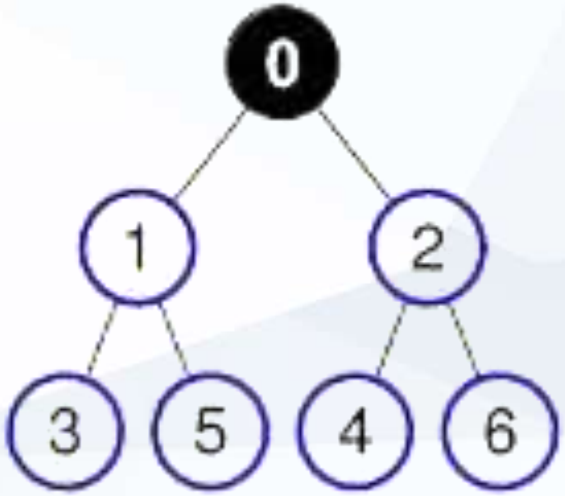
Performance Comparison of MPI Broadcast Algorithms

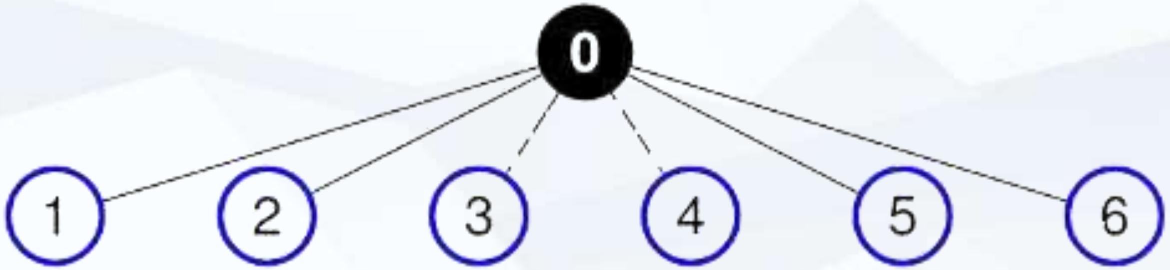
Gaia Alessio

Algorithms



(c) Binary tree

(b) Chain tree



(a) Flat tree

Binary tree

```
if (my_rank==my_root_rank){
    if (my_left_child_rank   my_procs)
        MPI_Send(my_data, my_num_elements, MPI_INT, my_left_child_rank, 0, MPI_COMM_WORLD);
    if (my_right_child_rank   my_procs)
        MPI_Send(my_data, my_num_elements, MPI_INT, my_right_child_rank, 0, MPI_COMM_WORLD);
} else
    MPI_Recv (my_data, my_num_elements, MPI_INT, my_parent_rank, 0, MPI_COMM_WORLD,
    &my_status);
    if (my_left_child_rank   my_procs)
        MPI_Send(my_data, my_num_elements, MPI_INT, my_left_child_rank, 0, MPI_COMM_WORLD);
    if (my_right_child_rank   my_procs)
        MPI_Send(my_data, my_num_elements, MPI_INT, my_right_child_rank, 0, MPI_COMM_WORLD);
    }
}
```

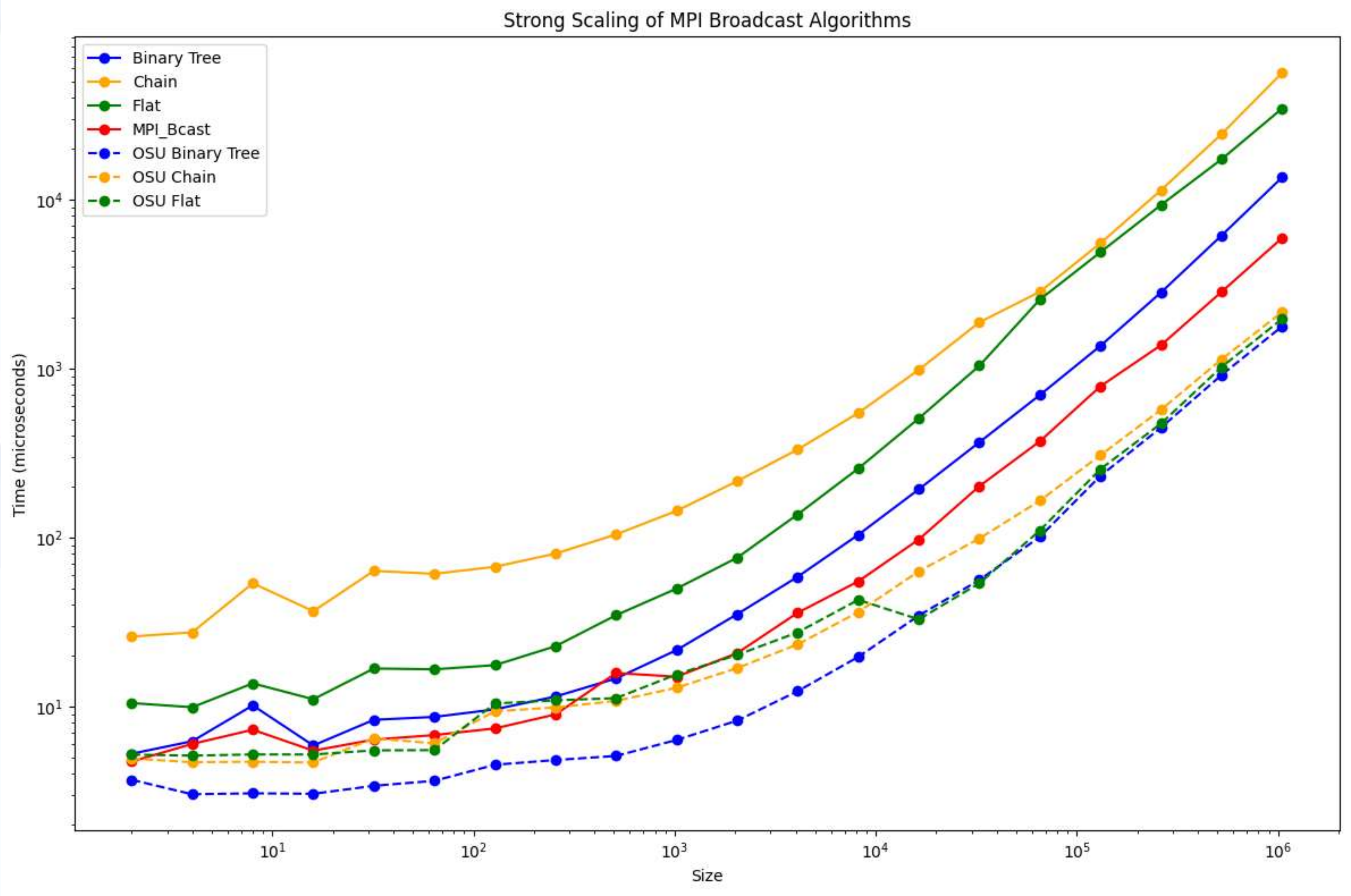
Chain

```
if (my_rank==my_root_rank){
    MPI_Send(my_data, my_num_elements, MPI_INT, my_child_rank, 0, MPI_COMM_WORLD);
} else {
    MPI_Recv (my_data, my_num_elements, MPI_INT  my_parent_rank, 0, MPI_COMM_WORLD,
    &my_status);
    if (my_child_rank   my_procs)
        MPI_Send(my_data, my_num_elements, MPI_INT, my_child_rank, 0, MPI_COMM_WORLD)
    }
}
```

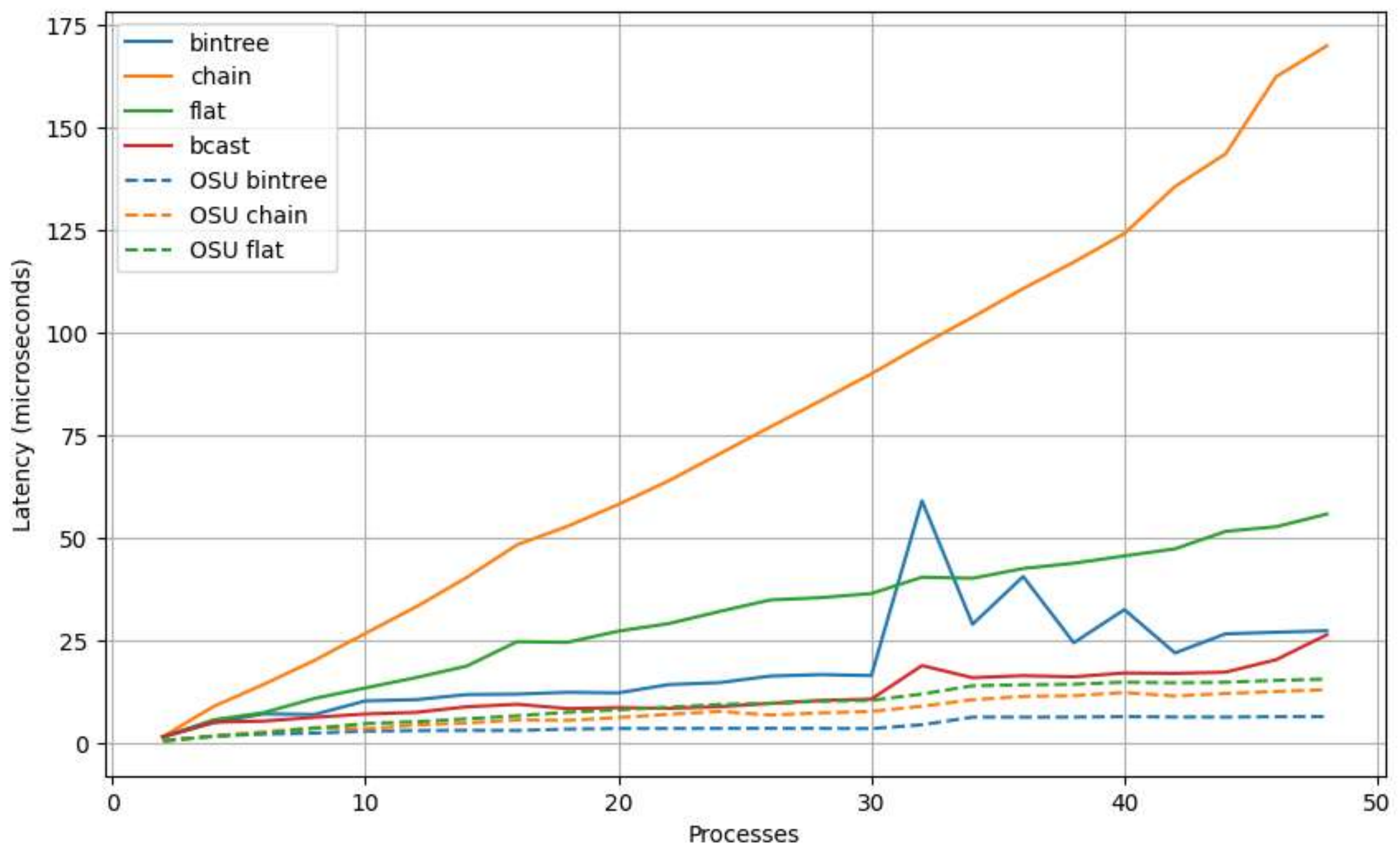
Flat

```
if (my_rank==my_root_rank){
    for(my_i; my_i< my_procs; my_i++){
        MPI_Send(my_data, my_num_elements, MPI_INT, my_i, 0, MPI_COMM_WORLD)
    }
}
else {
    MPI_Recv (my_data, my_num_elements, MPI_INT, my_root_rank, 0, MPI_COMM_WORLD,
    &my_status)
}
```

Strong scalability



Weak scalability



The Mandelbrot set

Introduction

Develop an efficient and scalable code for generating the Mandelbrot set using OpenMP.

Complex geometric structure defined by the iterative function $f_c(z) = z^2 + c$ in the complex plane.

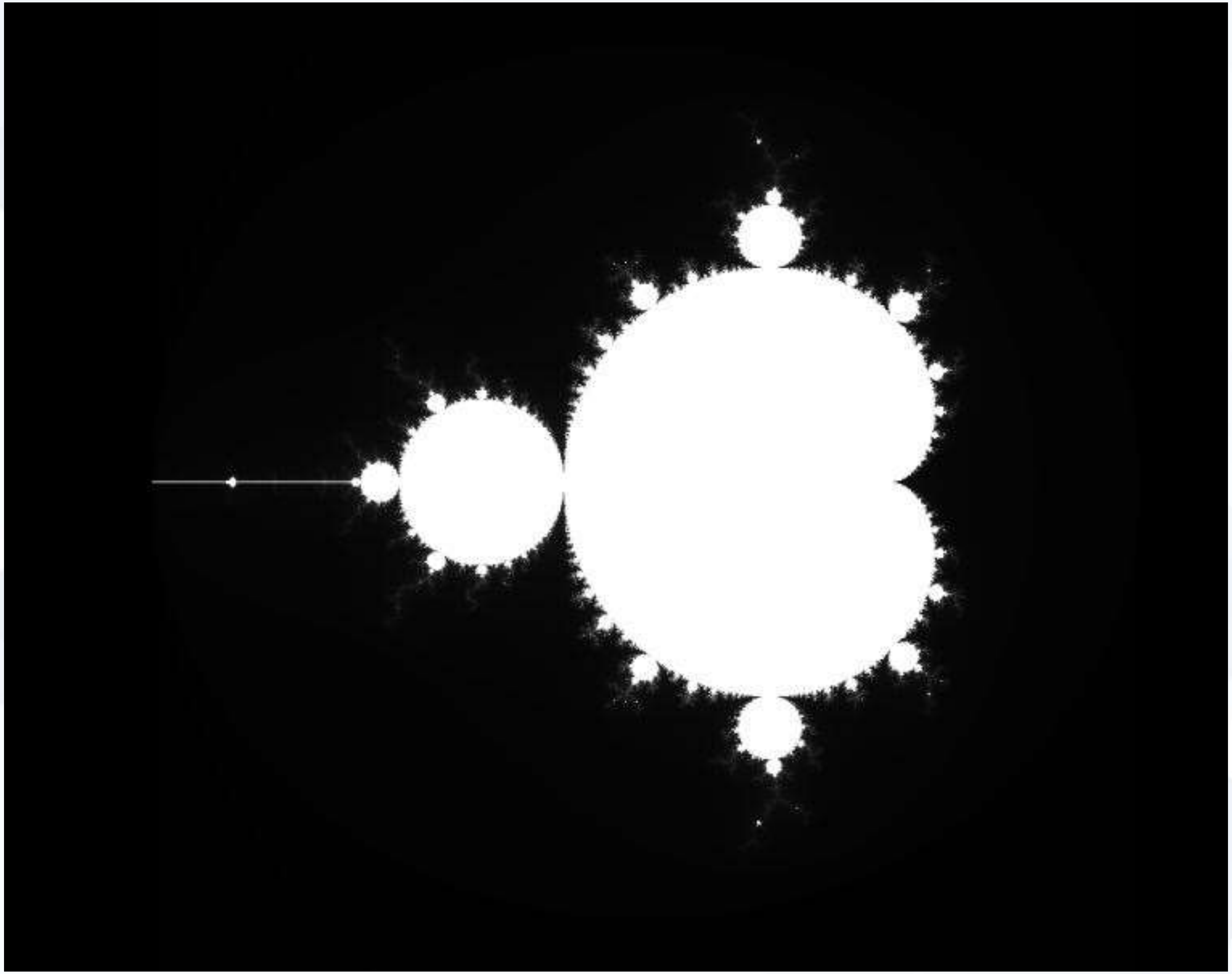
A point c is in the Mandelbrot set if the series does not diverge.

Implementation

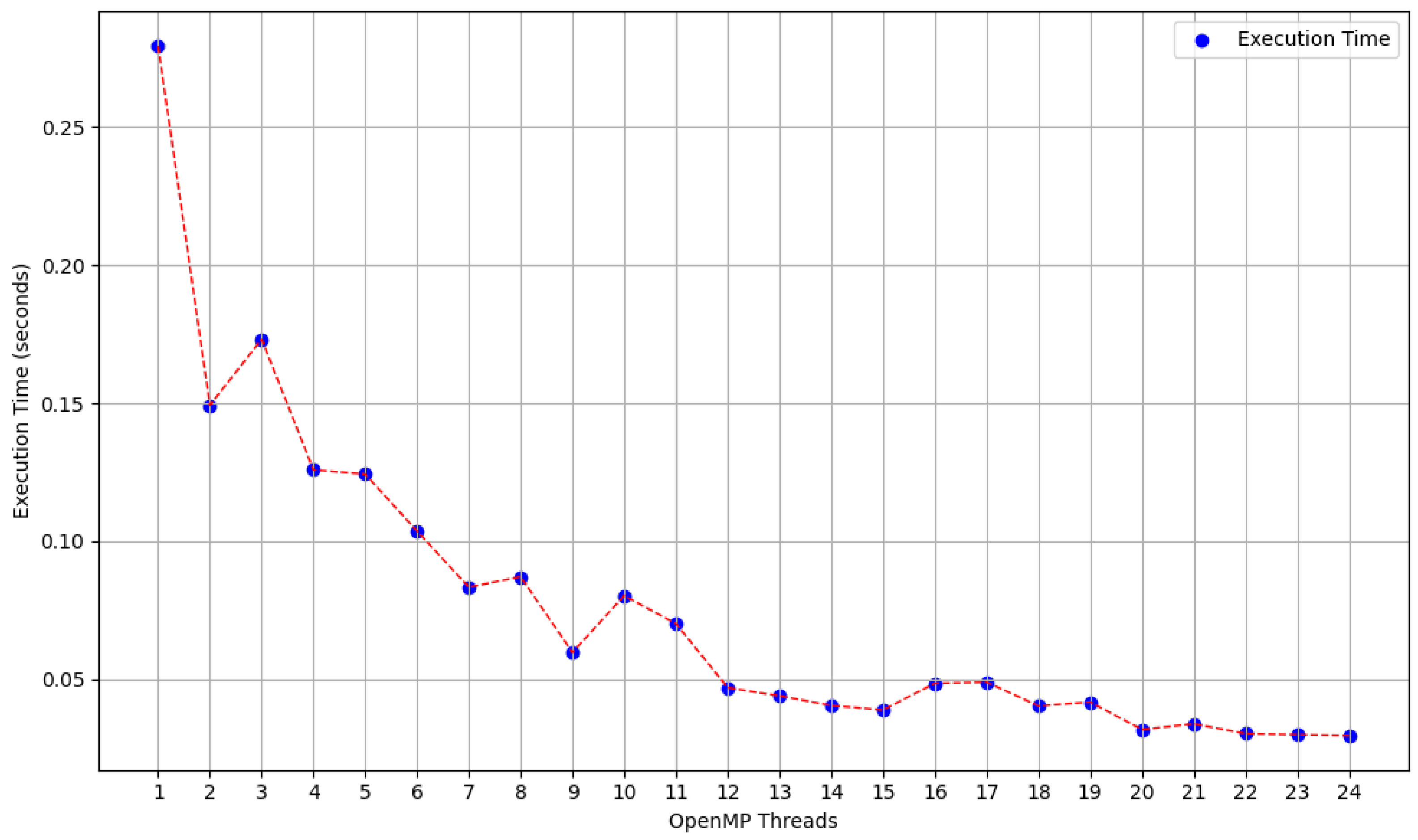
```
unsigned char mandelbrot(double real, double imag, int max_iter) {
    double z_real = real;
    double z_imag = imag;
    for (int n   0; n   max_iter; n++) {
        double r2 = z_real * z_real;
        double i2   z_imag * z_imag;
        if (r2 + i2   4.0) return n;
        z_imag = 2.0 * z_real * z_imag + imag;
        z_real   r2   i2   real;
    }
    return max_iter;
}
```

Implementation

```
#pragma omp parallel for collapse(2)
  for (int j = 0; j   height; j++)
      for (int i = 0; i   width; i++) {
          double x   x_left   i * (x_right   x_left)   width;
          double y   y_lower   j   (y_upper   y_lower)   height;
          unsigned char value   mandelbrot(x, y, max_iterations);
          image_buffer[j   width   i]   value;
      }
  }
```



Strong scalability



Weak scalability