Parallel and Distributed Computing - Fall 2025

Parallel Edge Detection using the Sobel Operator

Phase 2

By

Habiba Emad 202202091 Wessam Mohamed 202201656 Maya Tarek 202201864
Rawan Wagih 202201168

GitHub Link: https://github.com/mayatarek/image-processing
Date: 11/30/2025

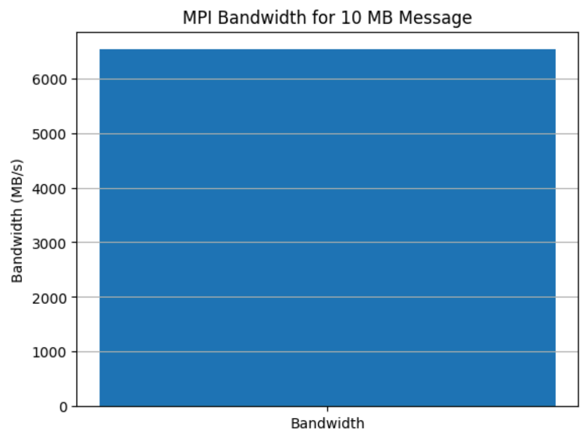
1. Design of decomposition

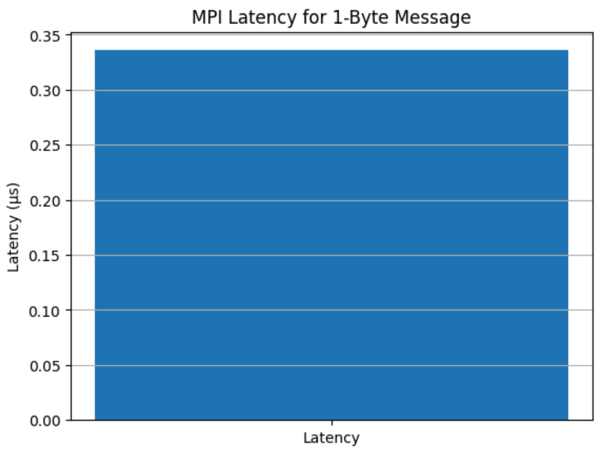
To apply edge detection using Sobel, an image's pixels undergo computations with surrounding neighboring pixels to detect any sharp changes in values, signaling an edge. With that being said, images processed using Sobel can be split evenly as data size is known beforehand, and thus, 1D block splitting was used in our application. Rows were divided mostly evenly among processes, with any extra rows being assigned to lower ranks first. Additionally, because the Sobel computation requires each pixel to be computed against all its direct neighbors (horizontal, vertical, and diagonal neighbors), ghost cells were passed between processes to exchange the vertical and diagonal neighbors.

2. Communication pattern

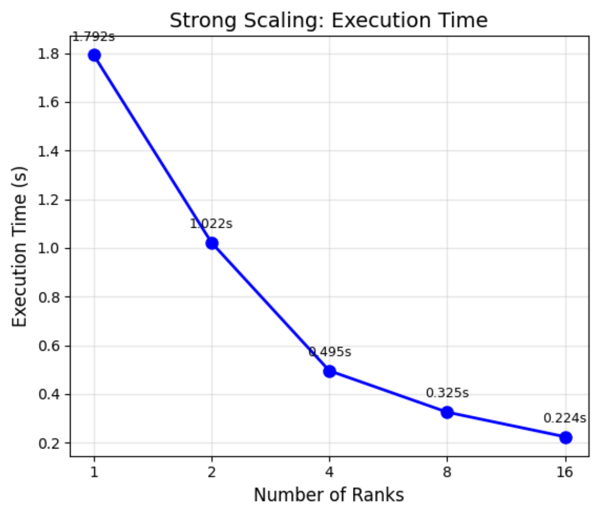
The communication used is non-blocking, as it is more efficient for our program than blocking. Since processes exchange ghost cells and also perform row computations, using a blocking communication will slow down the program. However, with the non-blocking, processes can skip ahead with the computations without the need to wait for all processes to finish their exchange first.

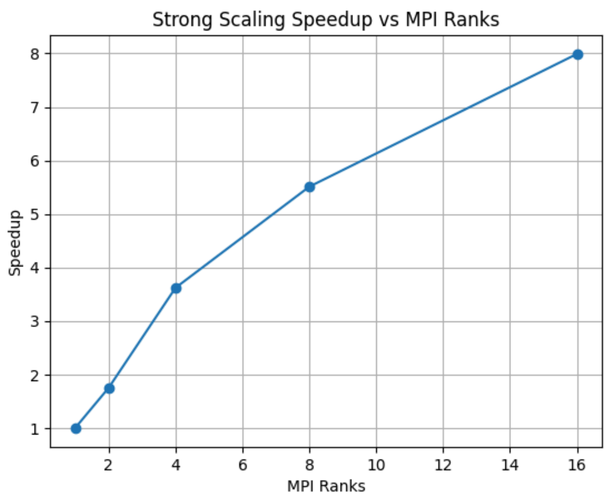
3. Diagrams

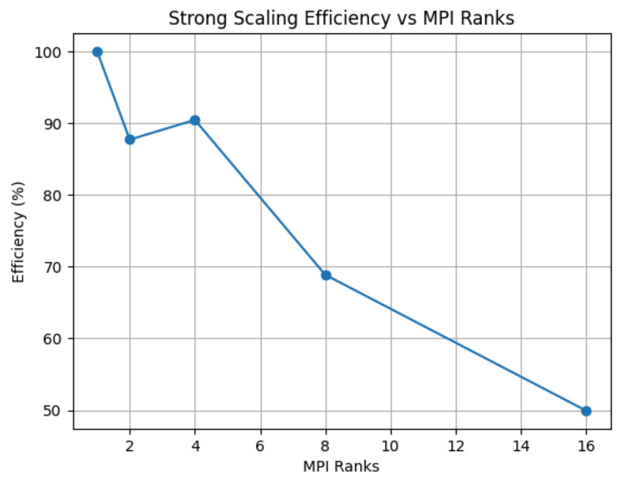


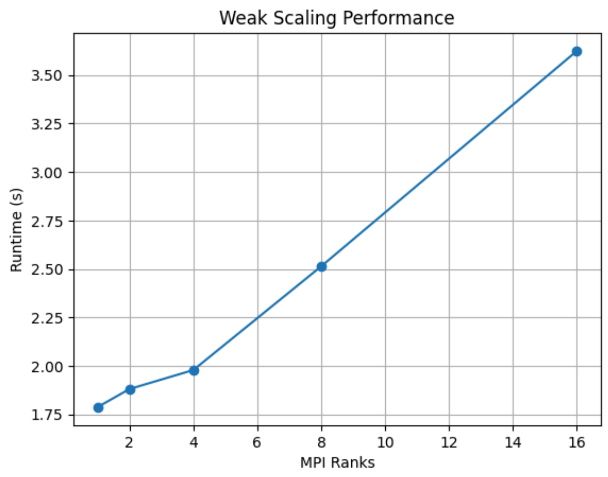


4. Scaling plots









5. Latency/bandwidth results

A)

1. Latency: average latency for 1 byte message: 0.335803 microseconds. This is a very small value.
2. Bandwidth : average bandwidth for 10485760 byte message: 6526.49 MB/s. MPI is efficiently using shared memory(same RAM), meaning all processes are running on the same machine aka single node test.

B) Strong scaling →. Fixed problem size but increasing MPI ranks, this results in measured runtime decreasing. Efficiency decreasing can also be seen with increasing ranks.

Ranks	Time (s)	Speedup	Efficiency %
1	1.79212	1	100
2	1.02173	1.75	87.70
4	0.49540	3.62	90.44
8	0.32527	5.51	68.87
16	0.22422	7.99	49.95

C) Weak scaling → increasing the problem size and processes proportionally, the ideal behavior for this is the runtime stays the same. However, it is observed that in reality, the runtime increases.

Ranks	Time(s)	Image Size
1	1.78696	4000x4000
2	1.88049	8000x4000
4	1.97858	16000x4000
8	2.51352	32000x4000
16	3.62494	64000x4000

6. Discussion of bottlenecks

While using distributed computing helps speed up the program, there are several bottlenecks that limit the speedup.

1. Halo exchange

Each process needs halo cells double the size of their content to compute, causing extra work for a single computation.

2. Communication latency

Because Sobel does not contain complex computations, most of the latency comes from communication overhead between processes.

3. Unequal distribution

While rows are divided mostly evenly, some processes may still receive more rows than others. For example, if an image is divided into 100 rows, and only 3 processes are used, process rank 0 will receive 34 while ranks 1 and 2 will only receive 33 rows.

4. Sequential segments

Code segments such as image loading, distributing tasks among processes, and gathering the image then creating it are all done by one process, or basically be performed sequentially.

7. Reflection: compare OpenMP vs MPI

	OpenMP	MPI
Type	Parallel	Distributed
Memory	Shared between threads	Memory not shared among processes
Speedup	• Strong scaling however limited by thread overhead. • Faster as minimal	• Strong scaling that is limited due to halos and latency

	computation and not communication bound	● Communication bound so slower
Scalability	Only across cores	Across cores and devices

The OpenMP implementation shows a faster speedup, this is because of the latency that MPI observes in its communication. Other computation expensive applications might show otherwise but the Sobel calculations are not heavy, thus the cost of communication slows down MPI greatly. If an image were to be very huge, OpenMP might start to struggle more than MPI as it can not work over multiple devices (less cores).