Andrew Frost
CS415
PA2 Report
03/12/2017

Overview:
This report details the results of generating a Mandelbrot using a sequential algorithm, a static task assignment parallel algorithm, and a dynamic task assignment parallel algorithm. All timings in this report are the result of averaging six different test runs of each image size and number of processes. The following image sizes were used: 512 pixels by 512 pixels, 2048 pixels by 2048 pixels, 8192 pixels by 8192 pixels, 16384 pixels by 16384 pixels, and 32768 pixels by 32768 pixels. The following number of cores were used on the parallel algorithms: 3 cores, 8 cores, 9 cores, 13 cores, 16 cores, 17 cores, and 32 cores. The number of cores tested were specifically chosen to test thresholds based on the number of processors on each machine in the cluster. The following factors are analyzed: run time, speed up, and efficiency. The complete data for this report can be found in Data.xlsx. Figure 1 displays a Mandelbrot produced by the program.

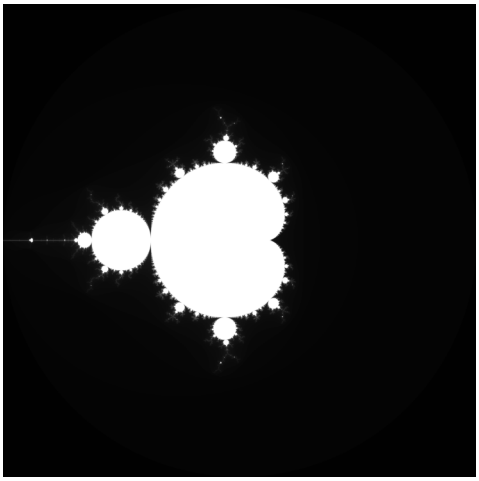

Figure 1: A Mandelbrot produced by the program.

Sequential Algorithm:
The sequential algorithm's run time will now be analyzed. The sequential algorithm was run on images of the sizes described in the overview section. Just as with the parallel algorithms, each time was the product of an average of six trial runs for each image. Figure 2 shows a graph of the sequential run times.

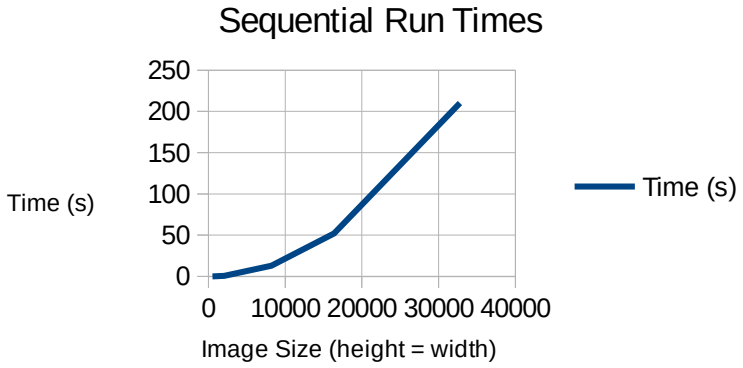

Figure 2: A line plot of the sequential algorithm run times. The computation time increases with the image size. The image sizes measured were 512 pixels by 512 pixels, 2048 pixels by 2048 pixels, 8192 pixels by 8192 pixels, 16384 pixels by 16384 pixels, and 32768 pixels by 32768 pixels.

As expected, the time to compute a Mandelbrot increases with the size of the image when using the sequential algorithm. Table 1 gives the averaged run times for the sequential algorithm.

Sequential Run Times

Cores	Image Size	Time (s)
1	512	0.0539536667
1	2048	0.8171153333
1	8192	13.119733333
1	16384	52.152583333
1	32768	209.9145

Table 1: This table contains the averaged run timings for each image size for the sequential algorithm.

The sequential algorithm will be used as a baseline for the effective improvement given by the parallel algorithms. The static task assignment algorithm will be discussed next.

Static Task Assignment Parallel Algorithm:

The static task assignment parallel algorithm divided up the computation of rows based on the number of cores allocated to the program. Each worker node was given a more or less even amount of work depending on whether or not the number of workers divided up the data evenly. Each slave node processed the number of rows divided by the number of slaves starting with the row number based on their task id minus one. The master node sat idle while computations were completed. The master node only did the management work of collecting data after it was processed. This approach minimized the amount of communication between the master and the slave nodes. Figure 3 shows the run time speeds of the static task assignment parallel algorithm and the sequential algorithm.

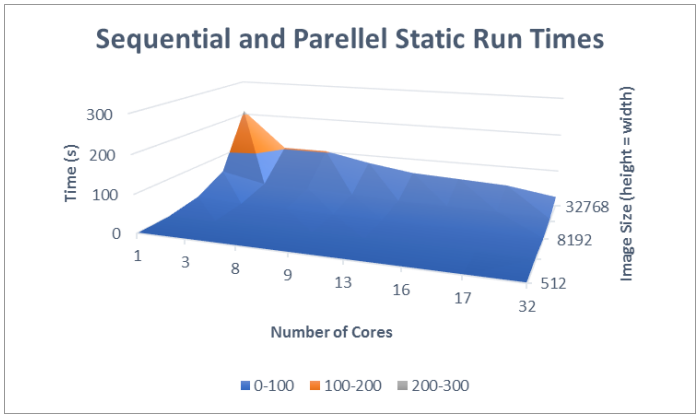

Figure 3: This graph shows the sequential run time for 1 node and the parallel static task assignment algorithm over a series of image sizes and processor allocations. The image sizes processed were 512 pixels by 512 pixels, 2048 pixels by 2048 pixels, 8192 pixels by 8192 pixels, 16384 pixels by 16384 pixels, and 32768 pixels by 32768 pixels. The number of cores used were 3 cores, 8 cores, 9 cores, 13 cores, 16 cores, 17 cores, and 32 cores.

As the number of processors are added, the static task assignment algorithm gradually and consistently requires less time to calculate the Mandelbrot across almost all image sizes. Table 2 shows the run times from the sequential and static task assignment algorithms.

Static Task Assignment Run Times (s)

	Image Size								
Cores		1	3	8	9	13	16	17	32
	512	0.0539536667	0.0321333333	0.031014	0.0263233333	0.0307136667	0.0445851667	0.0289351667	0.0564933333
	2048	0.8171153333	0.4133005	0.4058313333	0.3143496667	0.2480163333	0.2295121667	0.2102318571	0.5680805
	8192	13.119733333	6.4522533333	6.4877983333	4.86679	3.6526883333	3.3114166667	2.9069433333	1.8220366667
	16384	52.152583333	25.812533333	25.922183333	19.4492	14.664066667	13.110933333	11.6188	6.5179066667
	32768	209.9145	103.22116667	103.074	77.788283333	58.72215	52.679133333	46.35795	25.970833333

Table 2: The static task assignment run times are listed in seconds in this table. The sequential run times are listed under the 1 core section. As the image size increases the time needed to compute the Mandelbrot increases for both algorithms, but there is a noticeable decrease in the time required for the Static Task assignment algorithm in most cases.

In most cases, the static task assignment algorithm out performs the sequential algorithm. There are, however, some instances where the communication time exceeds the benefit of additional cores. For instance, when processing the 512 pixels by 512 pixels image the length of the run time begins increasing after 9 cores and the use of 32 cores is actually slower than the sequential algorithm. This is a case where the time of communication is greater than the time of computation. The speed up factor is further explored in Fig. 4.

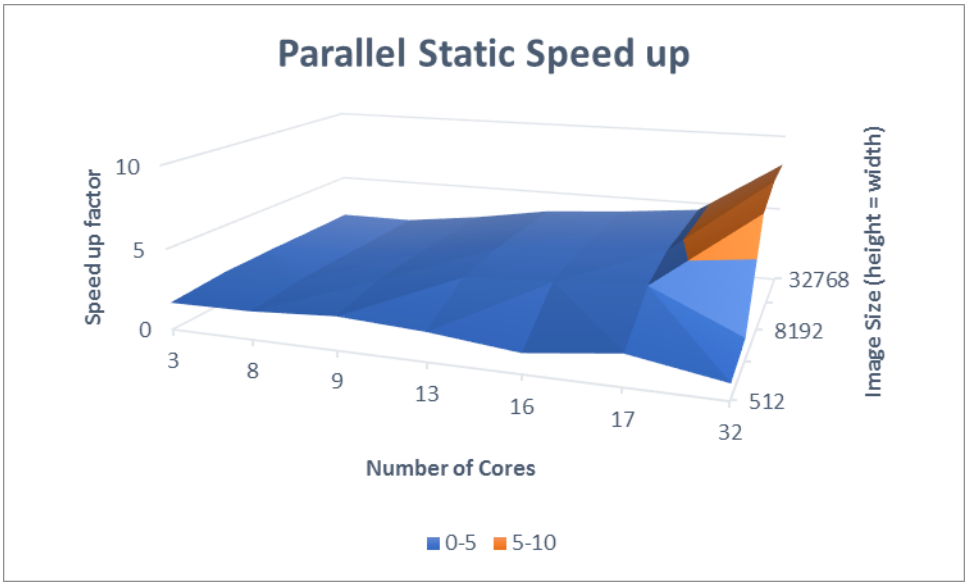

Figure 4: The static task assignment speed up relative to the sequential algorithm.

The speed up factor was calculated by dividing the sequential time by the parallel time. On the larger images, the speed up factors reaches a factor of over 8. On the smaller image, the speed up factor approached 1 as the number of cores increased. Table 3 contains the speed up factor for all of the averaged timings.

Speed up for Static Task Assignment

	Image Size							
Cores		3	8	9	13	16	17	32
	512	1.6790560166	1.7396552095	2.0496517665	1.7566664134	1.2101259387	1.8646399134	0.955044843
	2048	1.9770489833	2.0134357952	2.5993834891	3.2946029092	3.5602266547	3.8867341251	1.4383794785
	8192	2.0333568221	2.0222165763	2.695767299	3.5918020198	3.9619699524	4.5132401388	7.200586889
	16384	2.0204364546	2.0118900736	2.6814770445	3.5564884229	3.977793343	4.4886376677	8.0014314412
	32768	2.0336381265	2.0365417079	2.6985362191	3.5747073293	3.9847751229	4.528123008	8.0827017488

Table 3: The speed up factor of the average run times for the static task assignment algorithm for each core and image size.

The greatest speed up occurred on an image of 32768 pixels by 32768 pixels being processed by 32 cores. The worst speed up factor occurred when processing a 512 pixels by 512 pixels image with 32 cores. This demonstrates the case were the communication time outweighs the saved computation time. The efficiency for each number of cores and for each image size was calculated. The efficiency was calculated by dividing the time to compute the image with one core by the product of the time to compute the image with multiple cores and the number of the cores. The efficiency is displayed in Fig. 5.

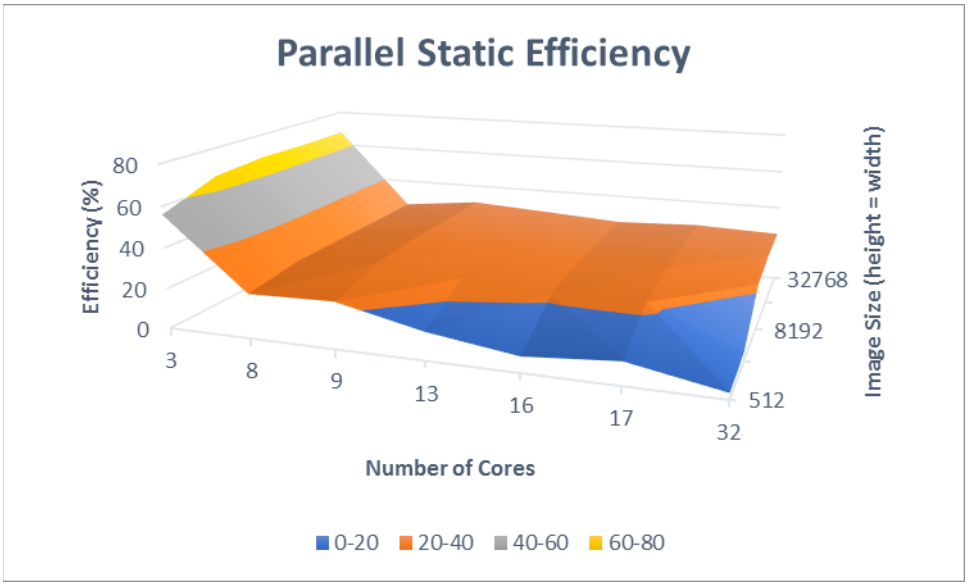

Figure 5: The efficiency of the static task assignment parallel algorithm. The efficiency is presented as a percentage. Interestingly, the greatest efficiency happened with 3 cores (2 workers) being used for all image sizes.

The static task assignment parallel algorithm achieved its greatest efficiency at 3 cores. This is when it most efficiently used the system resources. It is important to note that the efficiency of this algorithm never exceeded 68%. The full list of efficiencies for the test data is listed in Table 4.

Static Task Assignment Efficiency

	Cores					
Image Size		512	2048	8192	16384	32768
	3	55.968533887	65.901632778	67.778560737	67.347881821	67.787937552
	8	21.745690118	25.16794744	25.277707204	25.148625919	25.456771349
	9	22.773908517	28.882038768	29.952969989	29.794189383	29.983735768
	13	13.512818565	25.343099302	27.629246306	27.357603253	27.497748687
	16	7.5632871172	22.251416592	24.762312203	24.861208394	24.904844518
	17	10.968470079	22.863141912	26.548471405	26.403750986	26.636017694
	32	2.9845151345	4.4949358703	22.501834028	25.004473254	25.258442965

Table 4: The efficiency of static task assignment.

Overall, the static tasks assignment parallel algorithm computed the Mandelbrot faster than the sequential algorithm. In some cases the communication time outweighed the computational time.

Dynamic Task Assignment Parallel Algorithm:

The dynamic task assignment parallel algorithm utilized a master node and a series of workers. The master's entire job was to delegate and collect computations on rows. Unlike the static task assignment parallel algorithm, the dynamic task assignment parallel algorithm's master sends a row to each worker as they become idle. This allows for load balancing at the cost of more communications. The timing results from the dynamic task assignment algorithm is displayed in Fig. 6.

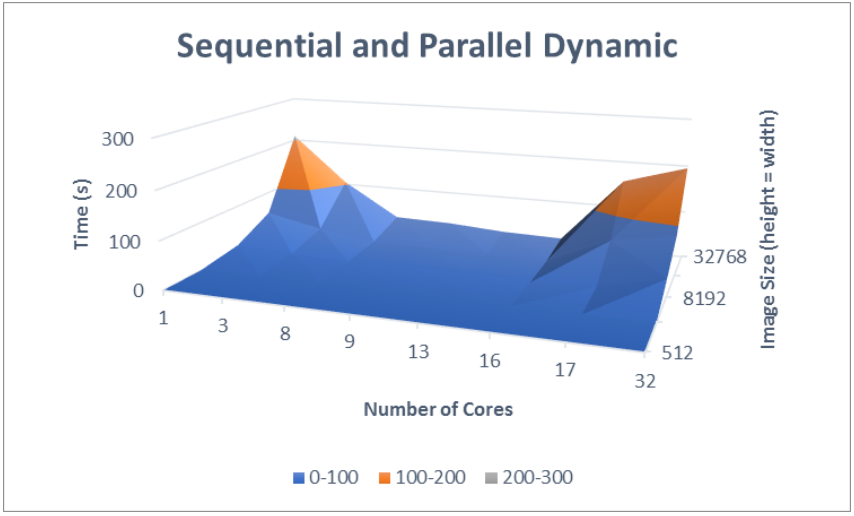

Figure 6: The averaged timing results of the parallel dynamic task assignment algorithm. The values for 1 core processing of images are those from the sequential algorithm. The image sizes processed were 512 pixels by 512 pixels, 2048 pixels by 2048 pixels, 8192 pixels by 8192 pixels, 16384 pixels by 16384 pixels, and 32768 pixels by 32768 pixels. The number of cores used were 3 cores, 8 cores, 9 cores, 13 cores, 16 cores, 17 cores, and 32 cores.

The time savings by adding more cores on the dynamic task assignment algorithm bottomed out at 16 cores. This is likely because of the increased communication time caused by message passing between more than two machines on the cluster outweighed the saved computation time. Even when only using 16 cores, the dynamic task assignment algorithm performed better than the static task assignment and sequential algorithms. This trend continued even when the static task assignment algorithm utilized more cores than the dynamic task assignment algorithm. It is the prediction of the author that with images larger than 32768 pixels by 32768 pixels that the trend of a decrease in time would continue to 32 cores and beyond. The averaged timing results for the dynamic task assignment algorithm are presented in Table 5.

Dynamic Task Assignment Timing Results (s)

	Image Size								
Cores		1	3	8	9	13	16	17	32
	512	0.0539536667	0.0304066667	0.010134	0.0149283333	0.017541	0.046336	0.032944	0.064355
	2048	0.8171153333	0.4142673333	0.1188478333	0.1261356667	0.0910995	0.0846213333	0.0891131667	0.0917191667
	8192	13.119733333	6.48625	1.863885	1.7322483333	1.1891866667	0.9705086667	7.4566483333	38.175483333
	16384	52.152583333	25.89985	7.44553	6.7603983333	4.6448133333	3.7665833333	61.697466667	84.94275
	32768	209.9145	104.84166667	29.960566667	26.863	18.237533333	15.398883333	157.30916667	195.58466667

Table 5: The dynamic task assignment parallel algorithm timing results in seconds. The 1 core values are from the sequential algorithm.

For all images, except 512 pixels by 512 pixels, the best timing occurred at 16 cores. This is when two machines in the cluster were being fully utilized. After 16 cores, the communication time outweighs the time saved by dividing the computational load over more processors. In every case, except when processing the 32768 pixel by 32768 pixel Mandelbrot, the use of 32 cores is slower than the sequential algorithm. With the exception of the 512 pixel by 512 pixel image, it is important to note that the processing times of the dynamic task assignment algorithm at 16 cores were less than the processing times of the static task assignment algorithm at 32 cores. The speed up of the dynamic task assignment algorithm, calculated by dividing the sequential time by the parallel time, is graphed in Fig 7.

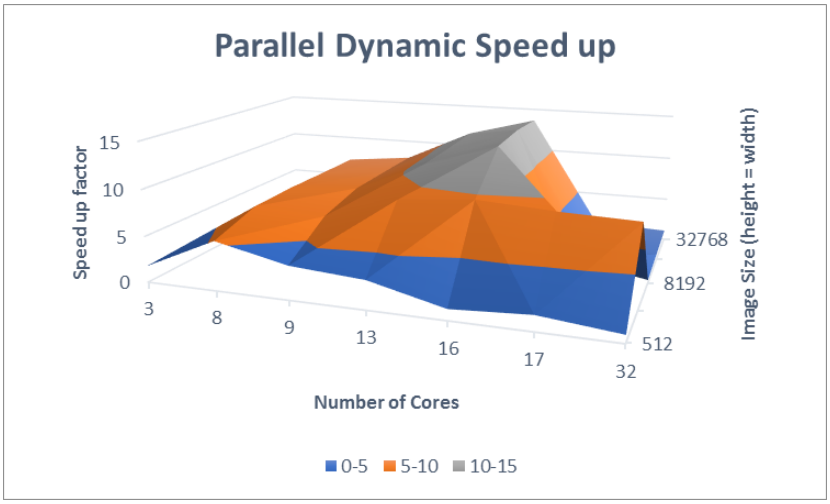

Figure 7: The speed up of the dynamic task assignment algorithm compared to the sequential algorithm. The dynamic task algorithm was able to achieve a greater maximum speed up than the static task assignment algorithm.

The speed up factor of the dynamic task assignment algorithm was maximized in most cases at 16 cores. The maximum speed up factor and the density of high speed up factors of the dynamic task assignment algorithm are greater than the static task assignment algorithm as well. Table 6 holds the speed up factor for each averaged timing of the dynamic task assignment algorithm.

Dynamic Task Assignment Speed up

	Image Size							
Cores		3	8	9	13	16	17	32
	512	1.7744025433	5.3240247352	3.6141788545	3.0758603652	1.16440061	1.6377387891	0.8383756766
	2048	1.9724348689	6.8753069401	6.4780672662	8.9694820864	9.6561387199	9.1694119275	8.9088830942
	8192	2.0226992998	7.0389178159	7.5738178417	11.032526433	13.518409247	1.7594678932	0.3436690826
	16384	2.013624918	7.0045494858	7.7144246185	11.228133316	13.846124914	0.8452953768	0.613973333
	32768	2.0022049122	7.0063594703	7.8142612515	11.510026941	13.631800141	1.3344072977	1.0732666501

Table 6: The speed up factor of the dynamic task assignment algorithm. The maximum speed up was over 13.

The maximum speed up factors exceeded 13 and the minimum speed values were less than 1. The typically low speed up of the use of cores above 16 is likely caused by communication time exceeding computational time. An interesting outlier occurs with images of 2048 pixels by 2048 pixels. For images of 2048 pixels by 2048 pixels, the speed up factor was barely changed between 16 and 32 cores. This implies that the size of the buffer being passed has a great effect on the latency incurred by the extra message passing. During the test, rows for each image were passed to a worker as they finished work on the last row they were given. Perhaps the passing of segments of rows to workers would overcome the latency caused by the extra communications with more cores; each message would be smaller and would take less time to copy and transfer between buffers. Another important measure of the performance of parallel algorithms is the efficiency. The efficiency of the dynamic task assignment algorithm is plotted in Fig. 8.

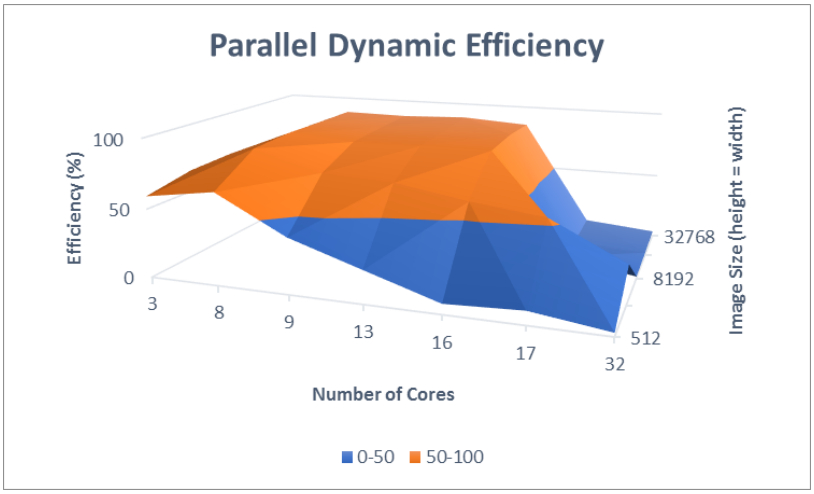

Figure 8: The efficiency of the dynamic task assignment algorithm. The efficiency consistently approaches 100% until the number of cores exceeds 16. At this point the efficiency drops off.

For the most part, the efficiency is maximized at 16 cores. It is important to note that the efficiency levels for the dynamic task assignment algorithm are consistently higher than the efficiency for the static task assignment algorithm up until 16 cores. The maximum efficiency of the dynamic task

assignment algorithm is greater than the maximum efficiency of static task assignment also. Table 7 shows the efficiencies of dynamic task assignment.

Efficiency for Dynamic Task Assignment

	Cores					
Image Size		512	2048	8192	16384	32768
	3	59.146751443	65.747828965	67.423309993	67.120830601	66.740163739
	8	66.55030919	85.941336751	87.986472699	87.556868573	87.579493379
	9	40.157542828	71.97852518	84.153531575	85.715829094	86.825125017
	13	23.660464348	68.99601605	84.865587943	86.370256275	88.538668775
	16	7.2775038127	60.350867	84.490057791	86.538280714	85.198750883
	17	9.6337575827	53.937717221	10.349811136	4.9723257459	7.8494546925
	32	2.6199239893	27.840259669	1.073965883	1.9186666657	3.3539582815

Table 7: the efficiency percentages of dynamic task assignment.

The maximal efficiency achieved by the dynamic task assignment algorithm was 88.54% on an image with dimensions of 32768 pixels by 32768 pixels processed by 13 cores. Static task assignment reached a maximal efficiency of 67.79% when only utilizing 3 cores. The dynamic task assignment algorithm is better able to utilize system resources than the static task assignment algorithm.

Conclusion

Both the dynamic task assignment and static task assignment parallel algorithms can outperform their sequential counterpart. However, depending on the size of the image and the number of cores both algorithms can fall victim to their communication time exceeding the additional division of the computations between more cores. In some cases, the parallel algorithm can and will perform worse than the equivalent sequential algorithm.

When comparing the parallel algorithms, dynamic task assignment is more efficient and faster than static task assignment. This even holds true when the dynamic task assignment algorithm is using less resources than the static task assignment algorithm. The static task assignment algorithm, however, behaves in a much more linear fashion. The speed up of the static tasks assignment is more predictable. Whereas the static task assignment speed up scales more or less consistently with the number of processors, the dynamic task assignment speed up experiences a major drop after 16 cores. This can be accredited to two things. First, the dynamic task assignment algorithm passes entire rows and the passing of entire rows can require large buffers to be copied and transferred across the network. The passing of subsections of rows may or may not improve the scalability of the algorithm past 16 cores. Second, the use of larger images may prove to make better use of 32 cores and beyond for dynamic task assignment.

In conclusion, both of the parallel algorithms offer a significant speed up from the sequential algorithm on images that are large enough. The dynamic task assignment algorithm was more efficient than the static task assignment algorithm and should be preferred in most cases.