1 Approach

To parallelize the operation of image convolution, we took the approach of dividing the image into multiple parts, where each available processing node would be responsible for the convolution operation of one part of the image. Every node receives an image with the height of $\frac{height}{\#\ nodes}$ except for the last node, which also receives the remainder of the height division.

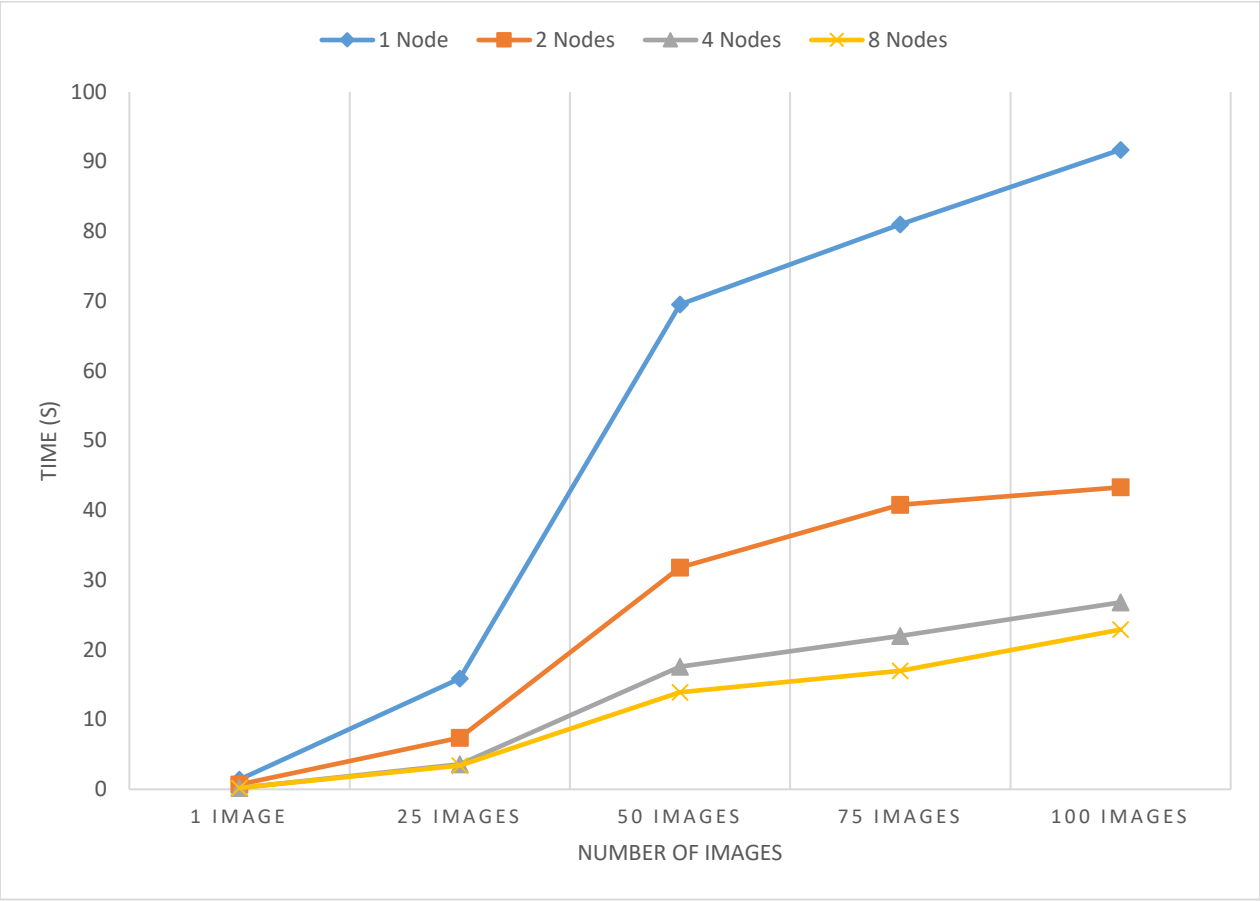
To get the correct convolution output, each node must provide the upper and lower borders of the image parts before and after its own part. Therefore, every node will send its upper border to the previous node and its lower border to the next node. The two exceptions are the first node, which only sends its lower border, and the last node, which only sends its upper border.

Finally, each part of the image is passed to the convolution function alongside the kernel, and the output is sent back to the master node to reassemble the whole image and save it.

2 Results

The program was tested on up to 100 images of different sizes, all in the same order, and the timing of each test run was recorded. The following table and graph summarizes our findings.

	1 Node	*2 Nodes*	*4 Nodes*	*8 Nodes*
1 Image	1.4	0.7	0.2	0.2
25 Images	15.9	7.4	3.6	3.4
50 Images	69.5	31.8	17.6	13.9
75 Images	81	40.8	22	17
100 Images	91.7	43.3	26.8	22.9



3 CONCLUSION

Our findings suggest that, at first, there is a substantial increase in performance as the number of nodes keeps increasing. However at a certain number, the communication overhead slows down the performance increase, and thus adding more nodes doesn't result in a noticeable increase in performance.