Wesley Chen

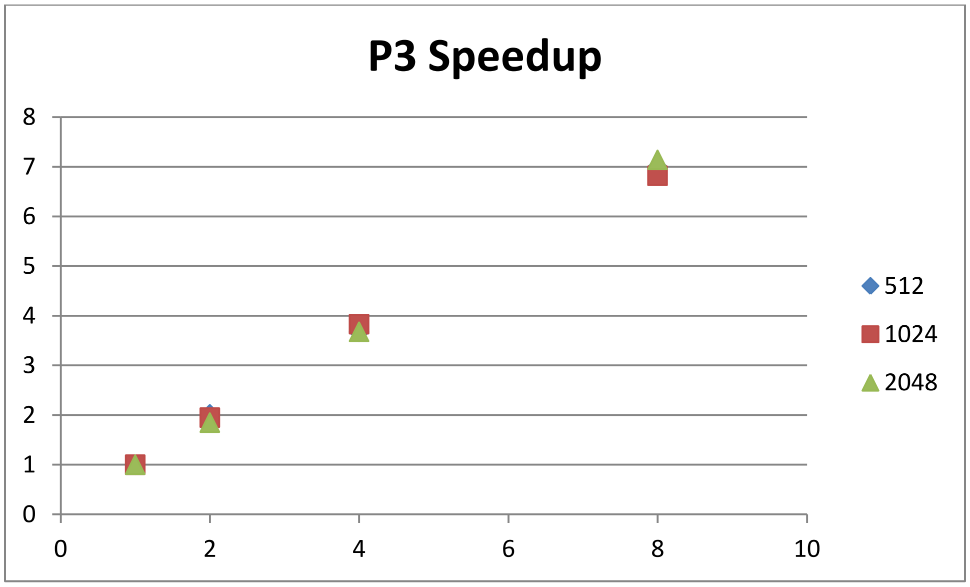
CS205 HW 2 P3

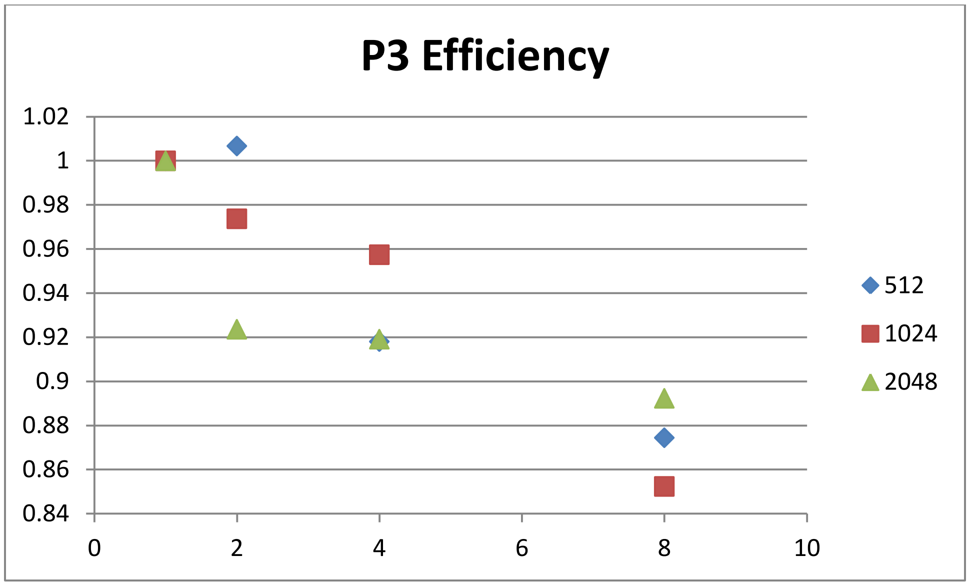
All of the reconstructed images give the same image within an arbitrarily set tolerance for each pixel of 1 e-7 (I guess I could have gone lower but…) This was calculated using a root mean square iterating through all the pixels of both images.

As we can see from the attached graphs and data, there are some trends. However, I will note that there seemed to be some variability in the time the programs took to run – which may throw off my inferences drawn from the data. In particular is the impossible >2 speedup with 2 cores (efficiency >1 magic), but I ran this once or twice, including the serial version and the numbers were similar. This may be an issue in the timing implementation. Also, the larger image sizes took much longer to run and I didn't run them multiple times.

As expected, larger image sizes resulted in a smaller decrease in efficiency as there was more to do for more processors – we can see from the data and graph that the efficiency decrease per additional core was steeper for the smaller image size of 512. But strictly by the numbers, the smallest image gave the fastest speedup and the best efficiency since the smaller task may divide better. But due to the variation of runtimes discussed above, these trends are only what I observed and sought to explain – there are some outliers.

Below is the tabulated data and graphs of varying image size 512, 1024 and 2048 with P values of 1, 2, 4, and 8 (some of the graphs due to Excel's silly large icons, makes it hard to see, look also at data:





Data:

512	1	2	4	8
Time	54.89372	27.26384	14.94844	7.847074
Speedup	1	2.013426	3.672205	6.995438
Efficiency	1	1.006713	0.918051	0.87443

1024	1	2	4	8
Time	201.7372	103.5922	52.67727	29.58516
Speedup	1	1.947417	3.829682	6.818863
Efficiency	1	0.973709	0.95742	0.852358

2048	1	2	4	8
Time	710.8186	384.7941	193.3394	99.57457
Speedup	1	1.84727	3.676533	7.138555
Efficiency	1	0.923635	0.919133	0.892319