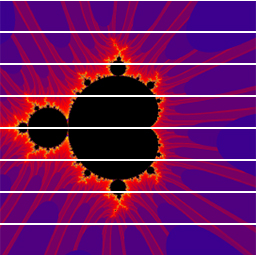
## **Fractal**

In our solution we chose to divide the picture horizontally into several parts, where each thread was assigned to one part. To do this we used our own struct called Data which told the thread where it should start and how far it should go. In the picture below this is visualized. The white lines show the borders for the parts. For example, the first thread goes from the top to the first white line, the second thread goes from the first white line to the second white line, and so on.



Results of parallelization compared to the sequential version:

Parallel implementation with 1 thread: 0.99x as fast.  
Parallel implementation with 8 threads: 2.44x as fast.

This shows that threading can give a massive speed-up, but if used wrong it might even be slower.

## **Matrix multiplication**

To parallelize this we chose to divide the final matrix into 4 different sub-matrices. These sub-matrices consists of the result of two different calculations, which means that we have four parts that needs two calculations each. Assigning one thread to each calculation gives us eight threads.

To do this we used the struct Data, which holds the start coordinates for the first and second array, how wide/high the block is, and the start coordinates for the third array, which holds the result.

C:\Users\Jonas\Desktop\blocks.jpg

Each of the red boxes shows one calculation that is performed in one thread. A1, B1 etc are the different sub matrices.  
  
Parallel implementation with 8 threads gave a speed-up of 24.36x faster than the sequential version, which is a lot faster!

## **Q-Sort**

Fyll på!