# HIGH PERFORMANCE COMPUTING

## Exercise 2

Barrasso Marco

May 13, 2024

# Overview

- Aim of the project
  - Generate Mandelbrot set using an hybrid MPI+OpenMP approach and determine strong and weak scaling
- Implementations
  - First with OpenMP
  - Then incorporating MPI
- Scaling
  - OpenMP: run with a single MPI task and increase the number of OMP threads
  - MPI: run with a single OMP thread per MPI task and increase the number of MPI tasks.
- Metodology
  - Bash script to obtain data using at most 4 THIN nodes on ORFEO cluster
  - R for data analysis

# Mandelbrot Set

The Mandelbrot set is defined in the complex plane $\mathbb{C}$ as the set of complex numbers $c$ for which the function $f_c(z) = z^2 + c$ does not diverge to infinity when iterated at $z = 0$, i.e. for which the sequence $z_0 = 0, z_1 = f_c(0), z_2 = f_c(z_1), \ldots, f_c^n(z_{n-1})$ is bounded.

The simple condition to determine whether a point $c$ is in the set is the following

$$|z_n = f_c^n(0)| < 2 \quad \vee \quad n > I_{max}$$

where $I_{max}$ is a parameter that sets the maximum number of iteration after which you consider the point $c$ to belong to the set.

# OpenMP

**Algorithm** Calculate Mandelbrot

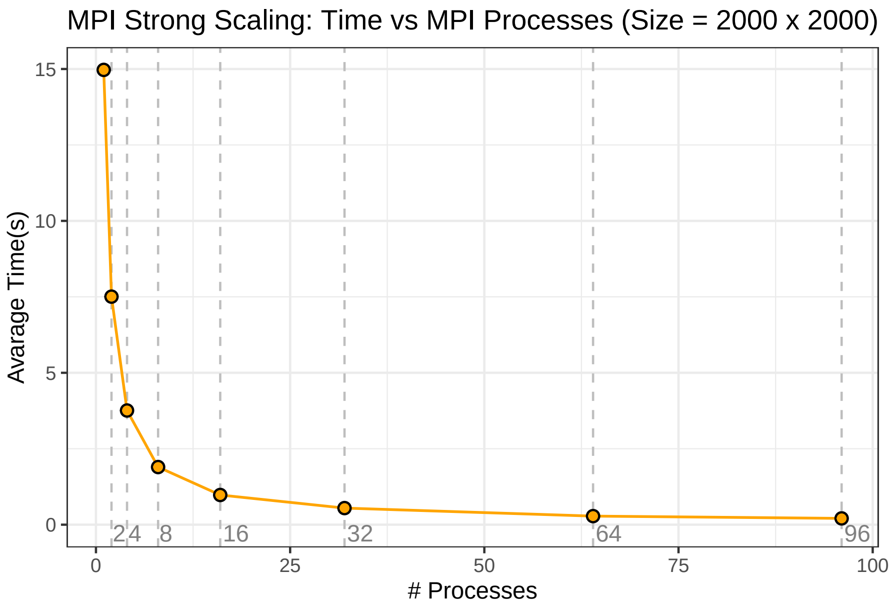
```
for $i = 0$ to $n_y$ do
    #pragma omp parallel for schedule(dynamic)
    for $j = 0$ to $n_x$ do
        $c \leftarrow (x_L + j \cdot \Delta_x) + i \cdot (y_L + i \cdot \Delta_y)$
        $val \leftarrow c$
        $k \leftarrow 0$
        while $k <$ Imax and $|val| < 2$ do
            $val \leftarrow val^2 + c$
            $k \leftarrow k + 1$
        end while
        if $k ==$ Imax then
            $image[i, j] \leftarrow 0$
        else
            $image[i, j] \leftarrow k$
        end if
    end for
end for
```

- $n_x$ and $n_y$ are the dimensions of the matrix
- $(x_L, y_L)$, $(x_R, y_R)$ coordinates to determine portion of the complex plane
- $\Delta_x = (x_R - x_L)/n_x$, $\Delta_y = (y_R - y_L)/n_y$

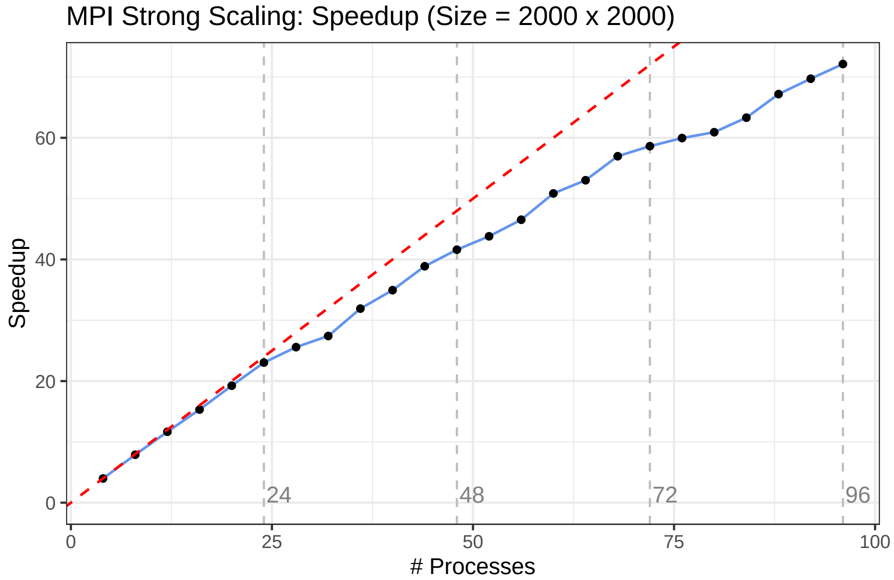
# MPI

- Distributing rows among processes in a round-robin fashion, each process receives rows that are spaced out by the total number of processes
- Each processes has its own matrix, on which will be called the Mandelbrot function, implemented using OpenMP
- Collect the result in a single matrix using MPI_GATHERv function
- Reordering the matrix to reproduce the orignial layout

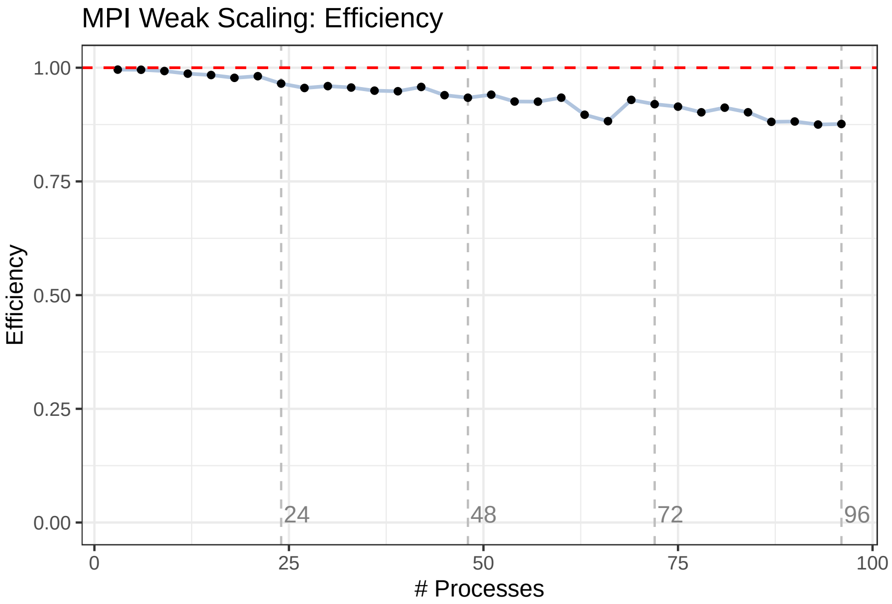
# MPI Scaling



MPI Strong Scaling: Time vs MPI Processes (Size = 2000 x 2000)

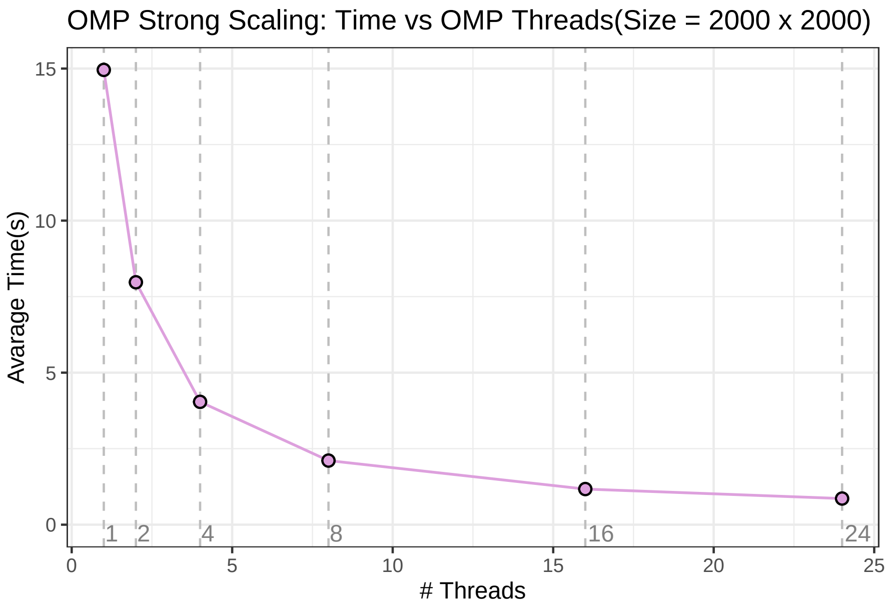


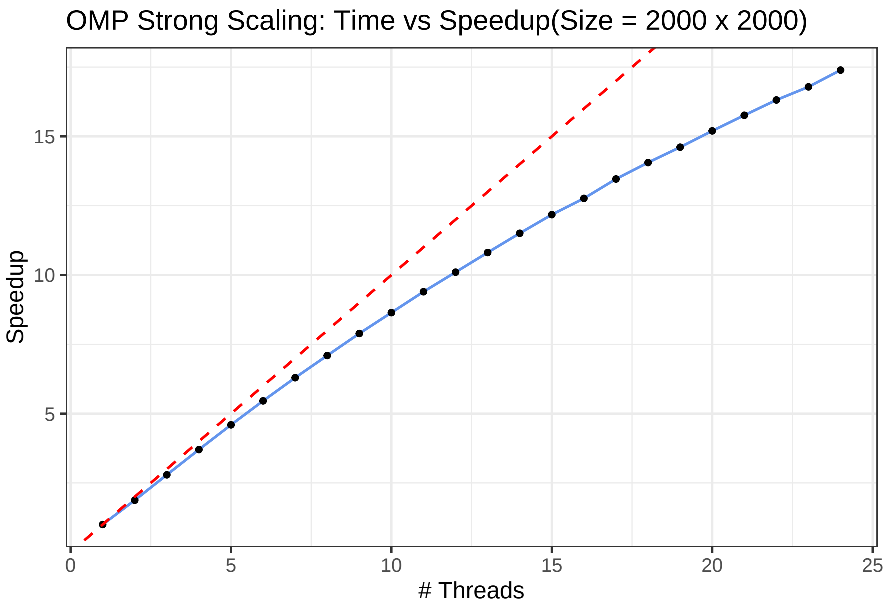
MPI Strong Scaling: Speedup (Size = 2000 x 2000)

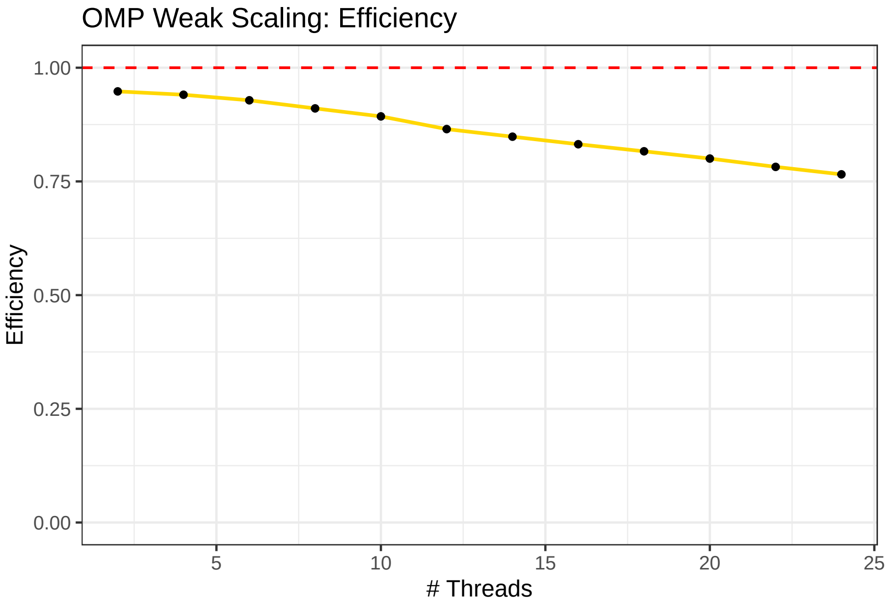


MPI Weak Scaling: Efficiency

# OpenMP Scaling


OMP Strong Scaling: Time vs OMP Threads(Size = 2000 x 2000)


OMP Strong Scaling: Time vs Speedup(Size = 2000 x 2000)


OMP Weak Scaling: Efficiency

# Generated Image