ASSIGNMENT 1 REPORT

VNTNIC019

METHODS

PARALLELISATION ALGORITHMS

The parallelization method used made use of a combination of a divide and conquer method and the fork/join framework. At the beginning of the program three arrays each containing the red, green and blue value of every pixel is created and then depending on these arrays sizes the work that needs to be done is split and assigned to different threads. A threshold value was used to gauge the size of the array and determine how many threads it should be run over. In each of the treads the program calculates new pixel values for a certain part of the array depending on the start and stop values that were passes when creating the thread. This allows for multiple parts of the arrays to be worked on at once.

VALIDATING THE PROCESSED IMAGES

I made use of a simple java script that looped through each pixel of the parallel and serial versions of an image and compared the values to ensure they were the same.

MEAN






MEDIAN







TIMING

To ensure correct timing the System.currentTimeMillis method was used to get the time just before the image processing began and again just afterwards. This allowed for the time spent processing to be calculated. To ensure that the code was well tested each test was run five times. This ensured that any anomalies would be picked up and a more accurate time would be achieved. Out of the five different tests the smallest time is chosen as we are focused on finding out the quickest times possible.

OPTIMAL SERIAL THRESHOLD

The serial threshold is an important part of the parallel programs as it determines how many threads are created. To properly calculate what this value should be multiple tests were run. A total of 100 tests were run(25 tests were run for each of the four programs). A fixed frame size of 9 pixels by 9 pixels was used across 5 different images varying in size (300x300 (person), 732×549(tea), 1000x1000(cat), 3840×2160 (castle), 5295×3530 (bridge)) with 5 different threshold values (50,100,250,500,1000).These values were used in both the Mean and Median parallel filters and after all the tests it was possible to see which threshold, on average, was best. For the Mean parallel filter the optimal serial threshold was 250 and for the Median the optimal serial threshold was 500.

MACHINE ARCHITECTURES

The programs were only tested on one architecture. This was an 8-core Apple M1 CPU. This chip is made up of four high-performance cores and four high-efficiency cores.

PROBLEMS/DIFFICULTIES

Initially my program was running much slower than I hypothesized. To help speed up my serial program I changed the way my algorithm reads and references each pixel of the image. Initially I was looping through all the pixels in the image and then reading in the pixels in the frame around it. This led to the rereading of pixels as many frames overlapped and this caused a significant slowdown in the program. To help fix this the program rather loops through each pixel in the image and stores each red, green and blue value in an array. After this the filter algorithm will run and all it will need to do it pull the RBG data from the arrays instead of re-reading the RGB values of all the pixels in the surrounding frame. Getting a variable from an array was much quicker than calculating the RGB value from a pixel.

RESULTS

OPTIMAL SEQUENTIAL CUTOFF FOR THE PARALLEL ALGORITHM

		Mean Parallel					Median Parallel				
		1	2	3	4	5	1	2	3	4	5
IMG SIZE	Cut off										
300x300 (person)	50	190	150	181	203	157	703	579	507	767	665
	100	147	209	115	121	118	618	543	548	**469**	589
	250	101	102	95	144	96	604	557	586	564	564
	500	96	99	80	101	88	785	812	813	826	819
	1000	**86**	107	105	111	122	715	806	819	812	840
732×549(tea)	50	240	347	214	248	310	1050	1037	1153	1042	1232
	100	208	203	193	179	194	999	970	**878**	1071	1041
	250	181	190	**156**	190	198	1035	1144	1071	1122	1114
	500	239	254	168	217	242	1474	1487	1437	1511	1491
	1000	254	306	313	236	263	2601	2752	2698	2713	2831
1000x1000(cat)	50	506	365	373	514	359	2153	2015	2123	1985	2010
	100	339	359	356	310	403	1830	1944	1714	2069	1981
	250	294	464	332	483	404	1549	1557	1746	1622	**1504**
	500	299	**282**	293	427	284	2174	2123	2076	2078	2112
	1000	316	378	323	385	340	3183	3332	2825	3189	3079
3840×2160 (castle)	50	1841	1914	1897	1767	1875	19937	17847	18638	19816	17186
	100	1470	1384	1553	1605	1457	14133	14401	13598	14956	13806
	250	1288	1296	1274	1423	1335	13358	**10597**	10670	13199	11331
	500	1415	1300	**1131**	1347	1153	10813	10697	10934	10824	10821
	1000	1503	1729	1342	1527	1541	17882	18206	17988	19390	18235
5295×3530 (bridge)	50	3120	3687	3278	3289	3604	21844	21486	22240	20830	20691
	100	3040	2730	2857	2967	2708	19120	16831	18256	18572	19882
	250	2701	2704	3035	3010	2657	15287	16756	**15173**	16073	18015
	500	2696	2630	2500	**2234**	2467	19722	20457	19446	19094	21654
	1000	4368	2499	3191	3084	3410	16911	16542	16073	16002	18787

All the data was recorded in milliseconds.

Mean Parallel Optimal Cut off: 500

Median Parallel Optimal Cut off: 250

OPTIMAL DATA SET SIZES AND FILTER SIZES FOR THE PARALLEL PROGRAMS.

Best Frame Size:

FRAMES TEST	IMG= cat(1000x1000)									
	Mean Series					Mean Parallel				
FRAME SIZE	1	2	3	4	5	1	2	3	4	5
3x3	181	197	198	153	225	186	202	187	198	181
5x5	256	243	254	255	282	235	233	265	228	221
7x7	328	331	327	339	346	228	240	338	339	242
9x9	585	491	548	593	483	349	331	299	321	308
11x11	741	674	687	767	645	392	442	387	367	413
	Median Series					Median Parallel				
	1	2	3	4	5	1	2	3	4	5
3x3	481	466	476	473	470	348	383	343	273	351
5x5	1361	1392	1581	1560	1554	527	580	603	582	530
7x7	3311	3013	3409	3135	2972	990	1200	1084	1157	1180
9x9	5738	5581	5779	5716	5396	1547	1525	1683	1545	1676
11x11	8471	8559	8377	8539	8455	2963	2386	2174	2414	2318

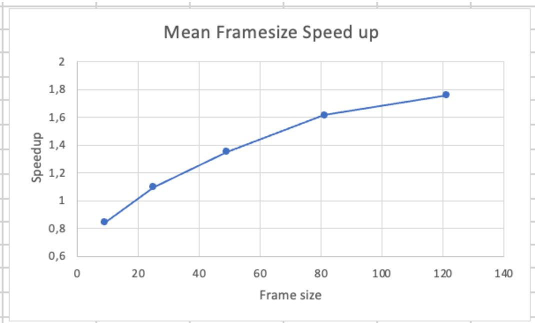
Best Image Size:

IMAGES TEST	Frame=5x5									
	Mean Series					Mean Parallel				
IMG SIZE	1	2	3	4	5	1	2	3	4	5
300x300 (person)	48	45	44	45	51	102	87	80	79	99
732×549(tea)	143	152	144	159	128	138	191	138	140	130
1000x1000(cat)	251	214	316	253	249	243	268	272	204	241
3840×2160 (castle)	2158	1905	2056	1820	1922	635	843	667	804	831
5295×3530 (bridge)	4713	4858	5243	4731	5148	1269	1234	1703	1540	1532
	Median Series					Median Parallel				
	1	2	3	4	5	1	2	3	4	5
300x300 (person)	206	194	258	212	262	189	248	187	185	173
732×549(tea)	719	696	691	708	694	385	391	369	387	366
1000x1000(cat)	1556	1552	1558	1525	1646	604	924	574	618	556
3840×2160 (castle)	14427	14295	14660	14844	14521	3476	3635	3658	3338	3155
5295×3530 (bridge)	22133	23643	23184	23481	23983	5455	4912	5660	5168	4808

SPEEDUPS OF PARALLEL ALGORITHMS

Frame Size Speedup:

Max speed up: 3,85:



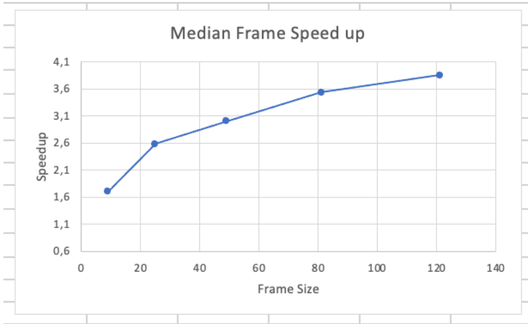
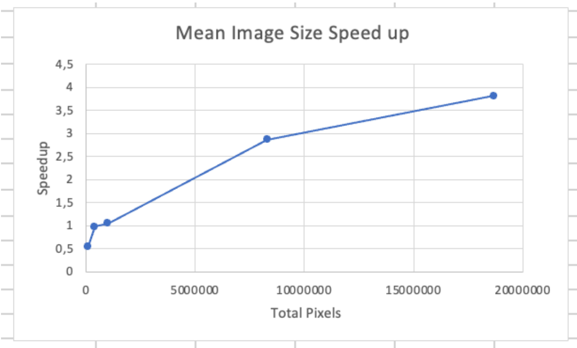
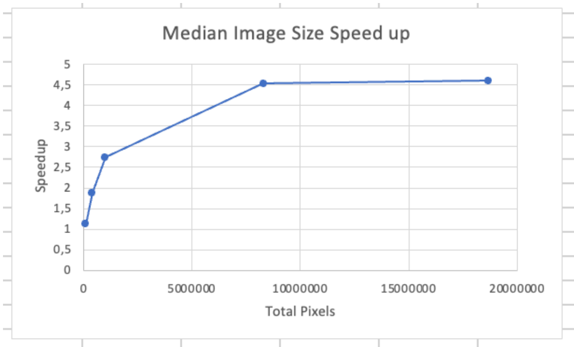


Image Size Speedup:

Max Speed up: 4,60





RELIABILITY

To ensure reliability each test was run 5 times. This helped to iron out anomalies as it would be easy to pick them up. While the tests were running all other programs running on the machine were terminated to allow for an even load while testing. These precautions showed in the results as everything was as predicted.

CONCLUSIONS

The parallel algorithms proved to be much quicker when implemented correctly. From the results we can see that for the larger frame sizes and larger image sizes the speed ups can be up to 4 times their serial counter parts. This proves than in cases where a lot of processing is needed parallel programs are definitely worth implementing. This case is further highlighted when comparing the maximum speed ups of the median and mean programs. The median program required more processing as arrays were used and sorting of these arrays was needed, this more intensive processing allowed for a greater speedup when compared to the speed up of the mean program. From all of this we can conclude that in the right situations parallel algorithms can be of huge benefit to the efficiency of a program.