Task 1 a)

We let every thread work on a subset of the rows in the dwell-buffer matrix. This results in an even workload given that the resolution is high enough, and as given that the matrix is stored row by row in memory, it should perform well with the cache.

Speedup was approx. 3.25, which is disappointing given that there were at least 12 threads running. This is probably simply due to much of the execution time being spent elsewhere.

Task 1 b)

Mariani multi-threaded border calc:

Border calculation is a cheap operation compared to the overhead added by threading and atomic operations.

This adjustment resulted in a speedup of 0.39, effectively making the algorithm slower.

Task 1c)

Naive parallelizing.

The hardware has a hard cap on how many threads can run in parallel. The lab PC used for this exercise had 12 cores each capable of running two threads in parallel for a total of 24 threads.

The number of threads will rapidly increase as each threads spawns one thread for every subdivision within its area. This quickly results in many more threads than the hardware allows, and the operating system would then deschedule the exceeding ones and reschedule them when the resources are available. As the cost of switching between threads is nontrivial and is done a lot of times as a result of this change, the amount of added overhead far outweighs the benefit of parallelizing.

Increasing the block dimension means the call tree reaches a leaf sooner. which decreases the overhead of keeping a huge number of threads.

The number of colours, zoom-level and the position will affect the largest size of same coloured squares. If it is known, through some heuristic, that there will be quite large same-coloured, scaling down the subdivision should make them easier to find. To counteract the recursion-depth this would give, increasing the block dimension as well could give positive results.