

PLSPS 2025 Assignment - Report

Aryan - MUKHERJEE - tmv834

September 2025

Contents

1 Design **2**

2 Results **2**

3 Analysis **5**

4 Conclusion **7**

1 Design

Overview

The Parallel Mandelbrot visualization is implemented in Julia with a parallel replicated workers model. The sequential path in `sequential.jl` computes F zoom frames, each an $N \times N$ grid. For each frame, the complex plane bounds are derived from the center, zoom factor α, and half width δ. Each pixel runs the escape time iteration up to M steps and stores a smoothed iteration count for coloring.

Parallel Replicated Workers

The parallel version in `parallel.jl` uses `Distributed` with a replicated workers model. Each worker loads the same sequential code via `@everywhere include("sequential.jl")`, so it uses the `Single Program Multiple Data (SPMD)` model. Work is decomposed by frame index, which keeps the per task payload large enough to amortize communication overhead. Furthermore, the frames are independent of each other. Hence the computation is embarassingly parallel.

A master process creates a remote job channel that holds frame indices and a result channel for completed frames. Each worker repeatedly pulls a frame index, computes it with `compute_frame`, and pushes the result back. The master collects results and writes them into an array. This dynamic scheduling balances uneven frame costs while avoiding write contention, since only the master writes into the output array.

The implementation measures wall clock time with `@elapsed`. While, the benchmark is repeated multiple times, only the minimum time is taken.

Strong Scaling Design

Strong scaling keeps the workload fixed and varies the number of workers. The benchmark suite uses three fixed workloads, $(N, F) \in \{(1024, 30), (2048, 5), (512, 120)\}$ with the default M, (iterations per pixel) value. For each workload, runs are executed for $P \in \{1, 2, 4, 8, 16, 32\}$ plus a sequential baseline, and the minimum runtime across 5 runs is reported.

Weak Scaling Design

Weak scaling grows the workload with processor count. The same three base workloads are used, but the problem size, defined as the resolution per frame, is increased by a factor of $\sqrt{P}$. Since each frame has N^2 pixels, this makes total pixel work scale roughly with P. The number of frames F is kept constant for these measurements. An alternative weak scaling approach for the animation might be to keep N fixed and increase the number of frames produced, which effectively scales work through a higher total frame count.

2 Results

Strong Scaling Experiment Results

This section summarizes the strong scaling runs for three problem sizes. The tables list runtime, speedup, and efficiency for each processor count.

P	T	Speedup S(P)	Efficiency E(P)
0	10.188733	1.00	1.00
1	10.336193	0.9857	0.9857
2	5.818260	1.7512	0.8756
4	3.266847	3.1188	0.7797
8	1.807634	5.6365	0.7046
16	1.232638	8.2658	0.5166
32	1.564289	6.5133	0.2035

Table 1: Results for (N, F) = (1024, 30)

P	T	Speedup S(P)	Efficiency E(P)
0	6.786493	1.00	1.00
1	6.915843	0.9813	0.9813
2	4.196994	1.6170	0.8085
4	2.943729	2.3054	0.5764
8	1.612924	4.2076	0.5259
16	1.630605	4.1619	0.2601
32	2.027474	3.3473	0.1046

Table 2: Results for (N, F) = (2048, 5)

P	T	Speedup S(P)	Efficiency E(P)
0	11.918628	1.00	1.00
1	12.068930	0.9875	0.9875
2	6.640514	1.7948	0.8974
4	3.673055	3.2449	0.8112
8	1.878813	6.3437	0.7930
16	1.137931	10.4739	0.6546
32	0.827992	14.3946	0.4498

Table 3: Results for (N, F) = (512, 120)

Execution Time, Speedup, and Efficiency Plots

Figure 1 shows the runtime trend with more processors, while Figures 2 and 3 summarize scaling behavior.

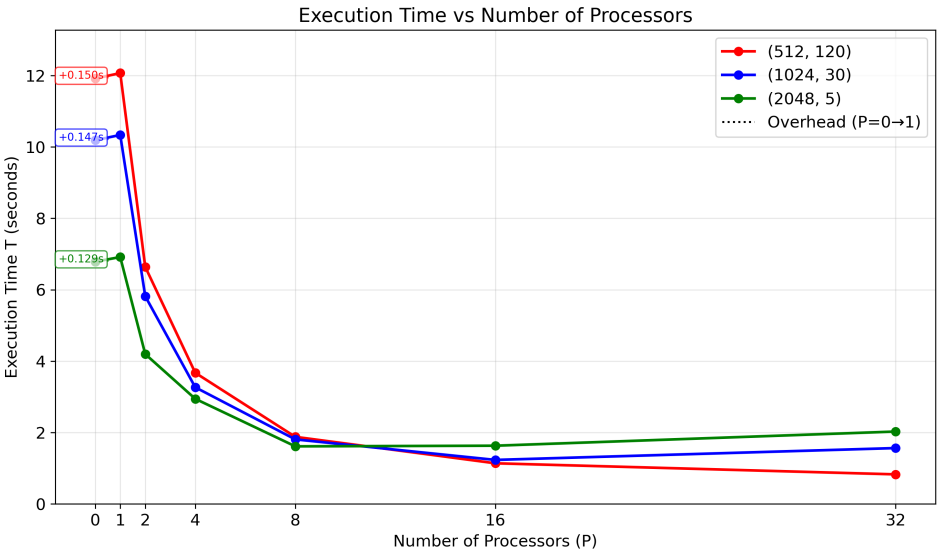


Figure 1: Execution time vs number of processors for each problem size

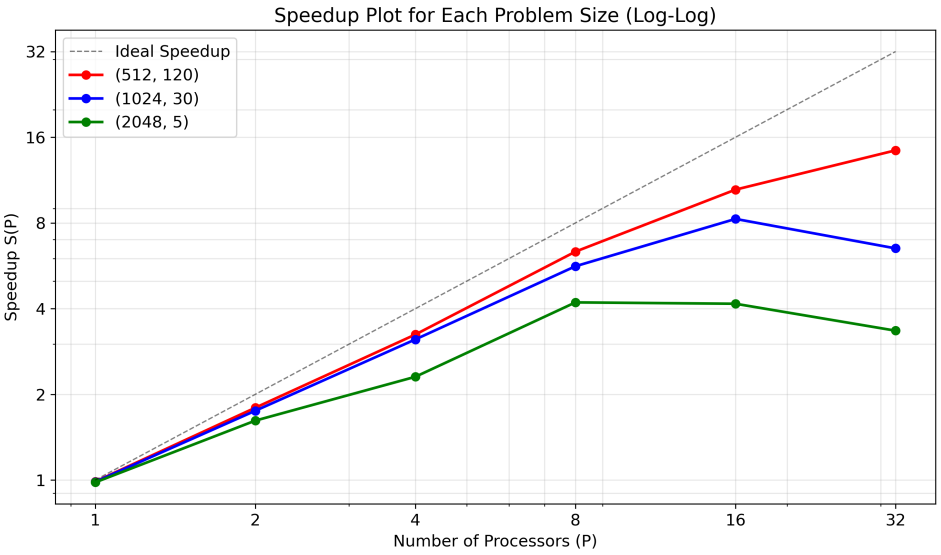


Figure 2: Speedup for each problem size

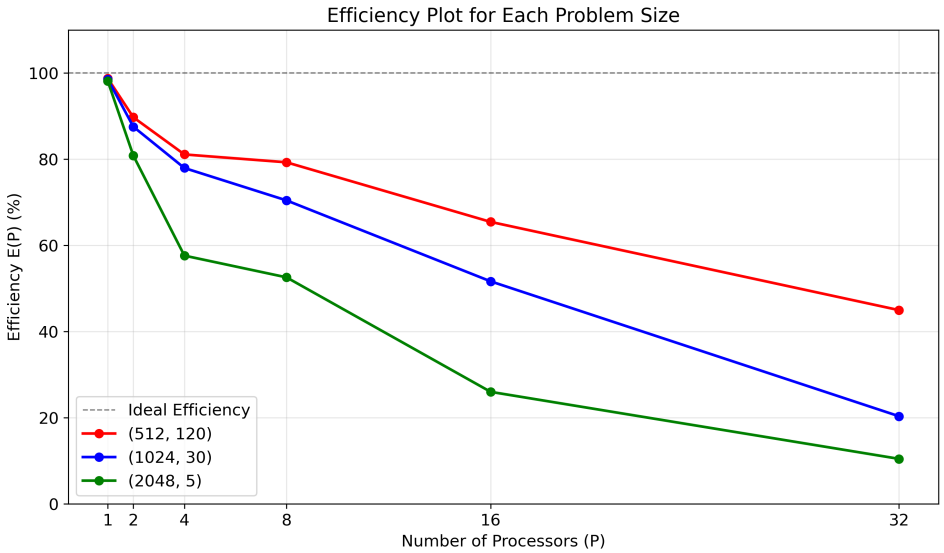


Figure 3: Efficiency for each problem size

Weak Scaling Experiment Results

Weak scaling measures how runtime changes when the problem size grows with processor count. Figure 5 shows the weak scaling trend for the selected configurations.

P	N	F	M	T
1	512	120	100	11.914119
1	1024	30	100	10.186431
1	2048	5	100	6.787158
2	724	120	100	13.241989
2	1448	30	100	11.625105
2	2896	5	100	8.467436
4	1024	120	100	13.999179
4	2048	30	100	13.155066
4	4096	5	100	11.477102
8	1448	120	100	14.976575
8	2896	30	100	14.698266
8	5793	5	100	15.215415
16	2048	120	100	18.136630
16	4096	30	100	20.467297
16	8192	5	100	26.133689
32	2896	120	100	29.662340
32	5793	30	100	44.973313
32	11585	5	100	52.049070

Table 4: Weak scaling results for the Mandelbrot benchmarks

3 Analysis

The overhead for $P = 1$ (Parallel implementation with 1 Process) w.r.t baseline ($P = 0$ / Sequential Implementation), is quite low.

The strong scaling results show clear runtime reductions as processor count increases from 1 to 8, with diminishing returns beyond 16 processors. For $(N, F) = (512, 120)$ the speedup con-

tinues to rise to 32 processors, while the two larger problems show a slowdown at 32 processors. This indicates overheads such as communication, scheduling, and load imbalance beginning to dominate at higher core counts.

Problem size impacts scalability. The smallest workload shows the best speedup at high processor counts, which suggests a larger portion of the work remains parallel for that configuration. In contrast, the larger configurations show earlier saturation and lower efficiency.

Figure 4 compares the measured speedups to Amdahl's Law fits. The fitted serial fractions align with the observed efficiency drop at higher processor counts, especially for $(N, F) = (2048, 5)$.

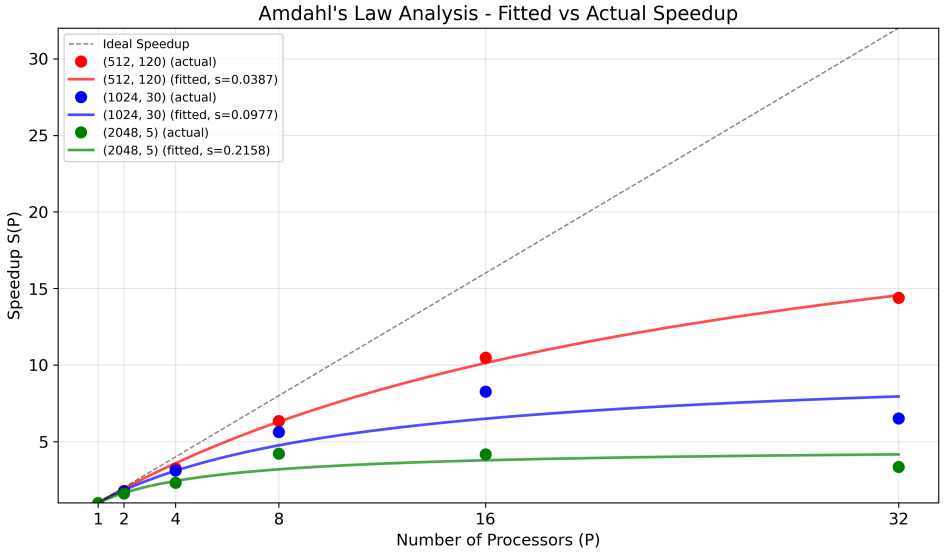


Figure 4: Amdahl's Law analysis comparing fitted and actual speedup

While it is known that a workload comprising of computing independent frames, like in this Mandelbrot animation, is embarassingly parallel and should show strong scaling, the aim with this section is to show this formally. Weak scaling deviates from the ideal constant runtime line. The runtime increases as P grows for all three base cases, rising from about 10 to 12 seconds at $P = 1$ to roughly 30 to 52 seconds at $P = 32$. This suggests fixed overheads, memory pressure, and communication costs grow faster than the added work per processor under the current weak scaling setup. In these weak scaling runs, the workload growth is achieved by increasing the resolution N by a factor of $\sqrt{P}$ since each frame has N^2 pixels.

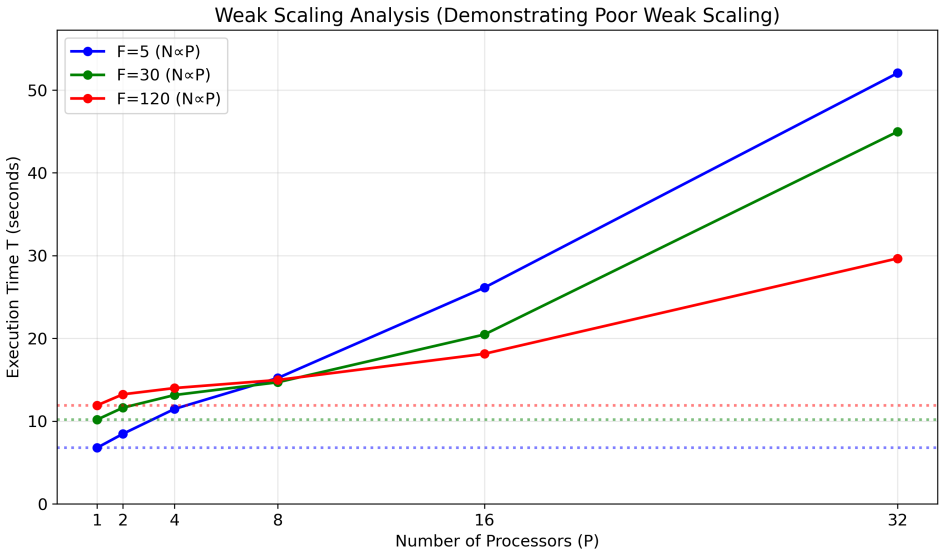


Figure 5: Weak scaling analysis for the Mandelbrot computation

4 Conclusion

The replicated workers approach provides clear gains over the sequential baseline for moderate processor counts. Strong scaling is effective up to 8 or 16 processors, with the best overall speedup in the $(512, 120)$ workload. Efficiency drops for higher processor counts, which indicates growing overheads and reduced parallel utilization.

Weak scaling analysis shows that runtime is not constant as the problem size grows with the number of processors.

During implementation, the main issues were overhead from worker startup and channel communication, plus uneven frame cost that can leave some workers idle. The dynamic job queue mitigates this to some extent, but the results suggest there is still room for improvement in workload balance and communication cost.