IMAGE DOWNSCALING

CS3658 – High Performance Computing

Repo Link: https://github.com/levxn/Image_downscaling_HPC

Levin MS - 21011101065
Akshay Shah - 21011101016

PARALLELISM IMPLEMENTED

Pragma Directive: The #pragma omp parallel for collapse(2) collapses the two nested loops (i and j) into a single loop, and each iteration of this loop will be executed by a separate thread.

Workload Division: The workload is divided among multiple threads in a data-parallel manner. Each thread will handle a subset of the iterations of the collapsed loop.

- The outer loop (i) iterates over the rows of the downscaled image (new_height).
- The inner loop (j) iterates over the columns of the downscaled image (new_width).
- Each iteration of the collapsed loop corresponds to processing a single pixel in the downscaled image.

Independent Subtasks:

- Within each iteration of the collapsed loop, independent subtasks are carried out to calculate the average color values for each pixel in the downscaled image.
- The summation of color values within each block (nested loops k and l) is an independent subtask. Since each iteration operates on different pixels within the block, and there are no dependencies between iterations, these computations can be executed concurrently.
- Additionally, the assignment of the average color values to the corresponding pixel in the downscaled image (output->data) is an independent subtask for each pixel.

Concurrency:

- As the collapsed loop is executed, each thread works on its assigned subset of iterations concurrently with other threads.
- The work of summing up color values within blocks and assigning average values to pixels can be performed simultaneously by multiple threads, maximizing CPU utilization and reducing overall execution time.

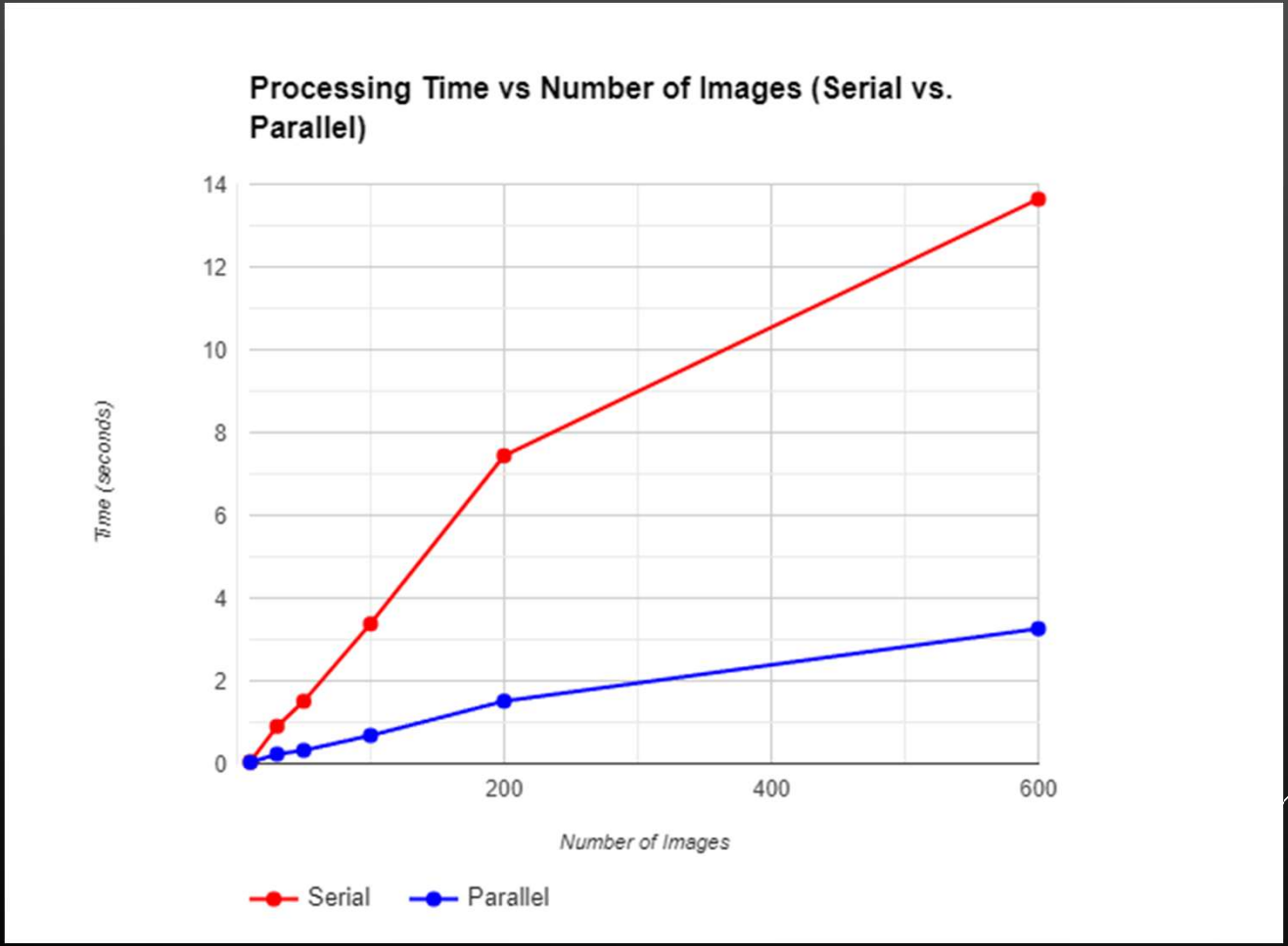
TABLE 1.1(NO.OF.IMAGES VS TIME)

No of Samples(n)	Serial Time (sec)	Parallel Time (sec)	Speedup
10	0.045	0.03	1.5
30	0.9008	0.23	3.9
50	1.5070	0.32	4.7
100	3.3764	0.68	4.96
200	7.439	1.51	4.92
600	13.6388	3.26	4.18

TABLE 1.2 (SCALING FACTOR VS TIME)

Scaling factor(n)	Serial Time (sec)	Parallel Time (sec)	Speedup
2	13.638	3.26	4.18
3	12.2624	3.03	4.04
4	12.4286	2.91	4.27
5	11.8142	2.83	4.174
6	11.8349	2.82	4.196
7	11.4005	2.85	4.0
8	11.1502	2.8	3.98
9	11.0248	2.82	3.9
10	10.8756	2.78	3.91
20	11.3877	2.79	4.08
30	10.6724	2.98	3.58
50	10.2483	2.8	3.67
80	10.4266	2.79	3.73
100	10.0296	2.67	3.75

GRAPH 1(NO.OF.IMAGES VS TIME)



GRAPH 1(SCALE FACTOR VS TIME)

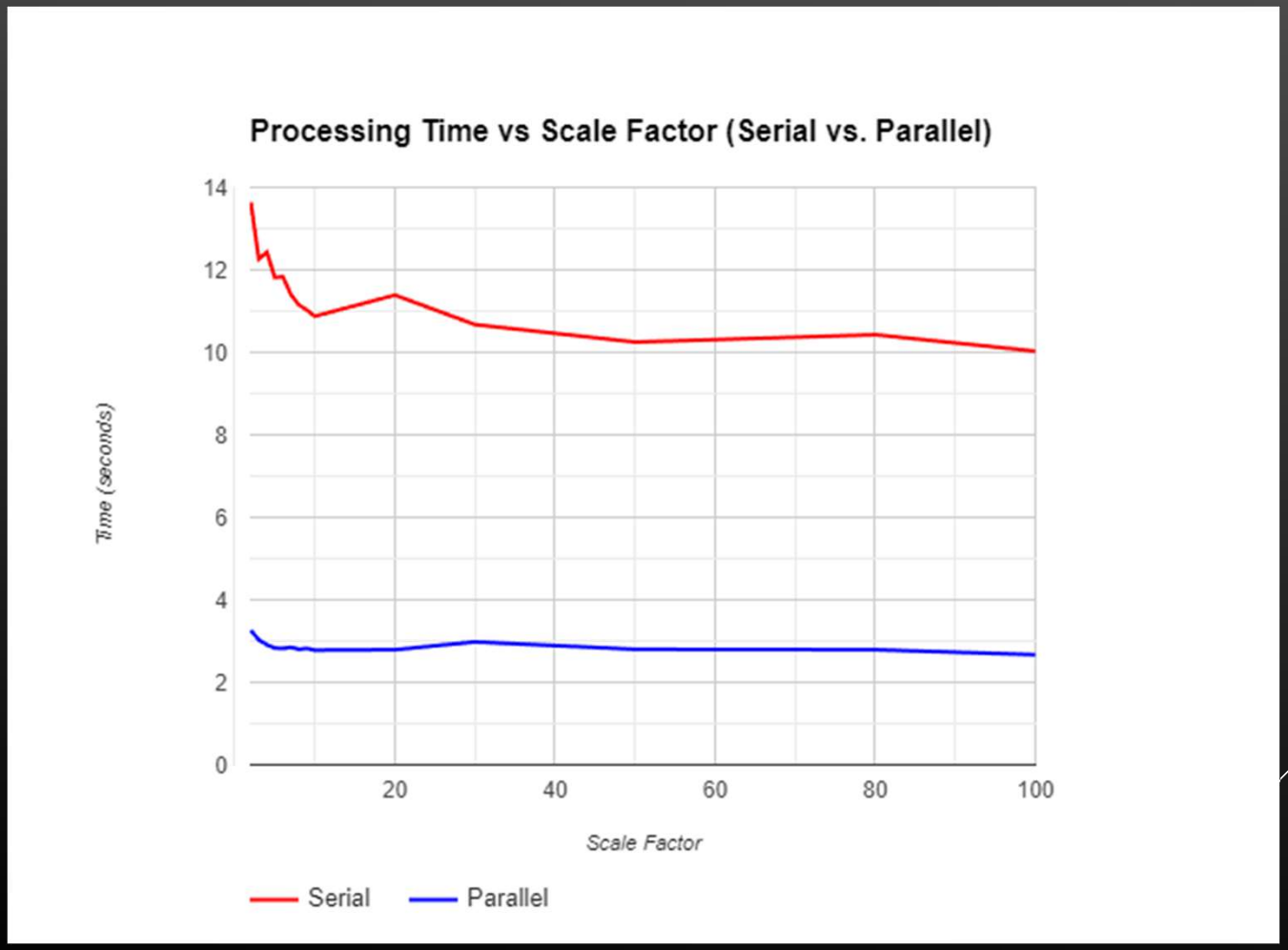
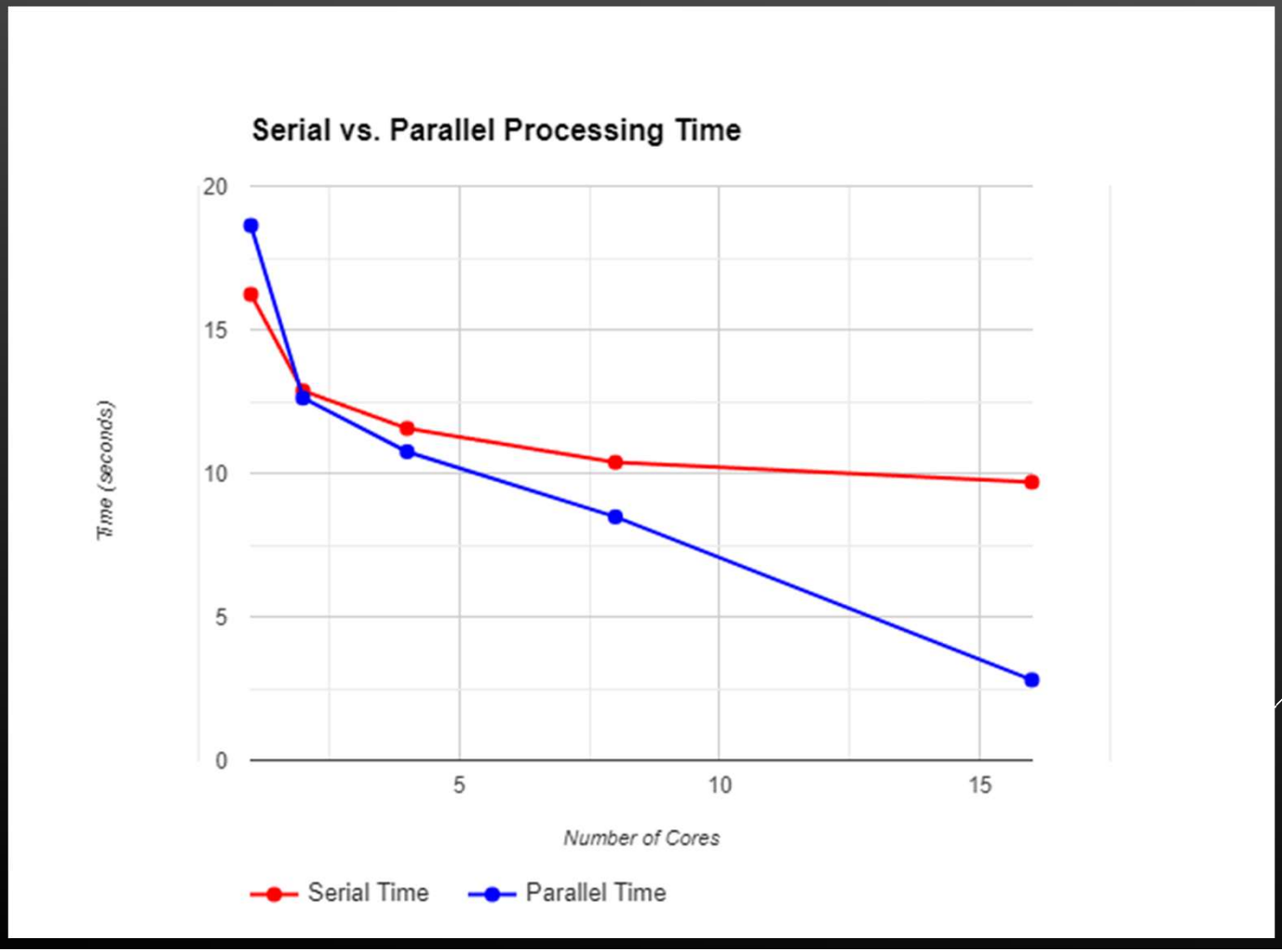


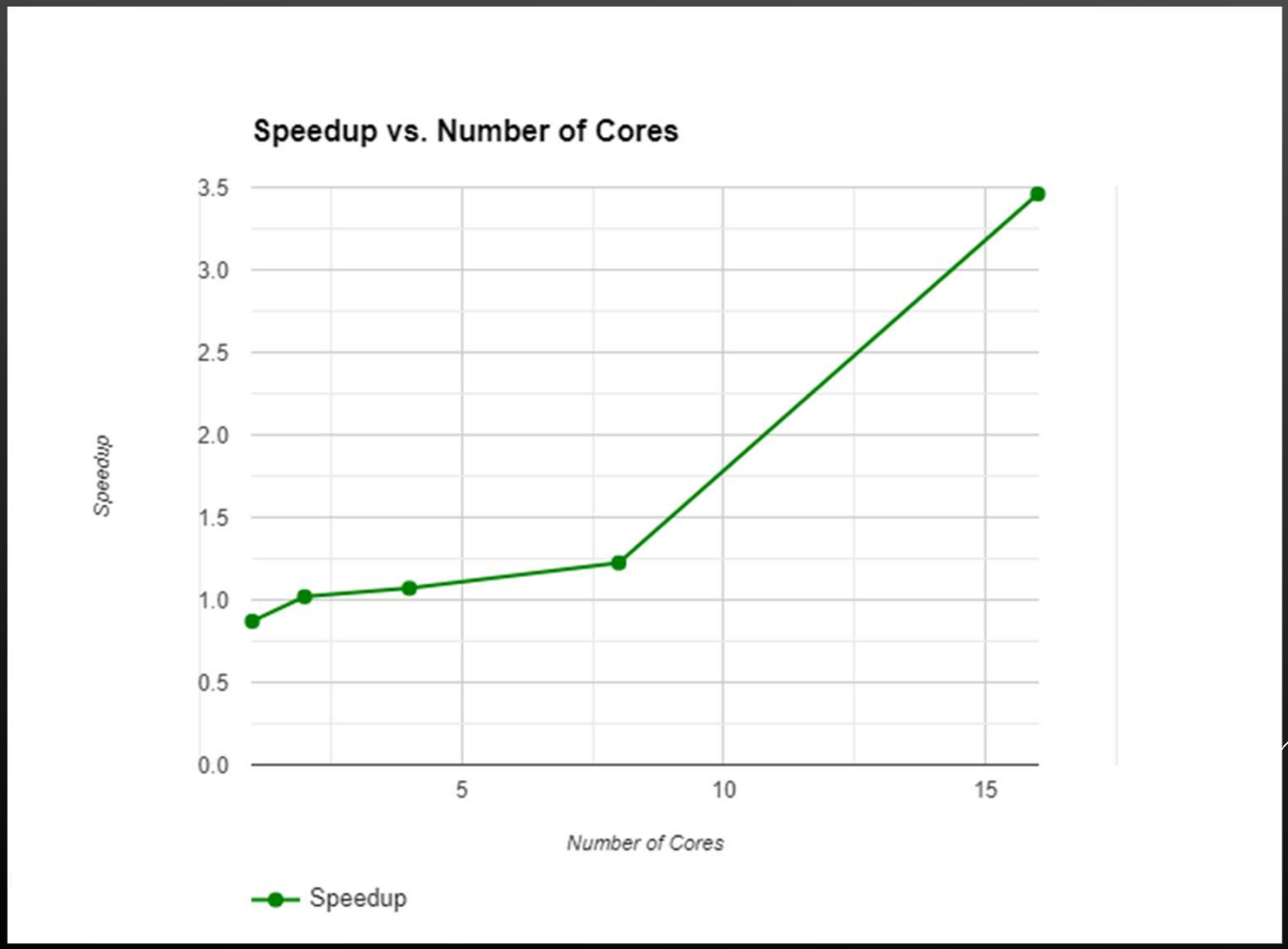
TABLE 2

cores	Serial Time (sec)	Parallel Time (sec)	Speedup	Parallel Efficiency
1	16.23	18.63	0.87	87
2	12.87	12.62	1.019	50.95
4	11.56	10.75	1.07	26.75
8	10.38	8.48	1.224	15.3
16	9.69	2.8	3.46	21.62

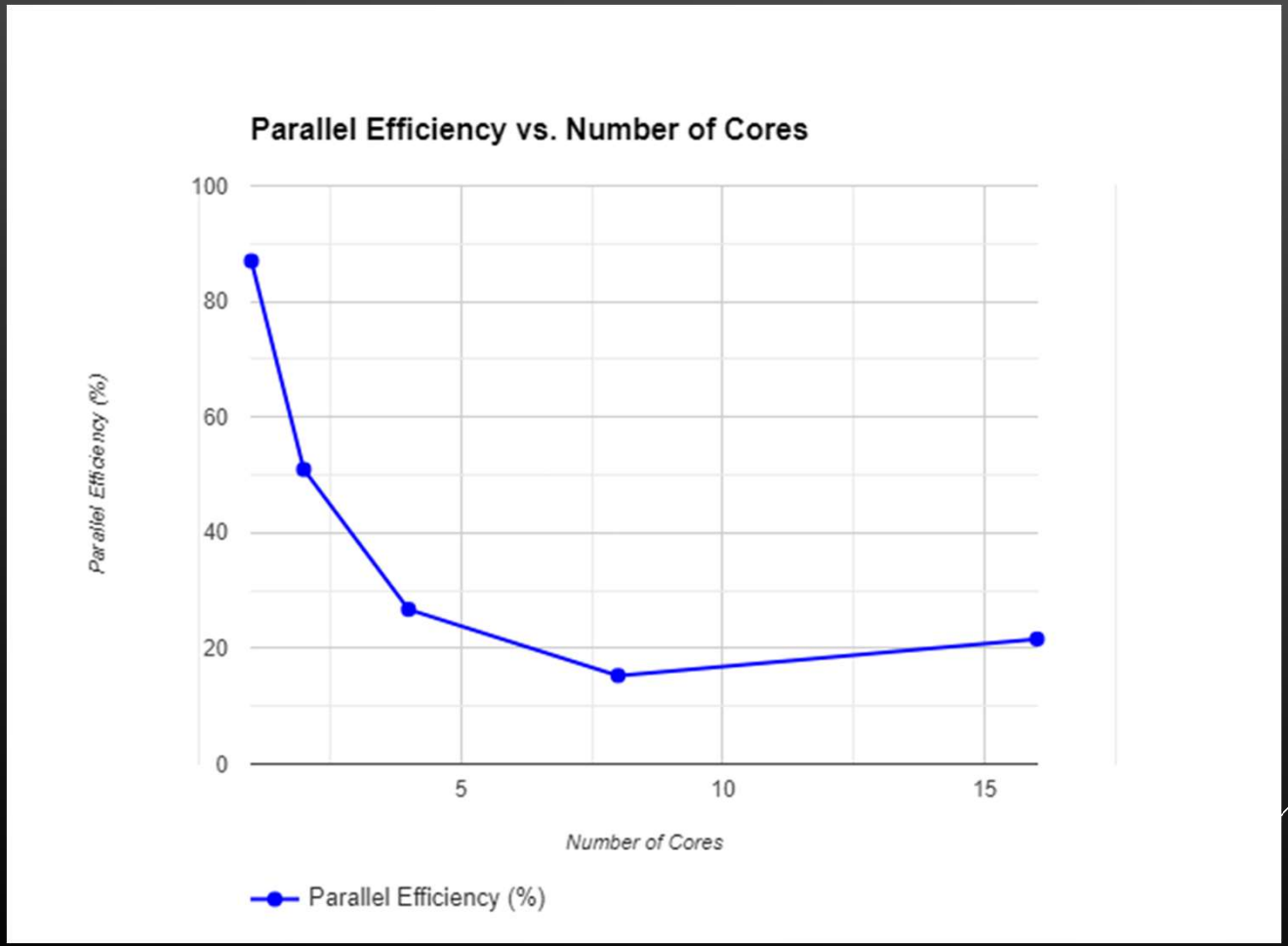
GRAPH 2.1(CORES VS TIME)



GRAPH 2.2(SPEEDUP)



GRAPH 2.3(PARALLEL EFFICIENCY)



SIGNIFICANT SPEEDUP THROUGH PARALLEL PROCESSING

- Parallel processing provides a powerful mechanism for accelerating the image downscaling process by distributing the computational workload across multiple threads. By partitioning the task, such as pixel processing and memory access optimization, among these threads, the overall processing time can be significantly reduced. Techniques like load balancing ensure that each thread receives a comparable workload, while minimizing synchronization overhead helps to maintain high parallel efficiency.
- Furthermore, exploiting hardware features such as SIMD (Single Instruction, Multiple Data) instructions allows for simultaneous processing of multiple data elements, such as color channels in pixels, enhancing computational throughput. Additionally, optimizing I/O operations, such as file reading and writing, can further expedite the process by reducing the time spent on data transfer. Through careful experimentation and profiling, developers can fine-tune the parallelization strategy to suit the specific characteristics of the hardware and workload, unlocking maximum performance gains and achieving efficient resource utilization.
- Load Balancing: Implement load balancing techniques to evenly distribute the workload among threads, ensuring that each thread has a similar amount of work to perform. This can help to avoid situations where some threads finish their work quickly while others are still processing, thus improving overall efficiency.

INFERENCE

- More CPU cores can improve parallel performance for downsizing multiple images concurrently.
- However, the degree of improvement depends on factors such as parallelization efficiency and system overhead.
- Parallel processing can expedite the downsizing of multiple images simultaneously.
- But the extent of acceleration varies based on image characteristics and resource utilization efficiency.
- Decreasing Execution Time with Increasing Number of Workers