid parallel implementation, leveraging MPI for inter-process communication and OpenMP for intra-process parallelism, to compute and visualize the Mandelbrot set efficiently.

4.2 Parallelization in terms of MPI & OpenMP

In the `mandelbrot_pgm.c` program, both MPI (Message Passing Interface) and OpenMP (Open Multi-Processing) are utilized to achieve parallelization. This hybrid approach combines the strengths of distributed and shared memory parallelism, allowing the program to efficiently compute the Mandelbrot set over a large region of the complex plane.

MPI Parallelization

MPI is used to distribute the workload across multiple processes, potentially running on different nodes of a computing cluster. The key aspects of MPI parallelization in this program include:

1. **Initialization and Finalization:**
   - The program begins by initializing the MPI environment using `MPI_Init(&argc, &argv)`, which prepares the processes for communication.
   - At the end of the program, `MPI_Finalize()` is called to clean up the MPI environment.
2. **Process Management:**
   - `MPI_Comm_rank(MPI_COMM_WORLD, &rank)` retrieves the rank (ID) of the current process.
   - `MPI_Comm_size(MPI_COMM_WORLD, &size)` determines the total number of processes involved in the computation.
3. **Work Distribution:**
   - The complex plane is divided into horizontal slices, each assigned to a different process. This is achieved by calculating `chunk_size`, which defines the number of rows each process will handle.
   - Each process computes the Mandelbrot set for its assigned rows, from `start_row` to `end_row`.

4. **Data Gathering:**

    - After computing its portion of the Mandelbrot set, each process sends its results to the root process (rank 0) using `MPI_Send`.
    - The root process gathers these results using `MPI_Recv` and combines them into a single matrix.

OpenMP Parallelization

OpenMP is used within each MPI process to further distribute the computation across multiple CPU cores. The key aspects of OpenMP parallelization in this program include:

1. **Parallel Region:**

    - The main computation loop is parallelized using OpenMP's `#pragma omp parallel for schedule(dynamic)`. This directive tells the compiler to parallelize the for loop that iterates over the rows assigned to the process.

    - `schedule(dynamic)` ensures that the workload is dynamically balanced among the threads, which is important because the computational effort varies significantly across different points in the Mandelbrot set.

2. **Thread Management:**

    - OpenMP automatically manages the creation and synchronization of threads. Each thread computes the Mandelbrot set for a subset of the rows assigned to the process.
    - The `mandelbrot` function is called concurrently by multiple threads, each working on different rows and columns of the matrix.

Combined MPI and OpenMP Parallelization

The combination of MPI and OpenMP allows the program to leverage both distributed and shared memory parallelism:

- **MPI distributes the overall workload** among processes, with each process handling a different section of the complex plane. This allows the program to scale across multiple nodes in a cluster.
- **OpenMP further divides the workload** within each process, utilizing multiple CPU cores to compute the Mandelbrot set for the assigned section. This improves the efficiency of computation on multi-core processors.

In summary, the `mandelbrot_pgm.c` program efficiently computes the Mandelbrot set by leveraging MPI for inter-process communication and workload distribution across nodes, and OpenMP for parallel computation within each process. This hybrid approach ensures that the program can scale effectively on high-performance computing clusters, making full use of available computational resources.

4.3 Experiment Plan for Evaluating Strong and Weak Scaling of Mandelbrot Set Computation

Objective

The primary objective of this experiment is to evaluate the performance of the hybrid MPI and OpenMP implementation for computing the Mandelbrot set. We aim to determine the strong and weak scaling of the code, leveraging the computational resources of the ORFEO cluster, specifically the THIN partition.

Scaling Strategies

The experiments are designed to assess both strong and weak scaling:

1. **Strong Scaling**: Keeping the problem size fixed and varying the number of processing units to measure how the solution time decreases as more resources are added.
2. **Weak Scaling**: Increasing both the problem size and the number of processing units proportionally to measure how the solution time changes, ideally remaining constant as more resources are added.

Experiment Setup

The experiments are conducted on the THIN partition of the ORFEO cluster, which comprises nodes equipped with Intel processors. Each node can handle up to 24 tasks.

Test Scenarios

1. **OpenMP Strong Scaling**:
   - **Script**: `strong_scaling_openmp_THIN.sh`
   - **Description**: This script evaluates the performance of the Mandelbrot set computation by increasing the number of OpenMP threads while keeping a single MPI task. The problem size remains constant throughout the tests.
   - **Parameters**: Fixed problem size of 1000×10001000 \times 10001000×1000 pixels.
   - **Threads Range**: 2 to 64 in increments of 2.
2. **MPI Strong Scaling**:
   - **Script**: `strong_scaling_mpi_THIN.sh`
   - **Description**: This script assesses the performance by increasing the number of MPI tasks while keeping the number of OpenMP threads fixed at one per MPI task. The problem size remains constant.
   - **Parameters**: Fixed problem size of 1000×10001000 \times 10001000×1000 pixels.
   - **MPI Tasks Range**: 1, 2, 4, 8, 16, 32, 64.
3. **OpenMP Weak Scaling**:
   - **Script**: `weak_scaling_openmp_THIN.sh`
   - **Description**: This script examines the performance by increasing the number of OpenMP threads and proportionally scaling the problem size. The number of MPI tasks is fixed at one.
   - **Parameters**: Base problem size of 1000×10001000 \times 10001000×1000 pixels, scaled proportionally with the number of threads.
   - **Threads Range**: 2 to 64 in increments of 2.
4. **MPI Weak Scaling**:
   - **Script**: `weak_scaling_mpi_THIN.sh`
   - **Description**: This script evaluates the performance by increasing the number of MPI tasks and proportionally scaling the problem size, with a single OpenMP thread per MPI task.
   - **Parameters**: Base problem size of 1000×10001000 \times 10001000×1000 pixels, scaled proportionally with the number of tasks.
   - **MPI Tasks Range**: 1, 2, 4, 8, 16, 32, 64.

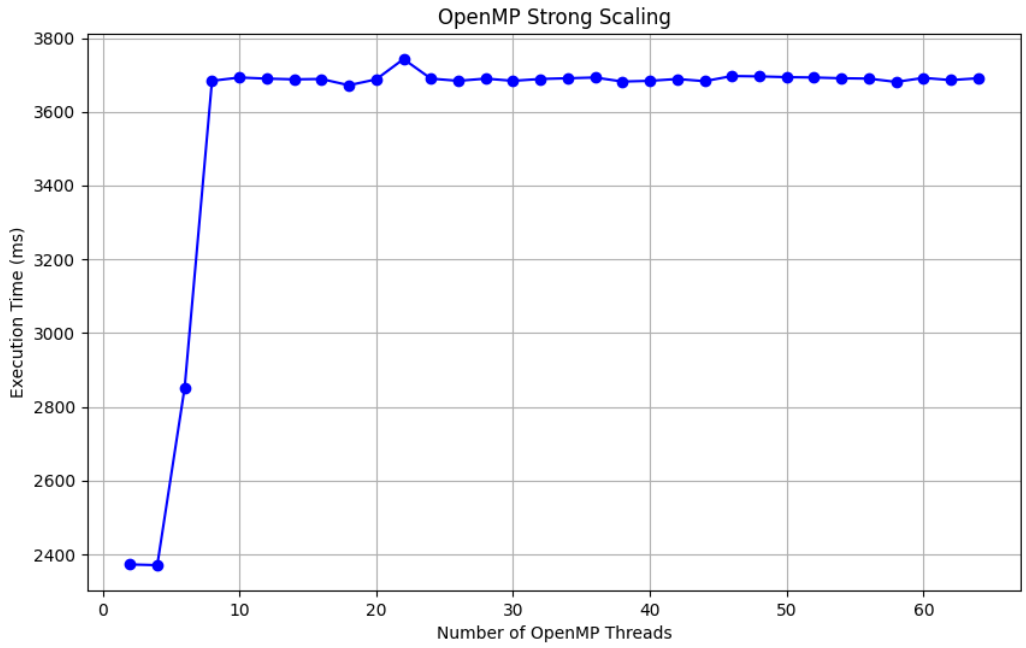
Expected Outcomes

- **Strong Scaling**: Ideally, the runtime should decrease as the number of processing units increases, demonstrating efficient parallelization.
- **Weak Scaling**: The runtime should remain relatively constant as both the problem size and the number of processing units increase proportionally, indicating good scalability.

This experiment plan provides a comprehensive framework for evaluating the performance and scalability of the Mandelbrot set computation on the ORFEO cluster using both MPI and OpenMP. The results from these tests will offer insights into optimizing the hybrid implementation and effectively utilizing high-performance computing resources for similar parallelizable problems.

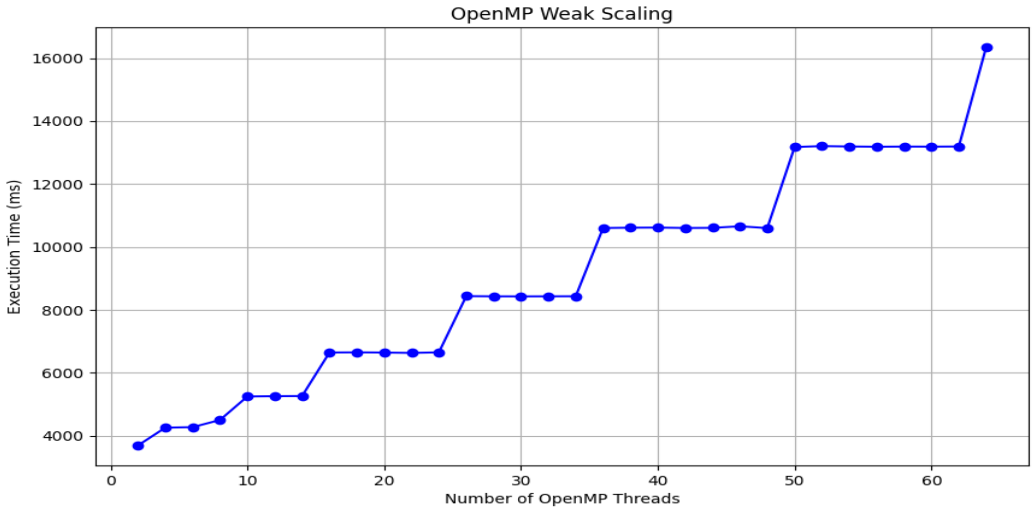
5.Results and Analysis

OpenMP Strong Scaling:



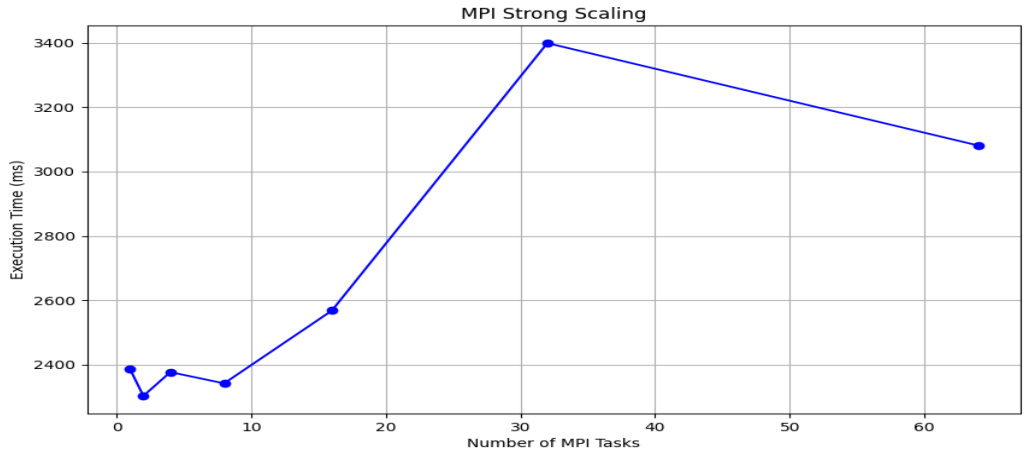
Analysis: The OpenMP strong scaling results show an initial increase in execution time as the number of threads increases from 2 to 8, after which the execution time stabilizes around 3680 ms. This plateau suggests that adding more threads beyond 8 does not significantly improve performance, indicating potential issues with parallel overhead or memory bandwidth saturation.

OpenMP Weak Scaling:



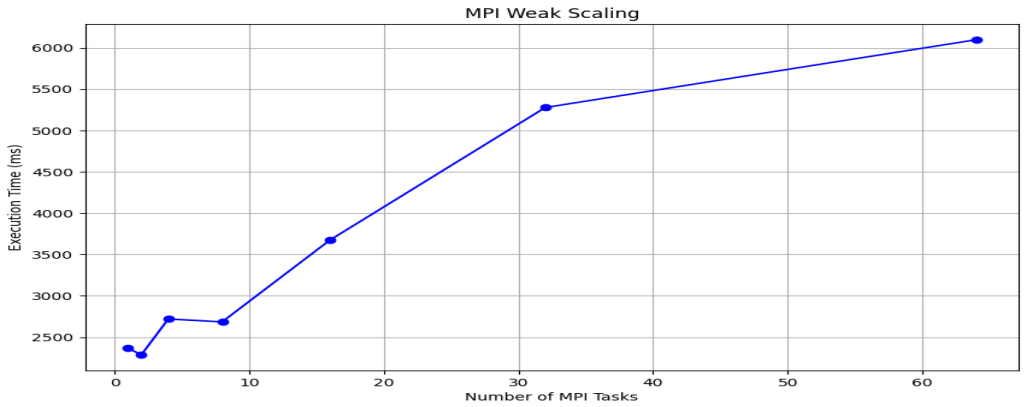
Analysis: The OpenMP weak scaling results show a consistent increase in execution time as the number of threads increases. This is expected as the problem size is scaled with the number of threads. However, the linear increase in execution time indicates a decent scaling performance, although there are noticeable jumps at specific thread counts, suggesting inefficiencies or load imbalance at those points.

MPI Strong Scaling:



Analysis: The MPI strong scaling results show a relatively stable execution time up to 8 MPI tasks, after which there is a significant increase in execution time at 16 tasks and 32 tasks. This indicates that beyond a certain number of tasks, the overhead of communication starts to outweigh the benefits of parallelism. The dip at 64 tasks suggests some performance improvement but not enough to offset the earlier increase.

MPI Weak Scaling:



Analysis: The MPI weak scaling results show an expected increase in execution time as the number of MPI tasks and problem size increase. However, the increase is not linear, indicating that the parallel efficiency decreases with the number of tasks. This could be due to communication overhead becoming more significant as the number of tasks grows.

6. Conclusion

Overall Findings

The Mandelbrot set computation, implemented using both OpenMP and MPI on the ORFEO cluster, provided comprehensive insights into the performance and scalability of parallel programming approaches. The strong and weak scaling tests revealed several key points:

1. **OpenMP Strong Scaling**: The execution time plateaued around 3680 ms beyond 8 threads, indicating that adding more threads did not improve performance. This suggests that parallel overhead or memory bandwidth limitations may be causing inefficiencies.
2. **OpenMP Weak Scaling**: The execution time increased proportionally with the number of threads, showing reasonable scaling performance. However, the step increases at certain thread counts highlighted inefficiencies and potential load imbalances.
3. **MPI Strong Scaling**: The execution time remained consistent up to 8 MPI tasks but increased significantly at 16 and 32 tasks due to communication overhead outweighing the benefits of parallelism. There was a slight performance improvement at 64 tasks, but it was insufficient to compensate for the earlier increase.
4. **MPI Weak Scaling**: As the number of MPI tasks and problem size grew, execution time increased non-linearly, indicating decreasing parallel efficiency due to communication overhead.