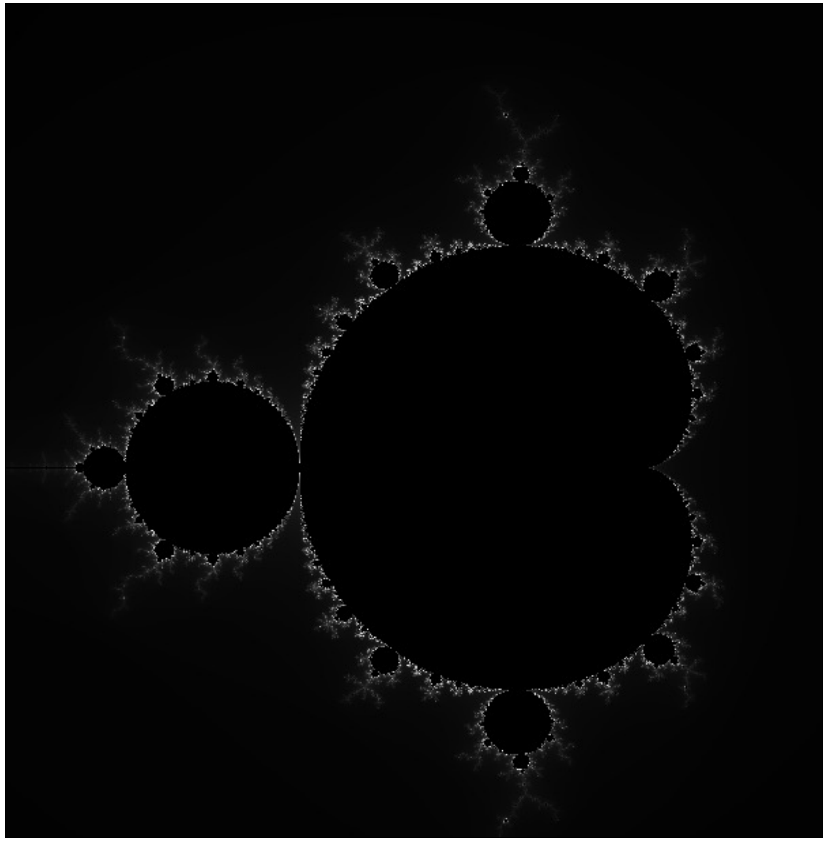
Mandelbrot Set Implementation via Hybrid MPI+OpenMP

Nigam Vishal
University of Trieste
July 2024

Objective



- To implement and analyze a hybrid MPI/OpenMP implementation of the computation of the Mandelbrot set which is generated on the complex plane C by iterating the complex function:

$$f_c(z) = z^2 + c.$$

- To determine the strong and weak scaling of code implemented above

- Share the insights and possible improvements

Experimental Plan

ORFEO CLUSTER

- 12 intel nodes: two equipped with [Xeon Gold 6154] and 10 equipped with [Xeon Gold 6126] cpus
- THIN partition
- 2 nodes -> 48 cores
- Mandelbrot_pgm.c program with Hybrid MPI/OpenMp constructs
- Python script for plots and images
- Slurm jobs for above 2

Parallelization Strategy

➢ **Role of MPI**

- **Master Thread (MPI Rank 0):** Gathers computed chunks from other MPI processes. Generates and saves the final image.

➢ **Role of OpenMP**

- **Parallel Computation**: Parallelizes the nested loops for computing the Mandelbrot set points.
- **Dynamic Scheduling**: Ensures efficient load balancing across threads.

```
# Compile the Mandelbrot program
mpicc -fopenmp -o mandelbrot_pgm mandelbrot_pgm.c
# Run the Mandelbrot program
mpirun -np 4 ./mandelbrot_pgm 1024 1024 -2.0 -1.5 1.0 1.5 1000
```

Parallelization Strategy-2

- **Hybrid Model Efficiency**: Leveraging both MPI and OpenMP maximizes resource utilization.
- **Scalability**: Effective distribution of workload across nodes and cores.
- **Dynamic Scheduling**: Adapts to variable workloads, improving performance. Usage of the #omp parallel for schedule(dynamic) directive ensures efficient load balancing among threads.
- **Modular Design**: Functions for computation and image generation are clearly separated.

Scaling Setups

- **OpenMP**

set process=1 ,vary the threads from 1 2 4… 32 64,boundaries, the maximum number of iterations and base size of the computation grid.

- Strong Scaling:n_x=n_y=1000

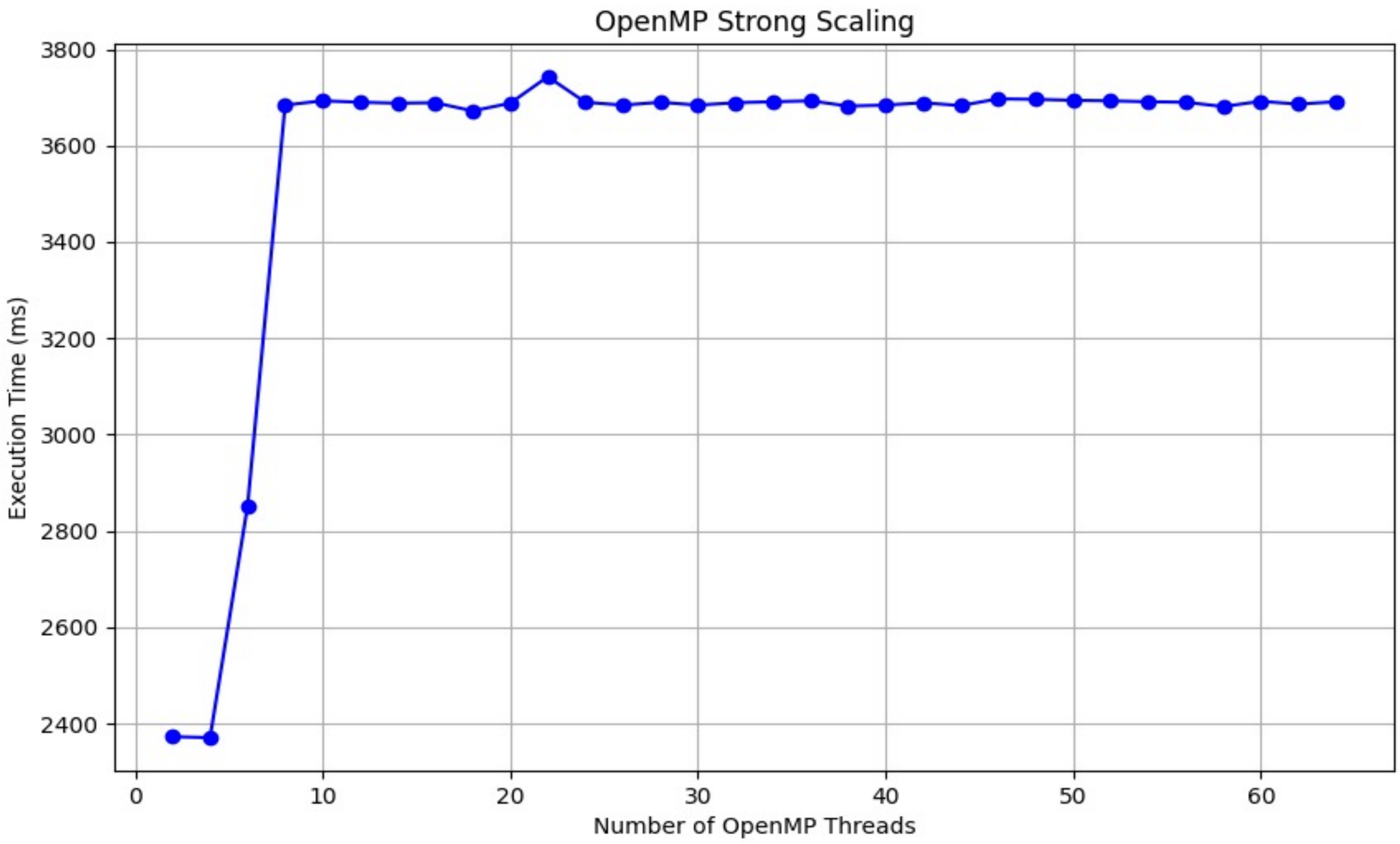
- Weak Scaling: problem size proportionally scaled based on the integer square root of the number of threads

- **MPI**

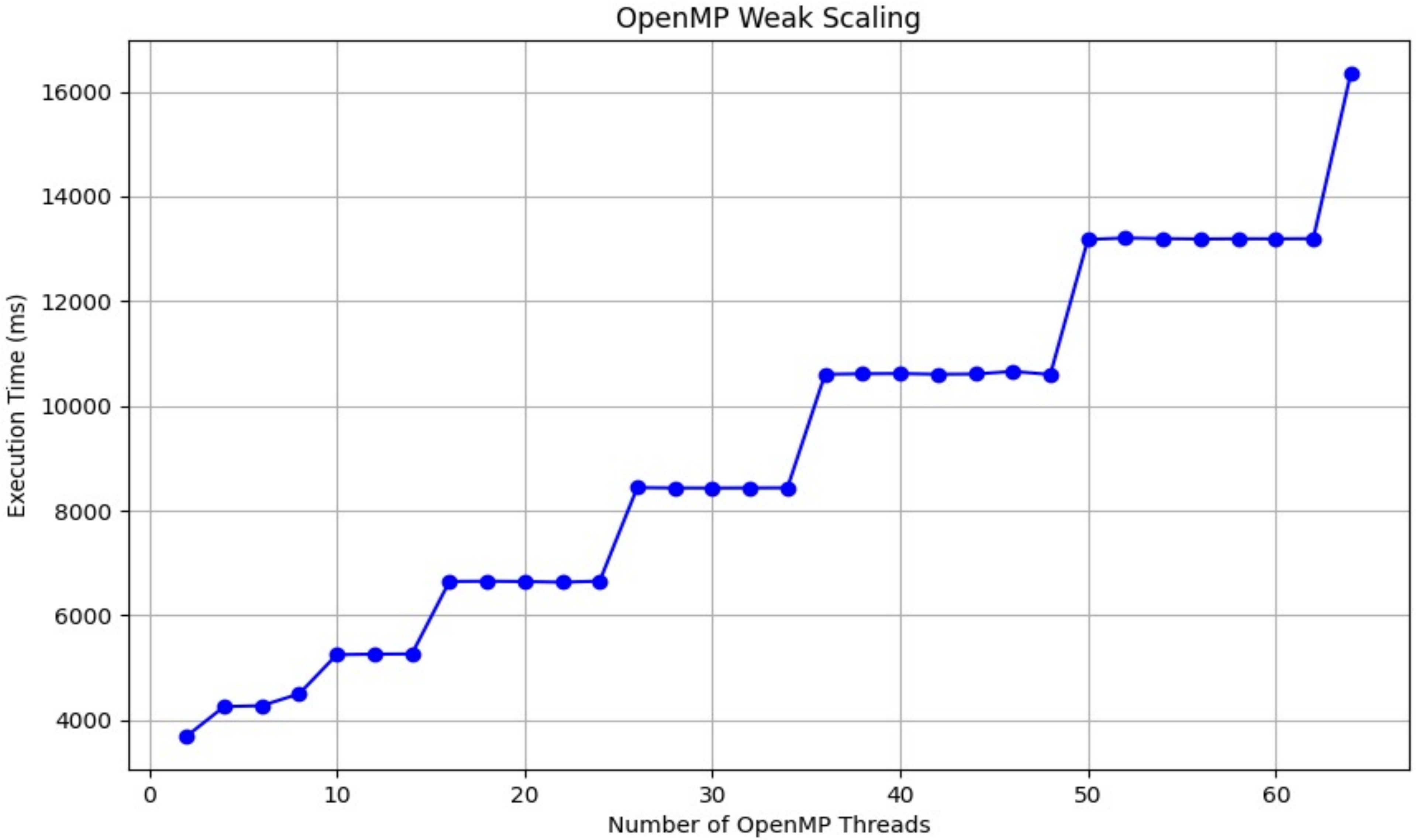
set num_threads=1 and vary the processes from 1 2 4… 32 64

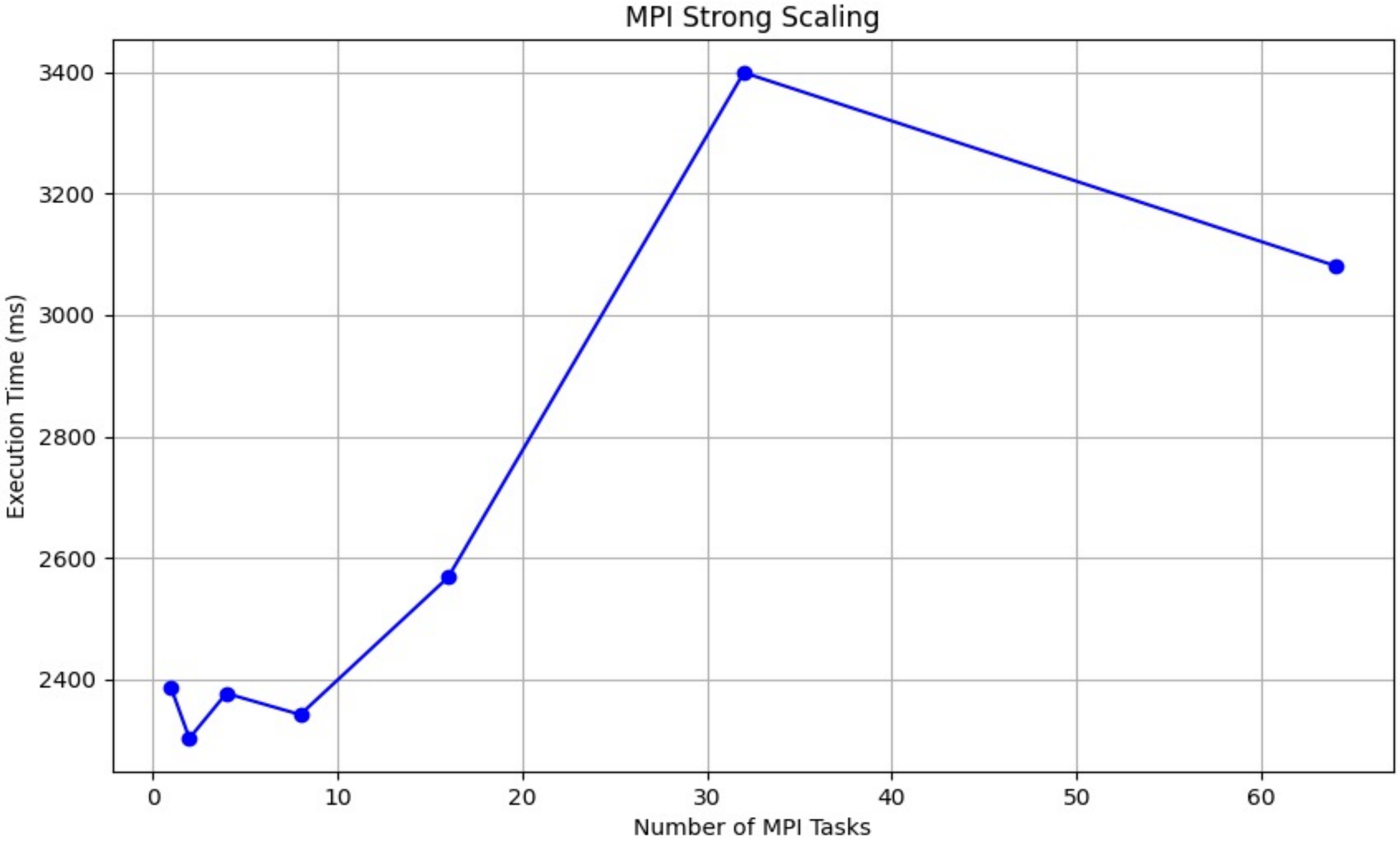
- Strong Scaling: n_x=n_y=1000

- Weak Scaling: problem size proportionally scaled based on the integer square root of the number of tasks

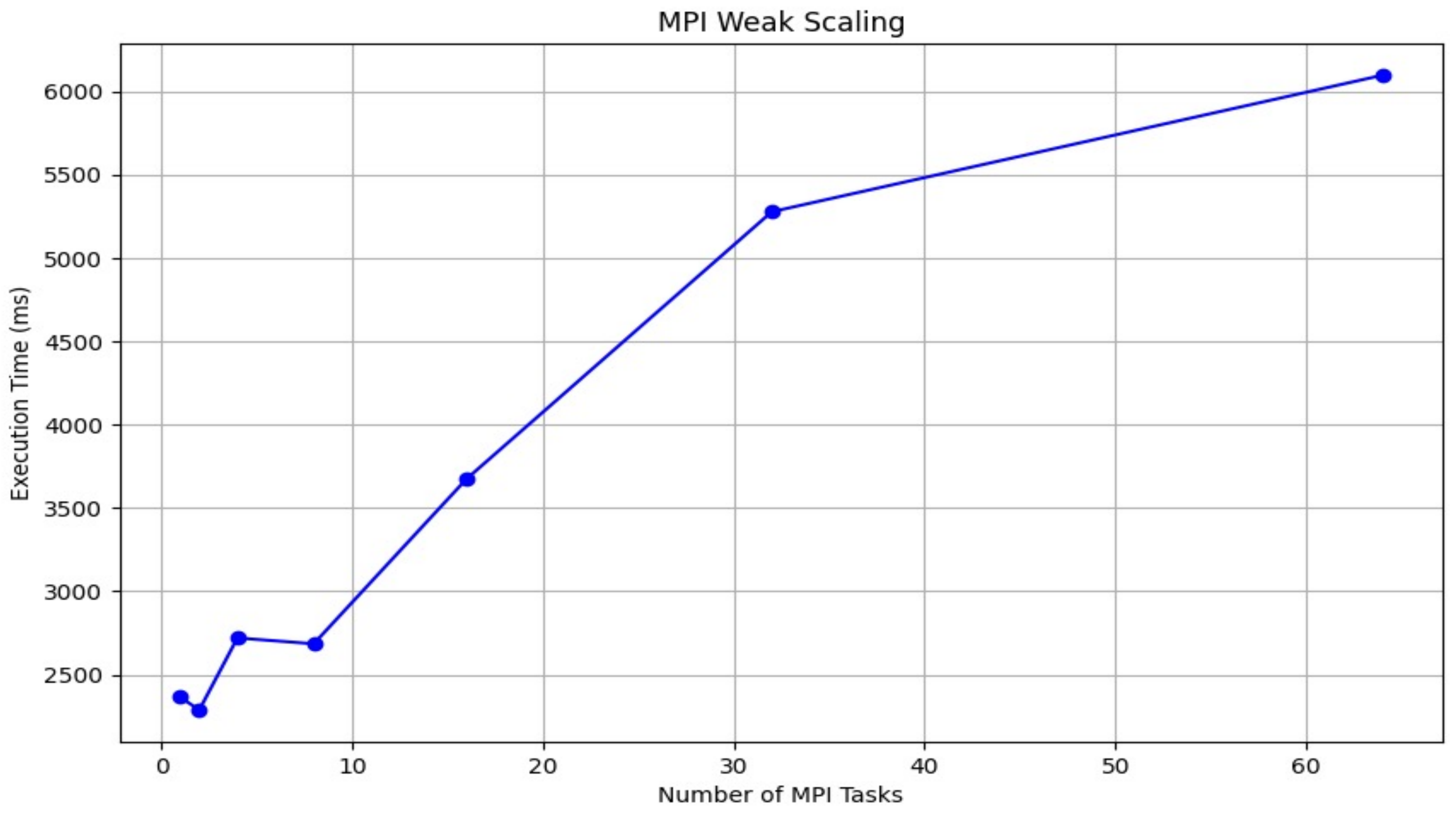


OpenMP Strong Scaling
Execution Time (ms)
3800
3600
3400
3200
3000
2800
2600
2400
0
10
20
30
40
50
60
Number of OpenMP Threads


OpenMP Weak Scaling
Execution Time (ms)
Number of OpenMP Threads
4000
6000
8000
10000
12000
14000
16000
0
10
20
30
40
50
60



MPI Strong Scaling
Execution Time (ms)
3400
3200
3000
2800
2600
2400
0
10
20
30
40
50
60
Number of MPI Tasks



MPI Weak Scaling
Execution Time (ms)
6000
5500
5000
4500
4000
3500
3000
2500
0
10
20
30
40
50
60
Number of MPI Tasks

Conclusion

➢ OpenMP Issues:

- **Load Imbalance**: Some threads finish their tasks earlier than others, leading to uneven workload distribution and underutilization of resources.
- **False Sharing**: Multiple threads writing to variables on the same cache line can cause significant performance degradation due to cache coherence traffic.

➢ Improvements:

- **Load Balancing Strategy**: Implement dynamic or guided scheduling to ensure more even distribution of work among threads.
- **Compiler Optimizations**: Use advanced compiler flags and directives for parallel optimization, such as loop unrolling and vectorization, to enhance execution efficiency.