# Report on Project 2

# Dynamic and Static Task Assignment.

CS415

March 14, 2017

Evan Su

Version 2.0

# Embarrassingly Parallel Speed Up and Efficiency

## Overview:

The goal of these test is to determine the speedup and efficiency of dynamic and static task assignment. These benchmarks are measured through the calculation of the Mandelbrot set. Three separates programs are written to measure sequential time, static assignment time and dynamic time.

## Test Methodology:

Five trials are performed to get accurate data. The lowest and highest of each test is ignored. Each trial consists of 4 different core sizes (2, 4, 8, 16) and 7 different square sizes (500, 1000, 2000, 4000, 8000, 16000, 32000). In total, 28 unique tests are performed for each trial. The time for the sequential is used as a basis for all other calculations. For further control variables and other information on each program please see below.

-Sequential

Timing starts when the first pixel is calculated and ends when the last pixel is finished.

-Static Assignment

Timing starts when the first pixels is calculated and ends when the last row is received. Important note on timing, the calculations for which rows to calculate is not included in the timing. The master does equal row calculation as the slaves

-Dynamic Assignment

Timing starts when the first row is sent and the last row is received. The master node does not perform any pixel calculations. The master node only handles the which rows need to be sent to slaves.

## Data Analysis:

Sequential vs Parallel

The sequential timing increased linearly as shown in Graph 1.1. These results were expected as the time increased exponentially. Static timing is shown in Graph 2.1 and Dynamic timing is shown in Graph 2.2. Both methods show that the time decreases as the number of cores increase. The speedup of both methods around 2 cores was approximately 1.5. As the number of cores increased, the speedup increased. The speedup can be seen in Graph 3.3 and 3.4. At this point, the difference between static and dynamic is very noticeable and the analysis will be discussed more in depth in the Static vs Dynamic sections. The efficient of both methods had various effects as the cores increased as seen in graph 3.1 and 3.2

Static vs Dynamic

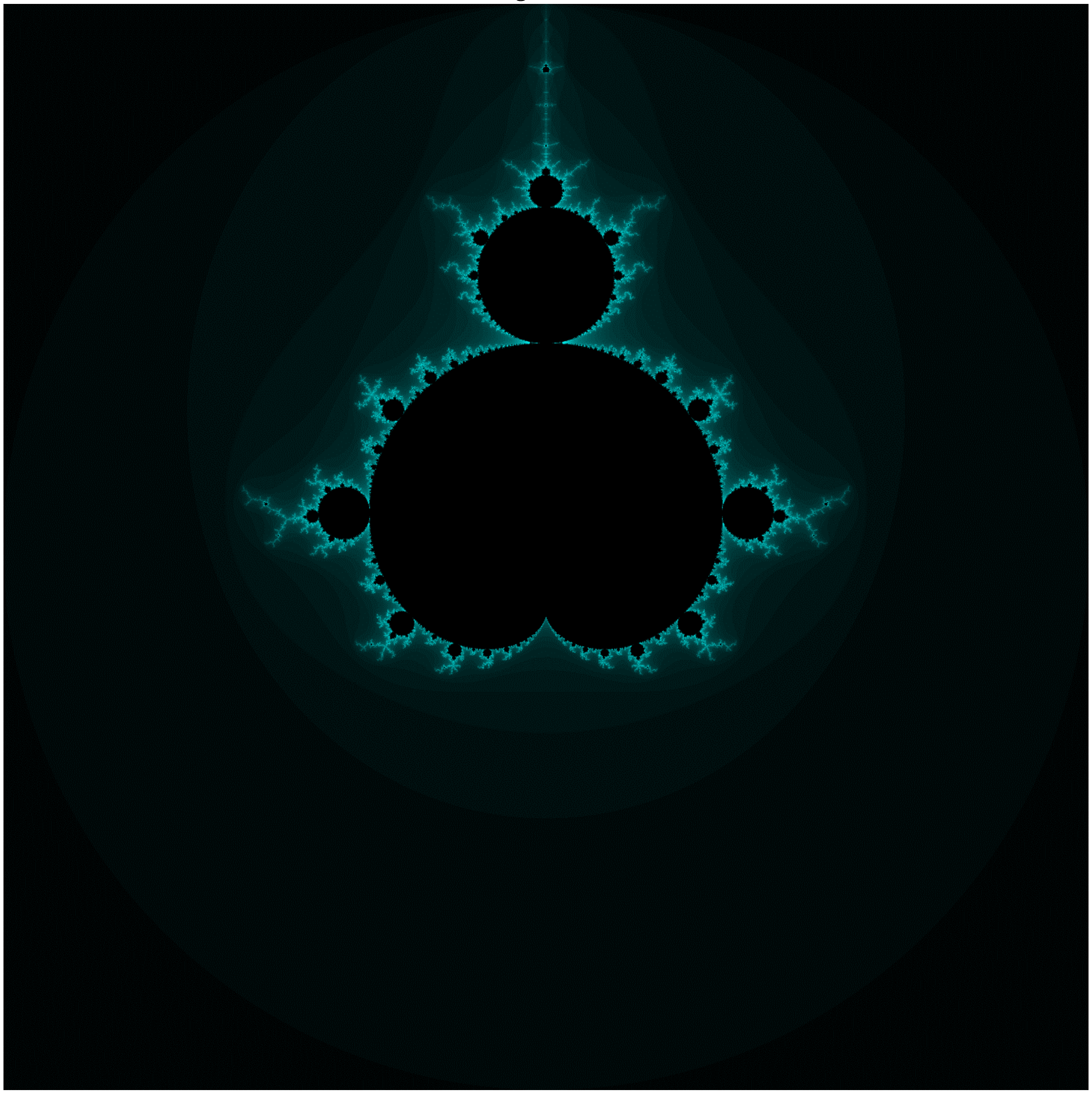
The most noticeable difference between the two is that dynamic decreases faster as the size of the image increase. The higher change in time is likely caused by the algorithm difference between the two. Dynamic uses load balancing to increase efficiency of all slaves while static must wait for all slaves to finish. This effect is seen in graph 2.1 and 2.2. This difference is also noticeable in the speedup analysis and efficiency analysis. The efficiency of the dynamic increases as the number of nodes increase. The exception is when dynamic has 16 cores and process an image size of 500x500 pixels. The efficiency was the same as static at this point. The efficiency of static decreases as the number of nodes increase. Graph 4.1 and 4.2 illustrate this effect. The speed up of static reached at most 5 in the tests performed. The speed up for dynamic reached up to 15 for 16 cores in the tests.

## Conclusion:

Dynamic assignment appears to be the best method for parallel computing if the data is embarrassingly parallel with some communication. In this Scenario, dynamic has the benefit of higher speed up due to load balancing. Static assignment appears to suffer for efficiency loss as it must wait for all slaves to finish. Given enough cores, the efficiency for static and dynamic would be effectively the same as the communication between the two are roughly equal. Dynamic gives better computing power than static in cases where communication is significant.

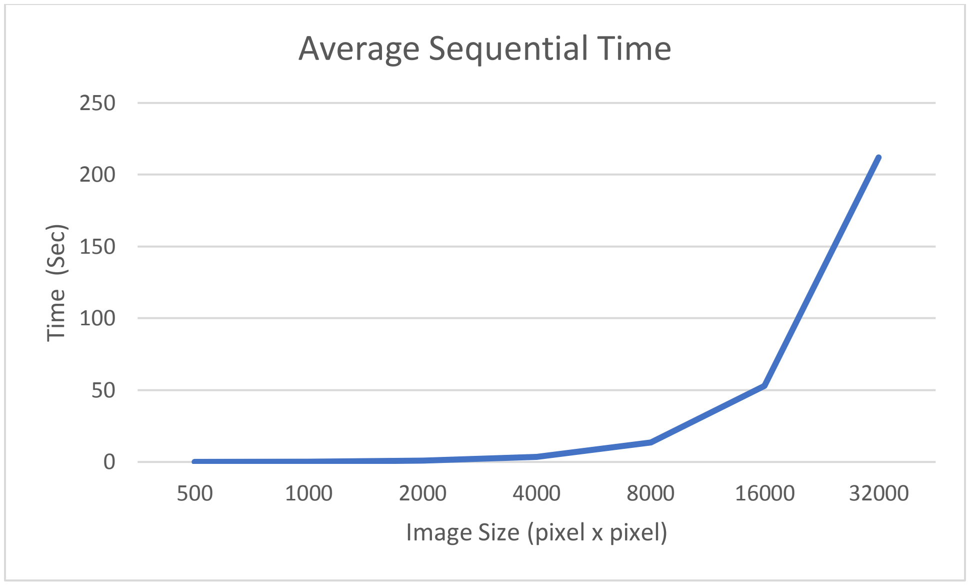
## Graphs and Tables

Figure 1.1



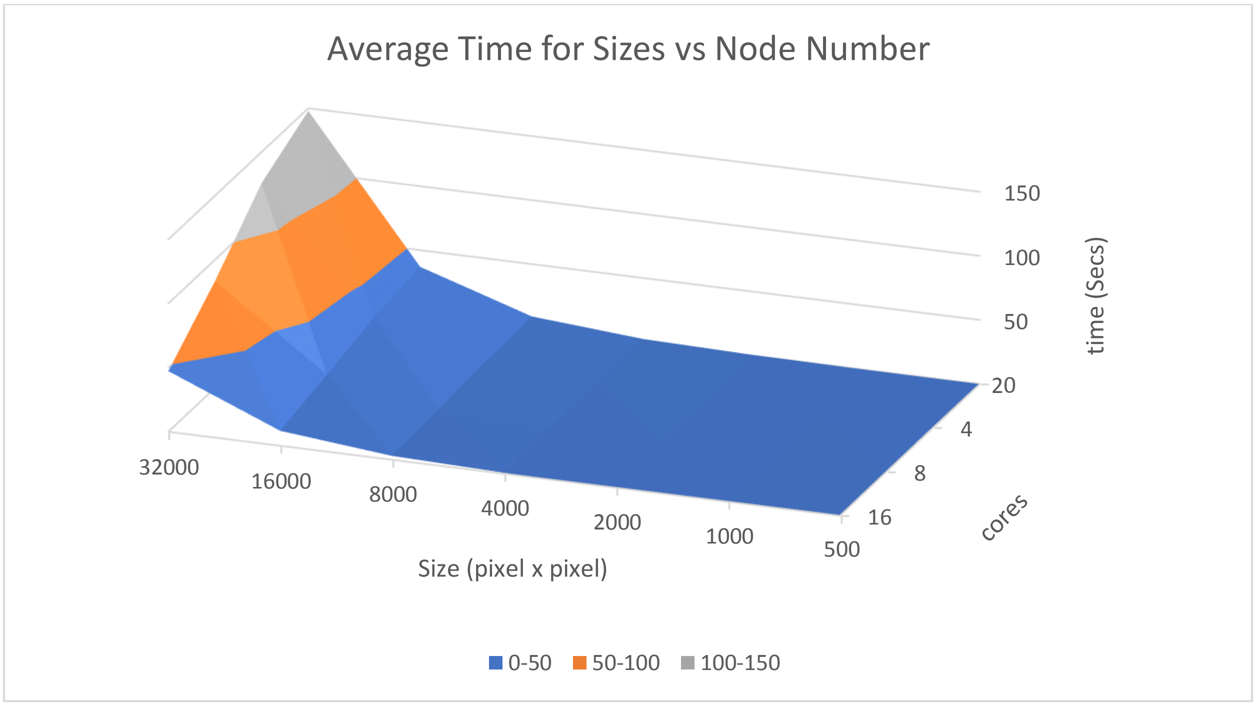
8000 x 8000 Mandelbrot image produced by the program.

Graph 1.1



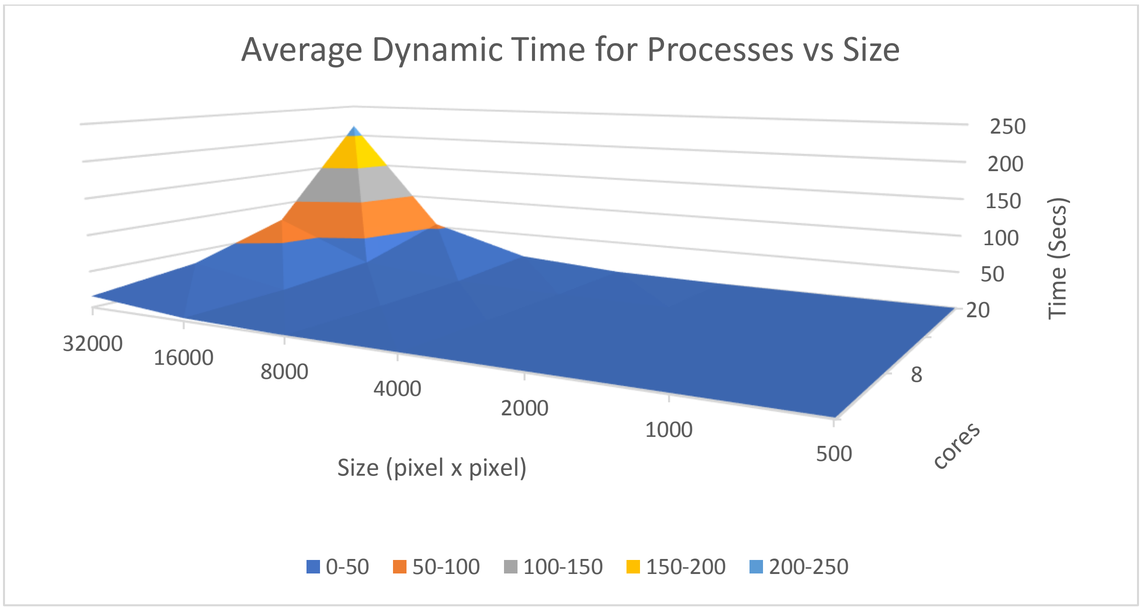
The sequential time increases exponentially as the size of the image increases.

Graph 2.1



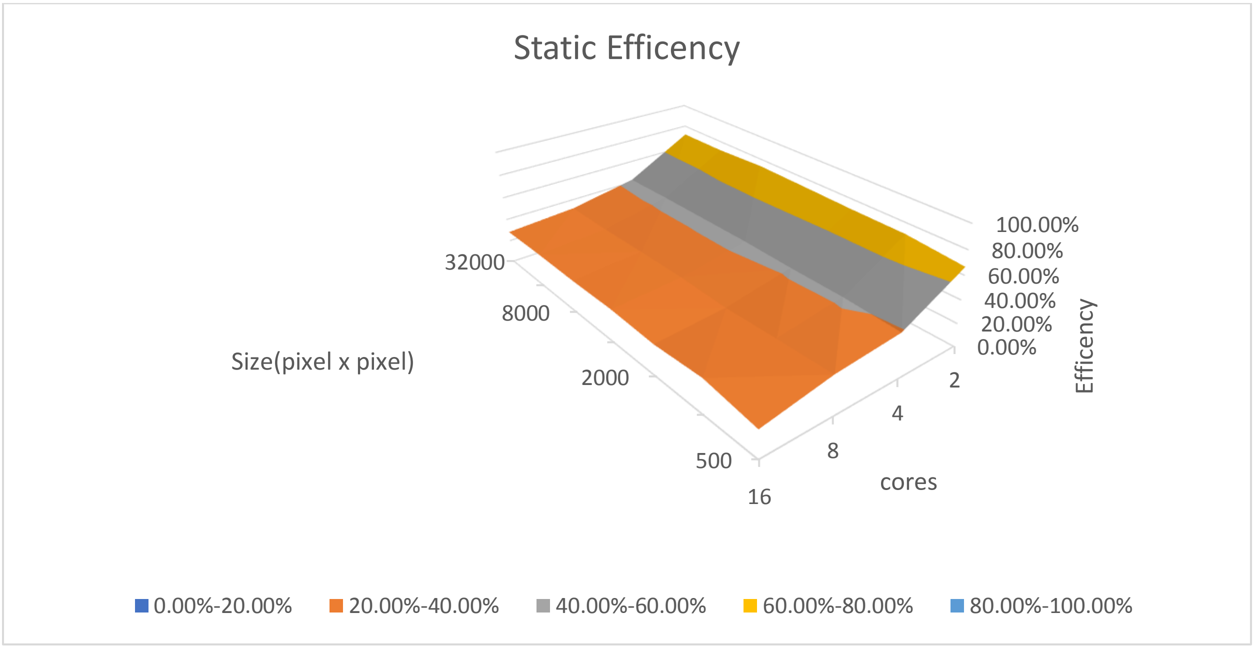
The time of static assignment decreases with number of processes and increases linearly as size increases.

Graph 2.2



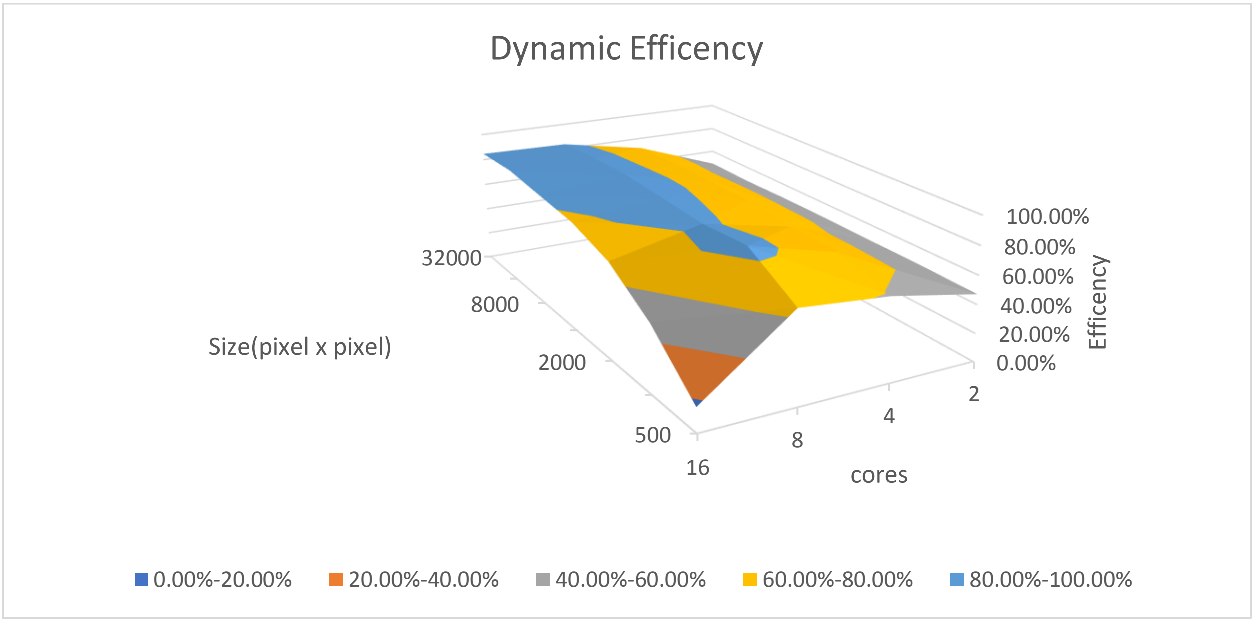
The time of dynamic assignment decreases with number of processes and increases linearly as size increases.

Graph 3.1



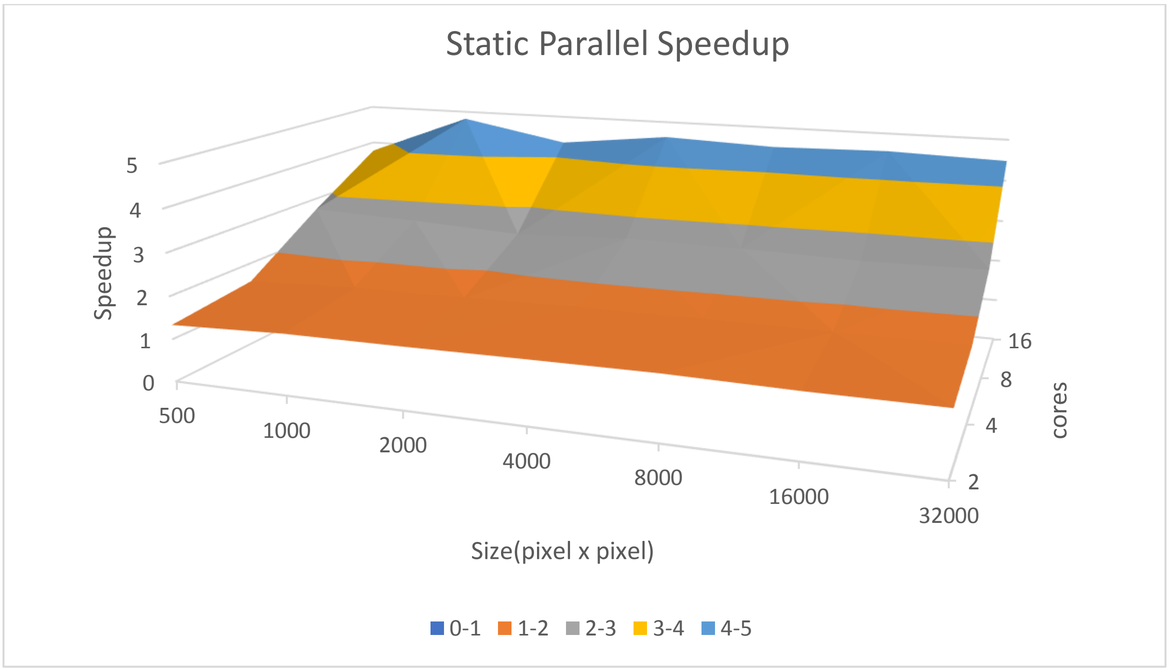
The efficiency decreases as the number of cores increase. The cause is likely due to the mass communication at the end of the program from slaves to the master.

Graph 3.2



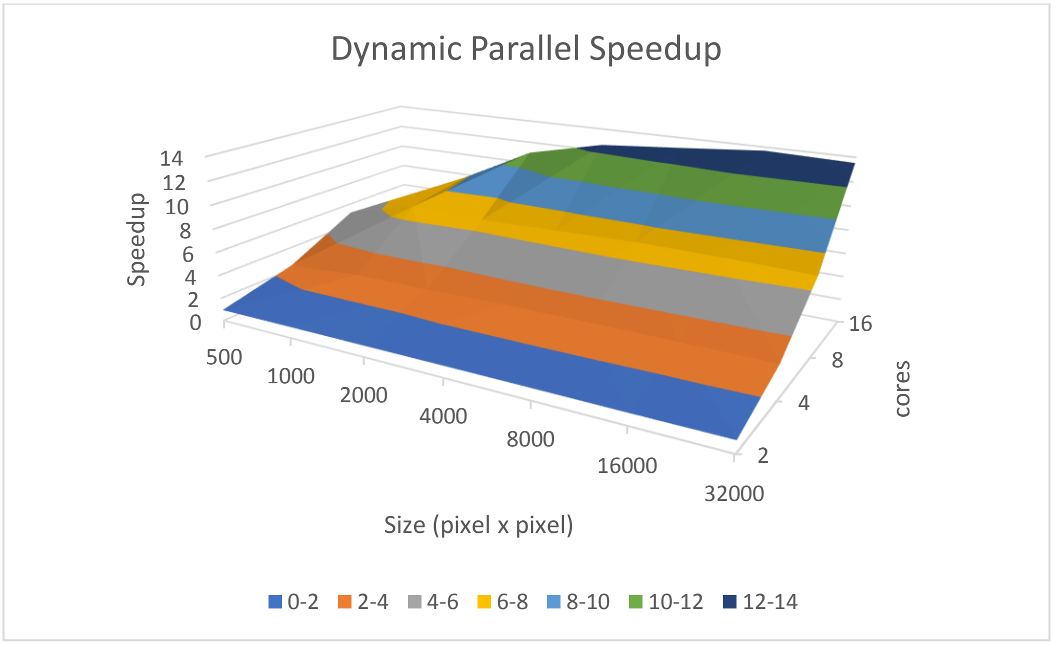
The efficiency increases as the number of cores increases. The efficiency increase is not seen at size 500, 16 cores. The likely cause is that communication time dominated over calculations. The minor efficiency drop from 8 cores to 16 cores is caused by the introduction of box to box communication time.

Graph 3.3



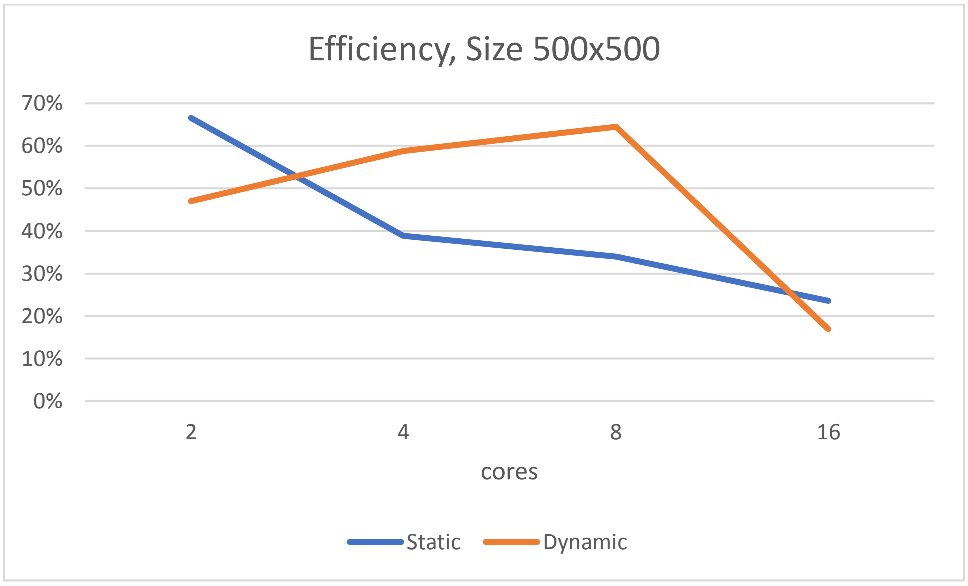
The speed up increases as the number of cores increase. Based on the slope of the graph, adding a single core will increase the speedup by 0.4.

Graph 3.4



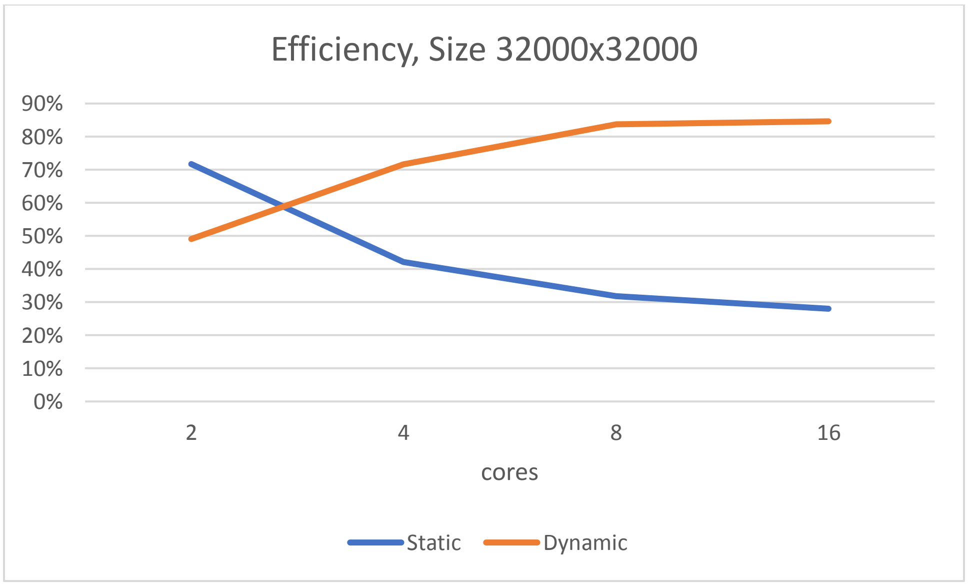
The speedup increase as the number of cores increases. The efficiency increase is not seen at size 500, 16 cores. The likely cause is that communication time dominated over calculations. In Dynamic assignment, all cores are working until the last row is complete and sent.

Graph 4.1



The efficiency at size 500x500 pixels is show above. Dynamic is more efficient until 16 cores. At 16 cores, Static and Dynamic are around the same efficiency percentage. The large drop in efficiency from dynamic is caused by the introduction of communication time between boxes and the small amount of overall calculations.

Graph 4.2



The efficiency at size 32000x32000 pixels is show above. Dynamic is more efficient as the number of cores increases whereas static decreases as cores. The static assignment has all cores finishes calculations around the same time and waits for the master core to receive all the data from the slaves. Dynamic assignment has the master node receive data as the slaves finish with a row.