NSC miniproject 1

Kazim, Kocan
kkocan19@student.aau.dk

1 Comparison between the algorithms

All the algorithms were run on a Ryzen 5 3600 at 4GHz with 32GB of RAM at 3000MHz. All implementations were run with a pixel density of 1000 making the array have a shape of (3000, 3000) and there were run 100 iterations. The naive implementation of the Mandelbrot algorithm had an execution time of 144 seconds whereas the vectorized version was 6.5 seconds which is 22 times faster. The unrolled version of the vectorized version was in the margin of error and had little difference. When the iterative function had the numba.jit decorator, the execution time fell down to 2.7 seconds which is a 2.4 times decrease in execution time over the vectorized version or a 53 times decrease from the non-numba iterative version. When the numba.jit decorator was used for the vectorized version it did not give a decrease in execution time, but a small increase instead.

The naive, vectorized, and numba implementations were then made to run parallel with the multiprocessing library using pool. Here there have been tested on 2, 4, 8, and 12 processes with chunk counts of 2, 4, 10, 12, 100, and 1000. This would make the chunk sizes 1500, 750, 300, 250, 30, 3.

By the testing done with all the different chunk sizes and processor numbers, having a chunk size of 3 was the optimal size as the execution time was the lowest or in the margin of error in all algorithms. Interestingly in the vectorized versions these being with and without numba there was a sharp decrease in execution time when going to a chunk size of 3 whereas there is mostly not much of a difference in the others.

There is mostly not much of a speedup outside of doing the naive version in parallel, with the exception being the aforementioned vectorized version with a chunk size of 3. Looking at the speedup for the naive version of the algorithm in parallel using 2 processors and the chunk size of 3 gives $S_2 = \frac{144.21s}{73.835s} = 1.9531$ which is an efficiency of $E_2 = \frac{1.9531}{2} = 0.9766$. Using 4 processors the speedup becomes $S_4 = \frac{144.21s}{41.269s} = 3.4943$ $E_4 = \frac{3.4943}{4} = 0.8736$

$S_8 = \frac{144.21s}{27.382s} = 5.2665$ $E_8 = \frac{5.2665}{8} = 0.6583$

$S_{12} = \frac{144.21s}{23.372s} = 6.1702$ $E_{12} = \frac{6.1702}{12} = 0.5141$

As seen from these equations the efficiency is highest when only using 2 processors and it drops when more processors have to be used.

Taking a quick look at speedup for the vectorized version with chunk size 3

$S_2 = \frac{6.5554s}{1.7443s} = 3.7582$ $E_2 = \frac{3.7582}{2} = 1.8791$

$S_4 = \frac{6.5554s}{1.4684s} = 4.4643$ $E_4 = \frac{4.4643}{4} = 1.1160$

$S_8 = \frac{6.5554s}{1.5757s} = 4.1594$ $E_8 = \frac{4.1594}{8} = 0.5199$

$S_{12} = \frac{6.5554s}{1.7004s} = 3.8543$ $E_{12} = \frac{3.8543}{12} = 0.3212$

there is weirdly an efficiency above 1, which should not be possible and is highly irregular. It can also be seen that the execution time increases with more than 4 cores, meaning that the cost of creating a new process is higher than how much you gain in execution time.

There is an additional elapsed times for the different chunk counts and number processors in the folder