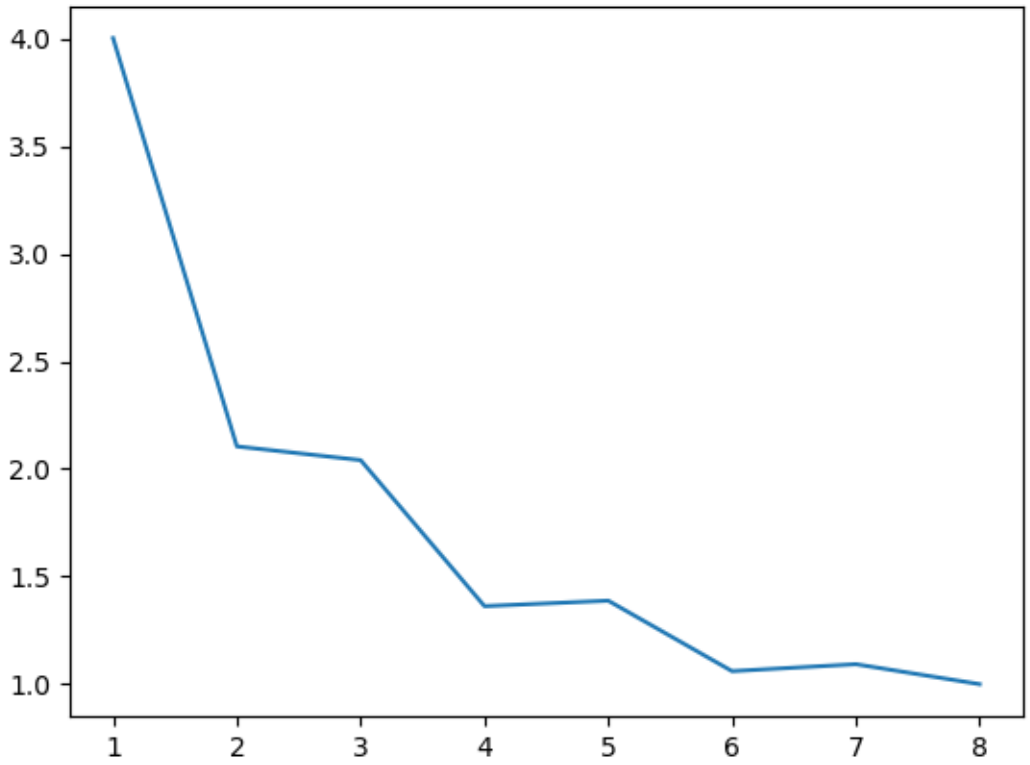
The algorithm calculates the Mandelbrot set by moving through the coordinate system and determining iterating each point to see how quickly it diverges by repeatedly applying the function the Mandelbrot is based upon to the point.

The parallelization approach taken splits all the pixels in the coordinate system up between n threads that calculate the divergence for the given number of iterations for each. By default open MP will do this statically, meaning that the points will be divided up before the threads start running and will not be redistributed if one thread runs out of computations to do. We changed the original program to allocate the number of pixels dynamically which allowing threads with nothing to do to pick up pixels from other threads. This is nice because some pixels will run take more iterations of the 3rd inner loop to determine if they will diverge or not.

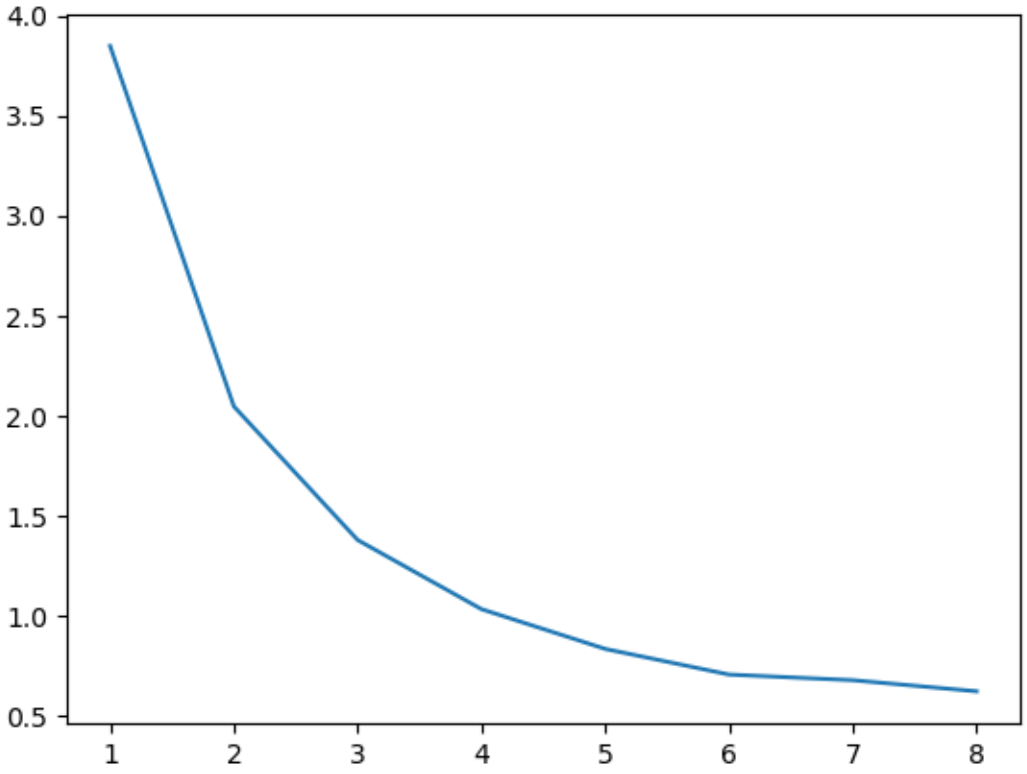
The data I have was collected on the Saturn machine, with 6 cores on an Intel(R) Xeon(R) CPU E5-1650 with hyperthreading. The CPU has 2 32k L1 caches, a 256k L2 cache, and a 15360 L3 cache.

The first plot comes from Open MP running in static mode. We dont see much improvement in some



place because the points near the center require much more iterations to find out if they diverge. The thread in the middle on odd thread numbers will still have to do a lot of work and bottle-neck the system since the workload cannot be transferred.

This plot was generated with the dynamic option for parallelization in Open MP. The improvement is much more consistent because the workload is shared and a thread will never sit and do nothing while there is work remaining.



The dynamic approach to parallelization works much better for this algorithm. The points white pixels near the center require more iterations, so splitting the ranges of pixels up at the start leaves one thread with much more work than the others.