Canny Edge Detection Using OpenMP

Identify the Edges in a Given Image As Fast As Possible

Robert Hughes

This project's goal was to take an image and identify all of the edges in the image. In this particular solution the edges in an image of the Lenna Image was found. The goal was to then improve the speed of the project by multi-threading in OpenMP

Image Processing, Cat, Edges

I. Edge Detection and Parallelization

The goal of this project is to create a program that can identify all the edges in an image. To do this a Gaussian filter and a Gaussian Derivative Filter will be used to process the image first. Then a number of different operations were preformed to give the final output image of all the edges in an image. The edges are an important part of the image because they reveal a lot of different information that humans can identify as different objects. We then took this program and then increased the execution time by adding OpenMP

II. Process of Detectio

A. Gaussian and Gaussian Derivative

After the image is read we have to apply a filter to the image this allows for making the pixels closer in intensity. The filter of choice is a Gaussian. To apply the Gaussian filter the Gaussian mask must first be found which is done by using a Gaussian Distribution and using the Formulas found in the equations section of the document to get this array, which will act as the mask. Once the mask and the input image are found then a convolution of the mask and the input image are found this will cause a blurring effect on the image. The Gaussian blur is one of the most fundamental image processing techniques. This process is then repeated but except this time it is done using the Gaussian Derivative Mask. We used three different sigma values to calculate the Gaussian filters.

B. Magnitude and the Gradient of the Image

Once the Gaussian and the Gaussian Derivative are found then the magnitude and the gradient images can be found but the vertical and horizontal intensity changes are needed. First the to find the vertical and the horizontal intensity changes are found by taking the Gaussian and Gaussian Derivative and convolving them. One the vertical and horizontal intensity change the magnitude can be found by using the distance formula for each in point in the image. For the gradient of the image each point is taken and arctan of the vertical over horizontal is used. The results are the images that can be used for the rest of the processes.

C. Non-Max Suppression

Once the magnitude and the gradient are found suppression is preformed. This process takes the image and sets the pixel to off if they are the no the local max in their respective direction. This direction is determined form the gradient image.

D. Hysteresis

Hysteresis is an important step in edge detection because it sets the pixels that are not off to be either a strong edge or a weak edge. It determines this by taking the sorted list of pixel intensity and the value at 90% of the number of pixels will be what makes a strong edge. So if the intensity is above that value it will be turned on to 255. If the intensity is below this value and above 20% of this then the edge is determined to be weak and it will be set to 125.

E. Edge Linking

The Final step requires is to make new image and retain all the strong edges and then retain only the weak edges that are neighbors to strong edges. This will make the final output be crisper.

F. Final Output and Edges

When all of these steps are added together he program is able to successfully identify all the edges in the image. The final image shows the output with what can be identified as a cat just based on the edges.

III. Parallelization

Although completing the canny edge detection is exciting and can give us lots of interesting insight into the image. The goal of this project was to use parallelization techniques to see how the execution time can increase and image the program run faster. These techniques can be applied in a number of different fields and they are important to now to make the overall program better and faster.

A. OpenMP

OpenMP is an API for shared memory systems in C, C++ and Fortran. In our case we are using the C variant in a linux based system. This API includes many libraries and functions that we can take advantage of. OpenMP works in a shared memory and is used for a nodes that can support multiple threads. In comparison MPI relies on message passing and therefore is a distributed memory system. The way that we will utilize the OpenMP pragma statements to make the system parallel.

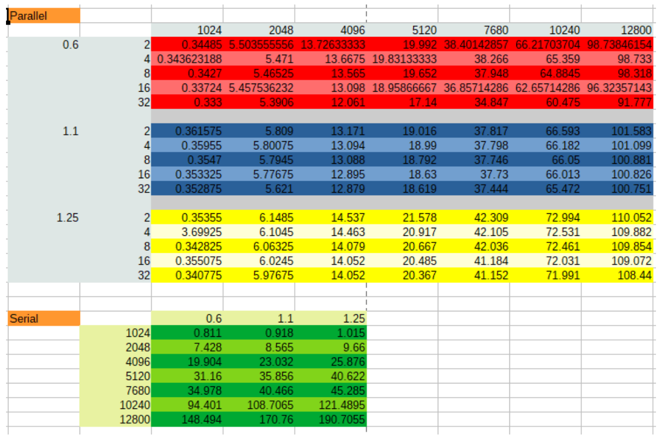
B. My OpenMP System

In my system I decided to focus on the for loops as they are the were the bulk of the operations are preformed and also because where the program spends the most amount of time. I looked at the convolution, suppression and hysteresis functions to parallelize first because they were my program was spending the most time and iterations to complete. I was able to determine this by using the valgrind tool which has a function allowing us to find the areas with the most iterations. To determine if this solution was better we can look at speedup and efficiency. We will also look at scalability based on the data.

IV. Data and Speedup

In order to determine how this program did and whether it was better the first thing that was looked at was the images that were produced. By using a diff comparison then we can determine that the pictures are the same. In this case the images yielded the same result which is good. The next thing to look at is how the performance of the program changed.

A. Data

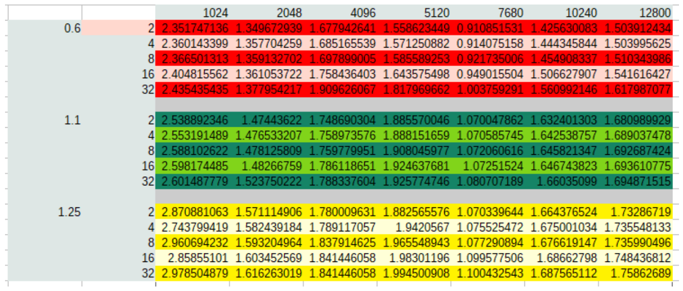


Parallel								
		1024	2048	4096	5120	7680	10240	12800
0.6	2	0.34485	5.503555556	13.72633333	19.992	38.40142857	66.21703704	98.73846154
	4	0.343623188	5.471	13.6675	19.83133333	38.266	65.359	98.733
	8	0.3427	5.46525	13.565	19.652	37.948	64.8845	98.318
	16	0.33724	5.457536232	13.098	18.95866667	36.85714286	62.65714286	96.32357143
	32	0.333	5.3906	12.061	17.14	34.847	60.475	91.777
1.1	2	0.361575	5.809	13.171	19.016	37.817	66.593	101.583
	4	0.35955	5.80075	13.094	18.99	37.798	66.182	101.099
	8	0.3547	5.7945	13.088	18.792	37.746	66.05	100.881
	16	0.353325	5.77675	12.895	18.63	37.73	66.013	100.826
	32	0.352875	5.621	12.879	18.619	37.444	65.472	100.751
1.25	2	0.35355	6.1485	14.537	21.578	42.309	72.994	110.052
	4	3.69925	6.1045	14.463	20.917	42.105	72.531	109.882
	8	0.342825	6.06325	14.079	20.667	42.036	72.461	109.854
	16	0.355075	6.0245	14.052	20.485	41.184	72.031	109.072
	32	0.340775	5.97675	14.052	20.367	41.152	71.991	108.44

Serial				
		0.6	1.1	1.25
	1024	0.811	0.918	1.015
	2048	7.428	8.565	9.66
	4096	19.904	23.032	25.876
	5120	31.16	35.856	40.622
	7680	34.978	40.466	45.285
	10240	94.401	108.7065	121.4895
	12800	148.494	170.76	190.7055

Figure 1: Times found by running the Program

Illustrate above are the times that were found by running the program on the cluster.



		1024	2048	4096	5120	7680	10240	12800
0.6	2	2.351747136	1.349672939	1.677942641	1.558623449	0.910851531	1.425630083	1.503912434
	4	2.360143399	1.357704259	1.685165539	1.571250882	0.914075158	1.444345844	1.503995625
	8	2.366501313	1.359132702	1.697899005	1.585589253	0.921735006	1.454908337	1.510343986
	16	2.404815562	1.361053722	1.758436403	1.643575498	0.949015504	1.506627907	1.541616427
	32	2.435435435	1.377954217	1.909626067	1.817969662	1.003759291	1.560992146	1.617987077
1.1	2	2.538892346	1.47443622	1.748690304	1.885570046	1.070047862	1.632401303	1.680989929
	4	2.553191489	1.476533207	1.758973576	1.888151659	1.070585745	1.642538757	1.689037478
	8	2.588102622	1.478125809	1.759779951	1.908045977	1.072060616	1.645821347	1.692687424
	16	2.598174485	1.48266759	1.786118651	1.924637681	1.07251524	1.646743823	1.693610775
	32	2.601487779	1.523750222	1.788337604	1.925774746	1.080707189	1.66035099	1.694871515
1.25	2	2.870881063	1.571114906	1.780009631	1.882565576	1.070339644	1.664376524	1.73286719
	4	2.743799419	1.582439184	1.789117057	1.9420567	1.075525472	1.675001034	1.735548133
	8	2.960694232	1.593204964	1.837914625	1.965548943	1.077290894	1.676619147	1.735990496
	16	2.85855101	1.603452569	1.841446058	1.98301196	1.099577506	1.68662798	1.748436812
	32	2.978504879	1.616263019	1.841446058	1.994500908	1.100432543	1.687565112	1.75862689

Figure 2: Speedup from the parallel to the Serial

From this speed up chart you can clearly see that there is a spedup that is noticed by the program. This is good because it means that the OpenMP library proved to be useful to us and did make the program better. One thing to note is that this program is scalable. We can deduce this by looking at the speedup as the more threads are added. Since the speedup does increase with the increase in threads we can say that the program is able to be scaled. As for the efficiency of the program we can say that there is efficiency because we do see speedup with increase in processors but the speedup could probably be improved. Given the information that we have it can be said that the program is weakly scalable. I think that with some work and rearrangement of the parallelization the program could be increased to be strongly scalable.

B. Difficulties with OpenMP/the Program/Final Thoughts

There was nothing that was too difficult using the OpenMP the only issue was that I did not quite understand that the -fopenmp had to be set as a flag and as an executable and once that was working the program was doing better. I did try this with smaller images but it was not as good as it is with the larger image size. If I had to compare this to Pthreads I think that I do not like it as much. Although OpenMP is very easy to use and pick up I felt that there is a sense of control missing that the pthreads have. Although I think that with some work with OpenMP I could come to like it more. The other problem that I had during this process was with the my repository having too many heads and it all getting jumbled up. I will not go too into as it does not pertain to the assignment. Another thing that I would like to mention is that this was the first time that I had to use the slurm bash script and the sbatch command. For me there was a bit of a learning curve here as when I used MPI I used he srun command so this was a little new and caught me off guard. Finally I do like using Canny Edge as a way to learn about OpenMp as it is a good candidate given the many for loops that it has. It is also nice to have a substantial output like an image when you are done.

Acknowledgment

Dr. Pallipuram for the lectures and some of the Psuedo code.

References

Canny Edge Detection Paper—By Robert Hughes from the ECPE 124 class. I used the description of my other algorithm for this paper but the results and the description of the OpenMP are not part of the other report.