Morphology Operations

Exploring TBB and Pthreads

Robert McInerney
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
ramciner@oakland.edu

***Abstract*—The purpose of this project is to explore how TBB and pthreads can impact execution speed.**

I. Introduction

The motivation behind this project is to determine the impact of different parallelization strategies. We will also explore how different aspects will change timings. This will include different sized structural element, amount of threads, picture size and hardware.

II. Methodology

A. *Strucutring Element*

A structuring element is the neighborhood of where we apply the morphological operation. The center of the structuring element will contain the input pixel of which the operation is being done. For the morphological operations we will be using a disk with radius 2 as well as radius 1. The mask that is applied we can call a kernel. This mask is hard coded into the cpp file. There is a switch available to determine if you want to use a disk size 1 or 2.

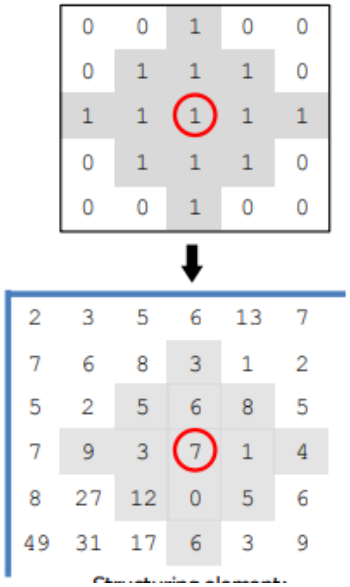

Figure 1: Structuring Element

B. *Erosion*

Erosion involves taking the minimal value of the image over the structuring element.

C. *Dilation*

Dilation involves taking the maximum value of the image in the structuring element.

D. *Opening*

Opening is when you do erosion followed by dilation. Opening is used for breaking narrow elements and eliminating small noise [2].

E. *Closing*

Closing is the opposite of opening in the sense that it should be dilation followed by erosion. Closing involves smoothing boundaries to join narrow breaks and small holes caused by noise. [2]

F. *Boundary Extraction*

Boundary extraction can be done as a inner boundary or an outer boundary extraction. To do an inner boundary you need to take the original image and subtract the eroded image. To do a outer boundary you need to take a dilated image – original image. Both of these operations have been explored in this project.

G. *Main Program Flow*

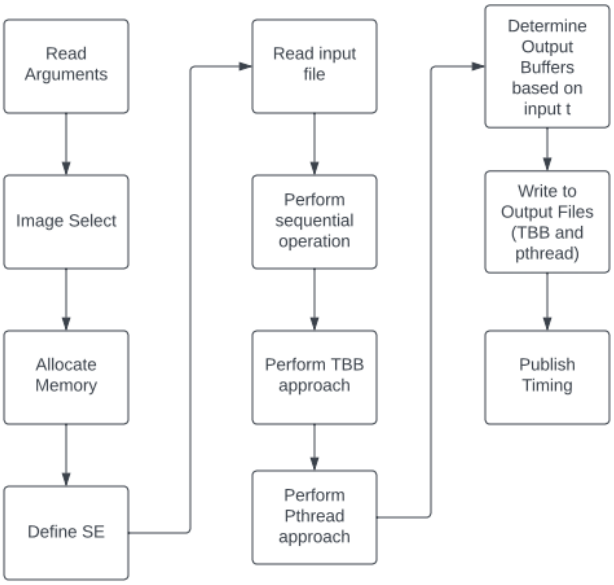

Figure 2: Main Program Flow Chart

The main program flow starts by reading elements that have been input by the user. This will be input initially when calling the program. There are four input elements. The first element is which morphology operation is needing to be performed. This can be a value from 1 – 6. Dilation is 1, erosion is 2, opening is 3, closing is 4, inner boundary extraction is 5, and finally outer boundary is 6.

The second input argument is an image selector. Since four images were used in the project this allows you to swap between the images. However, the input.bif needs to be updated before running again. This switch simply changes column and row values to account for different sized images. Now parameter 3 is the size of structuring element, this can be a 1 or a 2. Finally the last parameter is how many cores to launch for the pthread implementation.

The next steps in the program flow include allocating the memory for the tbb and pthreaded applications. We then define the values of SE based on the input parameters. Next we read in the input file into both the TBB and pthread buffers. The parallelization methods are then invoked. In order to write to the output image buffers we look at which operation was requested. This is due to the fact that some of these operations take two operations to complete. Thus it changes which buffer has the final output. The software uses O and O2 and buffers. In the case of opening for example we do erosion and store the results in O. Then we do dilation with the input to dilation being O. The output is then O2. So based on this information we will write O2 to the output file. The last part of the program flow is to publish the timing for sequential, tbb, and pthread implementation.

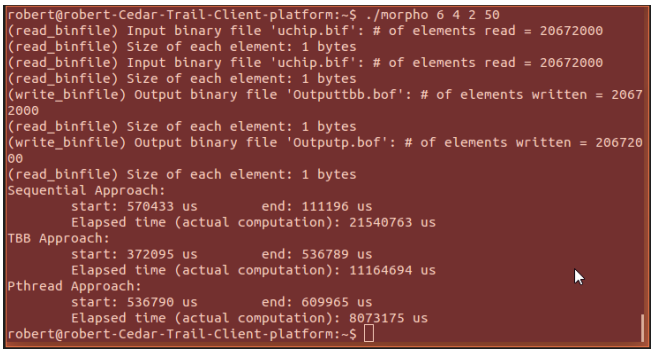

Figure 3: Application call

The two parallelization approaches that were used were TBB and pthreads. TBB was done by using nested parallel for loops. These were done along the lines of sX and sY. sX an sY are simply the rows and columns of the image. So, this allows each pixel to get the operation be completed in parallel. Now, for some operations it would take launching two separate parallel_for operations. For example in the case of opening you needed to perform erosion before dilation. So we would launch the parallel_for loops for erosion. Once all of these were complete then we would do the dilation of the output image from the dilation.

For the pthread approach threads were launched based off of columns from an image. Based on the number of threads an upper and lower bound would be calculated for each thread. This upper and lower bound would define how much of the image each thread would be responsible for. This essentially made the image into strips that each thread would take care of each row in those columns. Like TBB if there was a multi operation then we would need to launch the threads and wait for them all to complete or merge. After this, we would launch new threads for the second operation.

III. Experimental Setup

The project is being ran on a DE2i-150 board. This board is running linux operating system. Data was collected for multiple amount of threads as well as structuring element. The same test was similarly done on a desktop PC running Ryzen 7 3700.

IV. Results

The results will use a MATLAB script much like the ones used in assignments during the course. It will compare the image tested in MATLAB with the output file from the board. A sum of differences will be presented to the user. The matlab script will also support taking a JPEG image and converting it to a greyscale image that way multiple images can be tested. This will allow us to see the impact of different sized images regarding the 3 different approaches as well. Some elements of the matlab script may need edited to check different parameters such as image need to change to the image you are checking and SE needs updated accordingly.

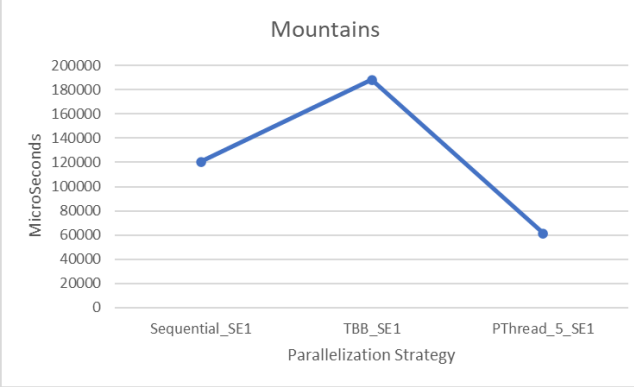

Figure 4: Mountains Dilation

We can see in figure 4 that TBB took slightly longer than the sequential approach for a relatively small image. This image was 600 x 400. For comparison sake of the three parallelization strategies we will use pthreads launching 5 threads. This is due to five threads being the best inflection point of time savings.

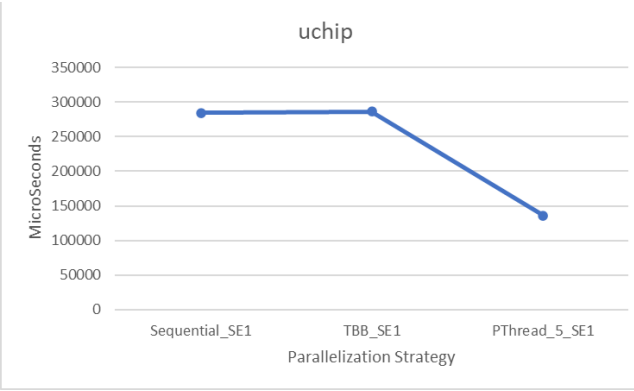

Figure 5: Uchip Dilation

In figure 5 we can see when we run a slightly larger image, 940 x 602 that there is some more benefit to TBB. The values are rather similar and seem to be only a slight gain. And in this case again pthreads is able to really give us a speed up.

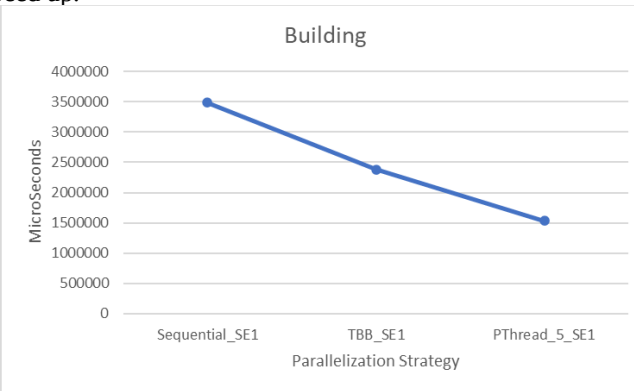

Figure 6: Building Dilation

For the building image the pixel count is 3472 by 2315. This is a considerable jump in pixel size compared to the previous two images. TBB now creates a significant time savings. And here again we see that pthreads saves the most time.

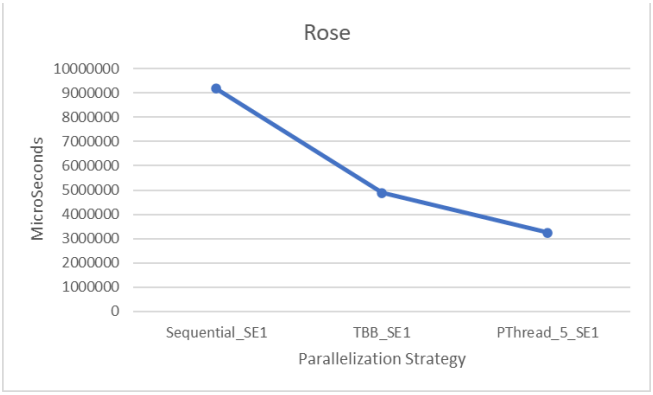

Figure 7: Rose Dilation

The final image is that of a rose that is coming in at 5168 by 4000 for the pixel count. Here we can see an even bigger jump in time savings for TBB. So it is safe to say as the image gets larger that TBB is able to see more and more benefits.

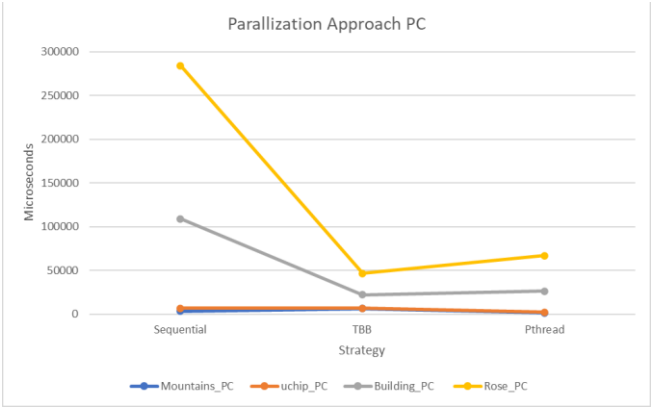

Figure 8: Parallelization on PC

The next point of comparison that was explored was that would these different strategies change if the hardware changed. It turns out that TBB had huge gains from extra resources available to the algorithm. This can be seen base on the differences of Figure 8 and Figure 9. Both of these operations were to do with dilation. Pthreads gained less by changing hardware than the TBB algorithm did.

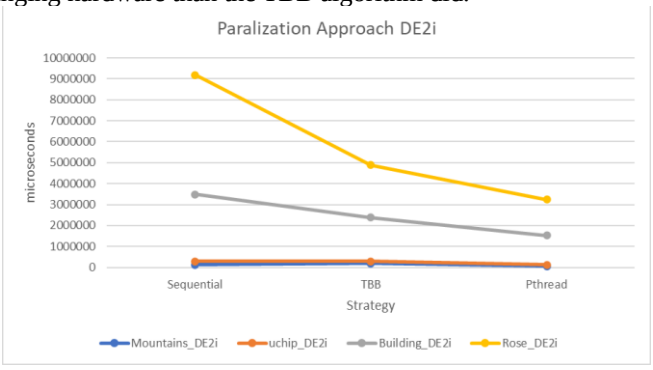

Figure 9: Parallelization on DE2i

The amount of pthreads executed is another avenue that I wanted to explore. It was found that going to 5 threads provided a very big jump in execution time. However, anything over 5 threads were minimal or negligible speed ups. This is due to the operation not being that intensive.

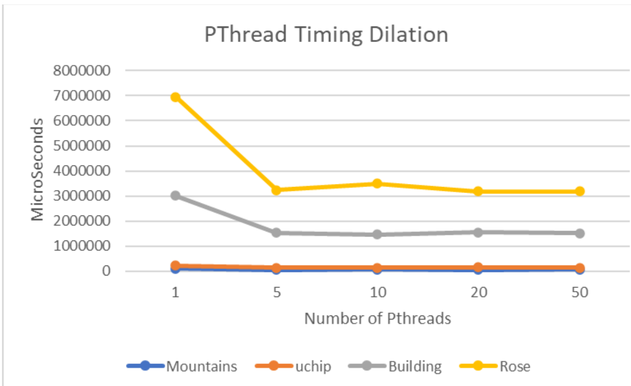

Figure 10: Amount of Pthreads

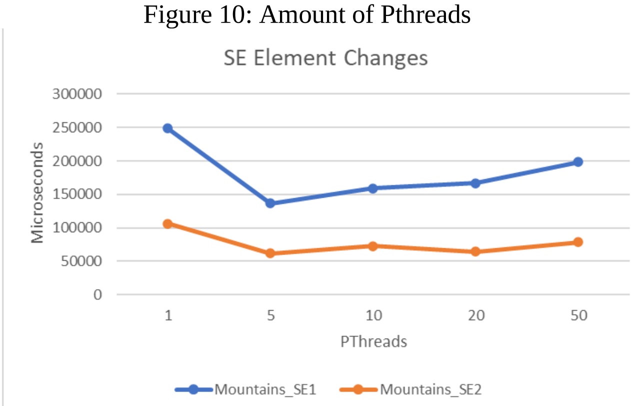

Figure 11: SE Element

The final test that was measured based on this experiment is how the changing of the structuring element could change the operations. Figure 11 is that of dilation and it can be seen that pthreads increasing did not change much even as the SE element increased. This is due to the pthreads were done at the pixel level, rather than launching pthreads throughout the kernel. Since the SE element itself is not very big it did not make sense to launch pthreads for every single pixel as the overhead would be huge. In the case of TBB we also did not do parallel_for during the phase of processing the kernel. Dilation and the other operations seen similar data trends. The rest of the data can be seen in the below tables.

	Seq_SE1	Seq_SE2	TBB_SE1	TBB_SE2	Pthread_1_SE1	Pthread_1_SE2	Pthread_5_SE1	Pthread_5_SE2	Pthread_10_SE1	Pthread_10_SE2	Pthread_20_SE1	Pthread_20_SE2	Pthread_50_SE1	Pthread_50
Mountains	120551	266941	188328	249967	106483	248659	61367	136608	72745	159148	64189	167055	78185	[illegible]
uchip	284897	639248	286583	458209	235195	530795	136239	360488	144802	328869	151804	280319	145954	[illegible]
Building	3490908	7995452	2383064	4206848	3014458	6403796	1534581	2984596	1459965	2819160	1547785	2812147	1516571	[illegible]
Rose	9187276	20676697	4892829	9933921	6950798	17463811	3241793	8014986	3503279	7824402	3178183	7985584	3183726	[illegible]

Table 1: Dilation Data DE2i

	Seq_SE1	Seq_SE2	TBB_SE1	TBB_SE2	Pthread_1_SE1	Pthread_1_SE2	Pthread_5_SE1	Pthread_5_SE2	Pthread_10_SE1	Pthread_10_SE2	Pthread_20_SE1	Pthread_20_SE2	Pthread_50_SE1	Pthread_50
Mountain	109497	270290	166332	180354	108796	204882	56545	138771	54633	118830	79284	148571	59238	[illegible]
uchip	275906	597850	246386	387816	228420	521549	138603	277480	128045	219109	117190	301112	205552	[illegible]
Building	3563168	7945285	2139637	4185801	2960328	6375754	1476377	2917283	1573125	2871926	1439835	2788243	1548017	[illegible]
Rose	9230117	20552584	4852511	9769482	7290506	17830155	3642061	8088283	3245409	7770532	3200814	7679463	3225424	[illegible]

Table 2: Erosion Data DE2i

	Seq_SE1	Seq_SE2	TBB_SE1	TBB_SE2	Pthread_1_SE1	Pthread_1_SE2	Pthread_5_SE1	Pthread_5_SE2	Pthread_10_SE1	Pthread_10_SE2	Pthread_20_SE1	Pthread_20_SE2	Pthread_50_SE1	Pthread_50
Mountains	265683	508764	259479	485170	242009	429252	162562	216339	112589	220920	113615	192433	162971	[illegible]
uchip	560971	1239350	512313	859476	428677	1007702	294637	508050	255860	561899	311793	501940	326972	[illegible]
Building	6965288	1.5E+07	4505080	7346544	5293005	12984597	2537143	5983332	2390379	5756866	2350235	5640977	2381606	[illegible]
Rose	1.8E+07	4E+07	9477284	1.9E+07	14548015	935497	6614419	16341261	6519064	15895372	6444468	15773273	6400396	[illegible]

Table 3: Opening Data DE2i

	Seq_SE1	Seq_SE2	TBB_SE1	TBB_SE2	Pthread_1_SE1	Pthread_1_SE2	Pthread_5_SE1	Pthread_5_SE2	Pthread_10_SE1	Pthread_10_SE2	Pthread_20_SE1	Pthread_20_SE2	Pthread_50_SE1	Pthread_50_SE2
Mountains	236346	550627	213472	431755	237199	454666	109375	240942	152729	225473	116752	216491	179678	213800
uchip	546862	1182265	533712	840305	435127	998788	246247	615287	74659	548462	254939	504174	352930	660183
Building	6946759	15729883	4625990	7544601	5440374	12966177	2560803	6017898	2440844	5749113	2410203	5662718	2434772	5641898
Rose	17942233	40670363	9800296	19336016	391385	35381258	6742345	16497556	6607383	15599754	6401961	15450576	6424020	15429605

Table 4: Closing Data DE2i

	Seq_SE1	Seq_SE2	TBB_SE1	TBB_SE2	Pthread_1_SE1	Pthread_1_SE2	Pthread_5_SE1	Pthread_5_SE2	Pthread_10_SE1	Pthread_10_SE2	Pthread_20_SE1	Pthread_20_SE2	Pthread_50_SE1	Pthread_50_SE2
Mountains	131575	291376	247733	291385	126771	259507	69210	152936	112527	228269	201918	139236	77837	237042
uchip	311453	643226	453709	491218	281295	537339	199310	378083	149308	351472	172511	319951	303606	392514
Building	3831475	8205205	4348381	4824815	3233851	6672525	1728854	2963653	1724438	2960514	1714295	2905538	1728905	2998509
Rose	10258274	21478900	6137103	11211958	8032395	18473486	3733489	8802762	3629113	8299326	3570036	8043697	3590185	8020998

Table 5: Boundary Inner DE2i

	Seq_SE1	Seq_SE2	TBB_SE1	TBB_SE2	Pthread_1_SE1	Pthread_1_SE2	Pthread_5_SE1	Pthread_5_SE2	Pthread_10_SE1	Pthread_10_SE2	Pthread_20_SE1	Pthread_20_SE2	Pthread_50_SE1	Pthread_50_SE2
Mountains	139626	314015	242933	294434	132002	254962	88063	161508	80152	179123	188611	139166	178860	150707
uchip	305018	659849	419194	525046	258725	545634	197351	343818	200391	357812	246757	370991	264634	391350
Building	3797182	8336426	3055152	4793180	3238831	6818662	1744684	3171949	1693964	3016680	1695004	2975828	1680505	2971570
Rose	10021615	21564223	6065090	10976780	7700796	18193783	3601145	8435874	3451119	8150420	3767192	8106542	3508802	8073175

Table 6: Boundary Outer DE2i

	Seq_SE1	Seq_SE2	TBB_SE1	TBB_SE2	Pthread_1_SE1	Pthread_1_SE2	Pthread_5_SE1	Pthread_5_SE2	Pthread_10_SE1	Pthread_10_SE2	Pthread_20_SE1	Pthread_20_SE2	Pthread_50_SE1	Pthread_50_SE2
Mountains	3151	7715	6130	6321	3607	7276	1176	1831	1648	1586	2145	2256	3866	3840
uchip	6618	17010	6726	7932	7761	15249	2102	3952	3089	3241	3865	3966	4890	5324
Building	108853	263097	21979	37041	96336	203068	26087	50474	33374	37852	34560	40876	37529	41096
Rose	284234	642836	46394	84444	241531	520014	66662	126781	82916	95445	84147	109359	88949	120041

Table 7: Dilation Data PC

	Seq_SE1	Seq_SE2	TBB_SE1	TBB_SE2	Pthread_1_SE1	Pthread_1_SE2	Pthread_5_SE1	Pthread_5_SE2	Pthread_10_SE1	Pthread_10_SE2	Pthread_20_SE1	Pthread_20_SE2	Pthread_50_SE1	Pthread_50_SE2
Mountains	4713	7719	6139	6498	3256	7084	1242	1885	1633	1611	2345	2639	3732	3610
uchip	11085	17949	6847	7610	7248	16442	2186	3745	2973	3092	3630	4001	4922	5457
Building	159015	235412	23733	34040	103585	215039	26473	50702	32115	35364	33123	38061	38002	39957
Rose	396654	634146	48444	81626	264117	570568	81501	142134	85662	102310	84187	112988	88534	108017

Table 8: Erosion Data PC

	Seq_SE1	Seq_SE2	TBB_SE1	TBB_SE2	Pthread_1_SE1	Pthread_1_SE2	Pthread_5_SE1	Pthread_5_SE2	Pthread_10_SE1	Pthread_10_SE2	Pthread_20_SE1	Pthread_20_SE2	Pthread_50_SE1	Pthread_50_SE2
Mountains	7385	14487	6500	7457	5946	14327	2139	3724	3070	3300	4226	4997	7344	7944
uchip	18142	33885	8638	10180	15124	33122	4343	8173	6008	67287	7781	8168	9661	11184
Building	248266	457422	37443	67143	197649	429951	51642	100761	64019	71982	64785	76275	72874	87898
Rose	641838	1192938	83734	157048	508381	1127883	144534	257063	179357	179974	169550	218593	197372	213232

Table 9: Opening Data PC

	Seq_SE1	Seq_SE2	TBB_SE1	TBB_SE2	Pthread_1_SE1	Pthread_1_SE2	Pthread_5_SE1	Pthread_5_SE2	Pthread_10_SE1	Pthread_10_SE2	Pthread_20_SE1	Pthread_20_SE2	Pthread_50_SE1	Pthread_50_SE2
Mountains	8002	14578	6785	7663	5888	13791	2247	3518	3013	3221	4187	4992	6973	7718
uchip	18100	34905	8578	10248	14149	30525	4490	7376	5895	6936	7415	8109	9643	10924
Building	261549	516210	42726	63338	202071	414612	52053	94553	64039	68306	67555	74105	72374	81289
Rose	679632	1257710	99054	162591	507223	1070790	147137	264031	181767	184688	204493	200908	179676	208678

Table 10: Closing Inner PC

	Seq_SE1	Seq_SE2	TBB_SE1	TBB_SE2	Pthread_1_SE1	Pthread_1_SE2	Pthread_5_SE1	Pthread_5_SE2	Pthread_10_SE1	Pthread_10_SE2	Pthread_20_SE1	Pthread_20_SE2	Pthread_50_SE1	Pthread_50_SE2
Mountains	4710	8179	6405	7019	3808	7936	2162	2811	2959	3148	3920	4332	6817	6347
uchip	10525	16945	7808	8792	8848	17666	4503	6158	5875	5819	6868	7446	9288	9073
Building	150570	250657	26486	41868	125213	258095	55909	79422	63322	69130	65653	70490	71879	77891
Rose	393802	620321	62903	100139	318577	647928	152697	233292	194010	181407	189068	178735	199816	190238

Table 11: Boundary Inner PC

	Seq_SE1	Seq_SE2	TBB_SE1	TBB_SE2	Pthread_1_SE1	Pthread_1_SE2	Pthread_5_SE1	Pthread_5_SE2	Pthread_10_SE1	Pthread_10_SE2	Pthread_20_SE1	Pthread_20_SE2	Pthread_50_SE1	Pthread_50_SE2
Mountains	3390	7771	7220	6781	4236	7262	2392	2847	2916	3374	3893	4195	6085	6695
uchip	8362	17512	13204	8718	9055	17822	4578	6073	5663	5886	6959	7422	9776	9207
Building	118908	236405	27683	40340	123752	226626	54408	81819	64479	69122	64611	69805	72281	73929
Rose	302092	625097	59889	117770	321099	599216	145541	208096	167124	181778	178008	191420	171192	202914

Table 12: Boundary Outer PC

CONCLUSIONS

In conclusion I have seen that parallelization can be a massive asset to performing image processing. I can also say that the more resources that your PC has available the further TBB is able to optimize your system. The pthread implementation was able to give me the best results. And the gains from pthreads seemed to level off at 5 threads allocated. Both of the parallelization techniques saw bigger gains as the pictures increased in size. The structural element being smaller or larger was not really sped up via TBB or pthreads. However, this is expected based on the implementation of the code. The next step for this project would be to investigate how to parallelize the solving of the kernel for each pixel.

REFERENCES

[1] Lecture 7 - web.cs.wpi.edu. (n.d.). https://web.cs.wpi.edu/~emmanuel/courses/cs545/S14/slides/lecture07.pdf

[2] Morphological image analysis - purdue university. (n.d.-b). http://www.cyto.purdue.edu/cdroms/micro2/content/education/wirth07.pdf

[3] GeeksforGeeks. (2022, March 14). *Boundary extraction of image using MATLAB*. GeeksforGeeks. https://www.geeksforgeeks.org/boundary-extraction-of-image-using-matlab/