The Mandelbrot Set
Hybrid MPI/OpenMP Implementation

Alessandro Della Siega

University of Trieste

May 2024

Introduction

The goal is to implement and analyze a **hybrid MPI/OpenMP** implementation of the computation of the Mandelbrot set, which is defined as:

$$\mathcal{M} = \{c \in \mathbb{C} : \lim_{n \to \infty} z_n < \infty\}$$

where $z_{n+1} = z_n^2 + c$ and $z_0 = 0$.

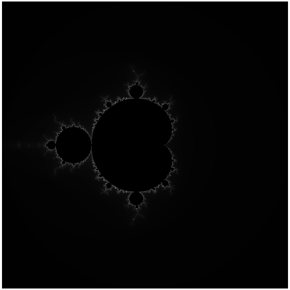


Figure: Rendering of the Mandelbrot set.

Encoding:

- each pixel represents a complex number c.
- the color of the pixel depends on the number of iterations before z_n diverges.

Computational architecture

ORFEO cluster
EPYC paritition

- 8 nodes
- 2x AMD EPYC 7742 64-Core Processor on each node
- 512 GB RAM

For our purposes we will use at most 2 nodes of the EPYC partition.

Parallelization strategy

We adopt a sequential fashion:

1. using MPI, initialize P processes
2. each process computes a portion of the image using T OpenMP threads (loop scheduling policy is set to `dynamic`)
3. the master process uses `MPI_Gatherv` to collect the results

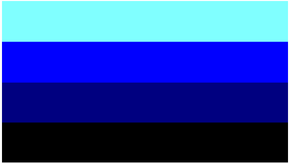


Figure: Suppose to have 4 processes. Each process will compute a portion of the image.

Let $N = n_x \times n_y$ be the total number of pixels. Each process will compute approximately N/P pixels.

Experimental setup

MPI scaling

Set $T = 1$ and vary

$P = 1, 2, 4, 8, \ldots, 112, 128$

- strong scaling:

$$n_x = n_y = 4096$$

- weak scaling:

$$n_x = n_y = 1024 \times \texttt{round}\{\sqrt{P}\}$$

OpenMP scaling

Set $P = 1$ and vary

$T = 2, 4, 6, 8, \ldots, 62, 64$

- strong scaling:

$$n_x = n_y = 4096$$

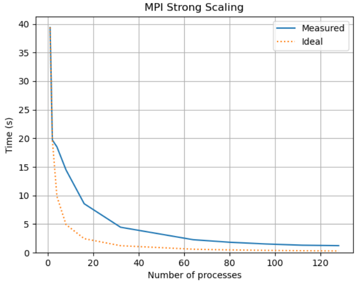
- weak scaling:

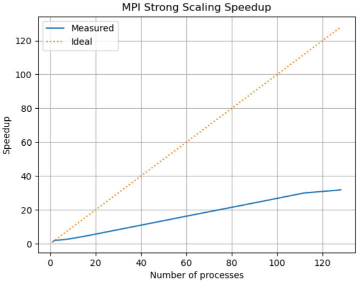
$$n_x = n_y = 1024 \times \texttt{round}\{\sqrt{T}\}$$

Other parameters

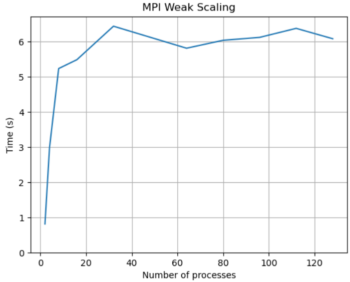
- For MPI, we set `--map-by core` and `--bind-to socket`
- For OpenMP, we set `OMP_PLACES=cores` and no binding policy.

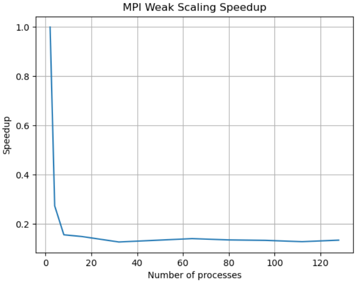
MPI **strong** scaling results



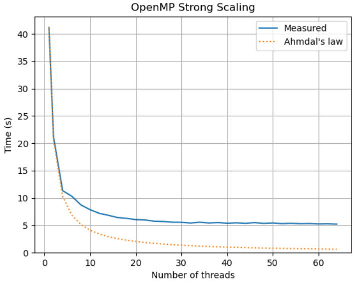


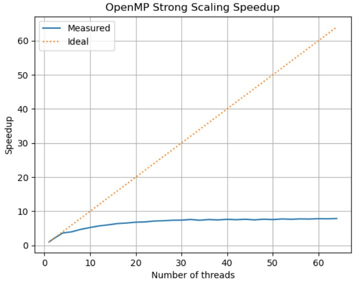
MPI **weak** scaling results



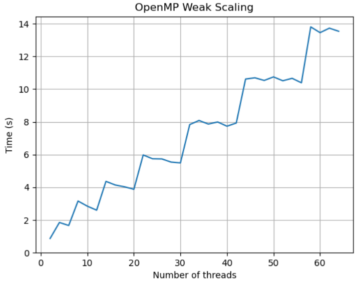


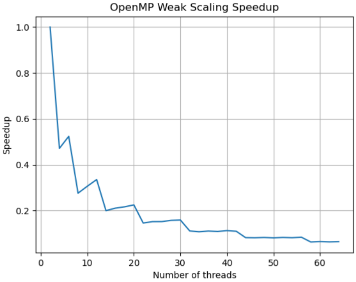
OpenMP **strong** scaling results





OpenMP **weak** scaling results





Conclusions

- MPI scaling shows an expected behavior for both strong and weak scaling
- OpenMP strong scaling shows that we reach a bottleneck

OpenMP issues:

- load imbalance
- false sharing

Improvements:

- smarter load balancing strategy
- exploit symmetries
- exploit compiler optimizations