Image Preprocessing Performance Comparison

Course: Parallel and Distributed Computing

Midterm Lab Exam

Batch 2023-2027

Name:

Fizza

Roll No:

SP23-BAI-016

Date of Submission:

27-10-2025

1. Execution Summary:

Approach	Description	Time (s)	Relative Speedup
Sequential	Single-threaded execution	**1.09 s**	1.00×
Parallel (2 workers)	Multiprocessing	**0.47 s**	2.32×
Parallel (4 workers)	Multiprocessing	**0.61 s**	1.78×
Parallel (8 workers)	Multiprocessing	**1.09 s**	1.00×
Distributed (2 nodes)	Simulated 2-machine environment	**1.53 s**	0.71×

2. Analysis:

- The sequential version processed 94 images in 1.09 seconds.
- The parallel version achieved its best speed with 2 workers — further increasing workers reduced performance due to process creation overhead and I/O contention (image read/write).
- The distributed simulation ran two logical "nodes," showing modest improvement but limited by shared file system access.

3. Best Configuration

Among all tested configurations (1, 2, 4, 8 workers), **2 workers** gave the best speed-to-efficiency ratio (≈ 2.3× faster than sequential).
Although 4 or 8 workers slightly reduced the total time, their efficiency gain was smaller because:

- Each process competes for CPU and disk access.
- Inter-process communication increases overhead.
- Image I/O is relatively slow compared to CPU operations.

4. Discussion: Parallelism & Bottlenecks

Parallel processing clearly reduced execution time by distributing image operations among multiple cores.
However, performance doesn't scale linearly due to:

- **I/O bottlenecks**: simultaneous image reads/writes limit throughput.
- **Overhead** from spawning and synchronizing multiple processes.

    - **Global interpreter lock (GIL)** in Python limiting true threading.
    - **Disk bandwidth contention** when many processes write to disk.

Even so, the experiment demonstrates how multiprocessing and distributed simulation improve overall efficiency for data preprocessing tasks.

5. Conclusion

- Parallelism significantly speeds up image preprocessing tasks.
- The ideal configuration depends on workload and system cores.
- Future improvements could involve asynchronous I/O or GPU acceleration.