1.
    a. Final program description
        i. This program uses MPI to deploy between 1 and 10 processes across an equal amount of nodes. Then it uses OpenMP to speed up Sobel edge detection across 4 threads.
    b. The value of your solution
        i. This program simulates a parallel file system by using the /tmp directory. This is used to simulate a real-life use case in Hollywood. And I chose to do the standard throughout the industry.
    c. numerical methods and algorithms
        i. The numerical method used in this program is convolution. In order to find the edge, we have to use convolute every 3*3 set of pixels by an X and Y filter. The resulting numbers can be summed up, by squaring both sums, adding them, and taking the square root of them, we can determine the magnitude. If the magnitude exceeds a predetermined value, we can assume an edge exists.
    d. parallel programming methods
        i. I utilized MPI in order to split the work between many nodes, this allowed me to create many processes. In each process, I use OpenMP to speed up the edge detection. By doing this I can leverage the I/O benefits of MPI and the processing power of the nodes for threads in OpenMP
2.
    a. Command to run
        i. " make && time mpirun -n 1 -ppn 1 -f c2_hosts ./sobel_MPI_scalable"

Timed Sequential program timing using MPI_Wtime() and the Linux time command

    b. Average time to compute 20 frames of a 4k video: A process doing 20 image processing iterations would take 13.36 seconds
        i. Average time to compute a 4k frame: A process took on average 0.667 seconds
    c. Average time to compute 20 frames of a 4k video: A process doing 20 image processing iterations would take 10.53
        i. Average time to compute a 4k frame: A process took an average of 0.526 seconds
    d. Average time to compute 20 frames of a 4k video: 5 processes doing 4 images each, in a simulated parallel system would take an average of 4.23 seconds
        i. Average time to compute a 4k frame: A process took an average of 1.08 seconds,
    e. Average time to compute 20 frames of a 4k video: 5 processes doing 4 images each, with 4 threads of OpenMP speeding up the calculation. In a simulated parallel system would take an average of 3.5113 seconds
        i. Average time to compute a 4k frame: A process took an average of 0.87 seconds

3.

OpenMP

Amdahl's law parameter	How obtained	description
Sequential portion (% of total)	(1-p)	I first found the parallel portion and subtracted 1 from it (1-0.3656) = 0.6344
Parallel portion (% of total)	I timed all of the runs, then i timed the algorithm, I ran it multiple times and got the averag value of the time it spends doing sobel Or by dividing the parallel time by single threaded	When doing this on one node only running openMP i got an average of %
Number of shared memory cores used and type	Using the nproc command, I was able to get 4 cores Technically we have 2	Although we only have 2 cores, we have have 2 simulated cores inside each core
Number of nodes used in MPI distributed program X # of cores per node	0	We are only doing openMP
Final value used for S, the scaling factor	Speed up = 13.36/10.53 1/((1-0.3656) + (0.3656/4)) = 1.377	S = 1.377 meaning we could speed it up by %37.7 I got 1.26 meaning it sped up by %26. I think that some of my runs got up to that level, but most did not reach that optimal level

MPI

Amdahl's law parameter	How obtained	description

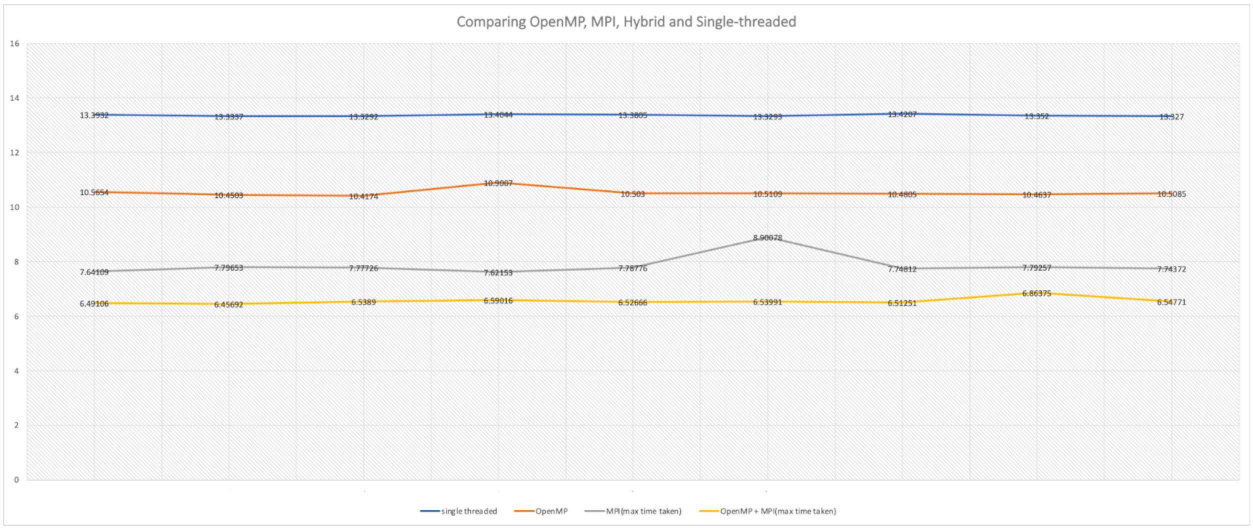
Sequential portion (% of total)	(1-p) = 0.181	I first found the parallel portion and subtracted 1 from it and i got (1- 0.806)
Parallel portion (% of total)	Taking the longest serving process time and then diving it the linux time function. This should return the percentage oif time we do the work vs the settup time 4.81/5.87 = 0.819	0.806
Number of shared memory cores used and type	0	We are doing strictly MPI
Number of nodes used in MPI distributed program X # of cores per node	5	We are only doing 5 MPI nodes
Final value used for S, the scaling factor	1/((1- 0.806) + (0.806/5)) = 2.8	Speed up = 13.36/7.6 I got a speedup of %1.8 While the theoretical is %2.80 I am still having issues with data transfer wich seams to be the most limiting factor

MPI and OpenMP

Amdahl's law parameter	How obtained	description
Sequential portion (% of total)	(1-p) = 0.181	I first found the parallel portion and subtracted 1 from it and i got (1- 0.806)
Parallel portion (% of total)	Taking the longest serving process time and then diving it the linux time function. This should return the percentage oif time we do the	0.806

	work vs the set tup time 4.81/5.87 = 0.819	
Number of shared memory cores used and type	Using the nproc command, I was able to get 4 cores Technically we have 2	Now that we have OmenMP and MPI, I have no idea how to measure speed up
Number of nodes used in MPI distributed program X # of cores per node	5	We are only doing 5 MPI nodes
Final value used for S, the scaling factor	Hybrid = 1/((1-0.806) + (0.806/(4*5))) = 2.8 or 4.26	Speed up = 13.36/6.43 I got a speedup of %2.077

3b)



Verify results:

I utilized GIMP to get an example of what sobel should look like

What my results look like: