Parallel programming

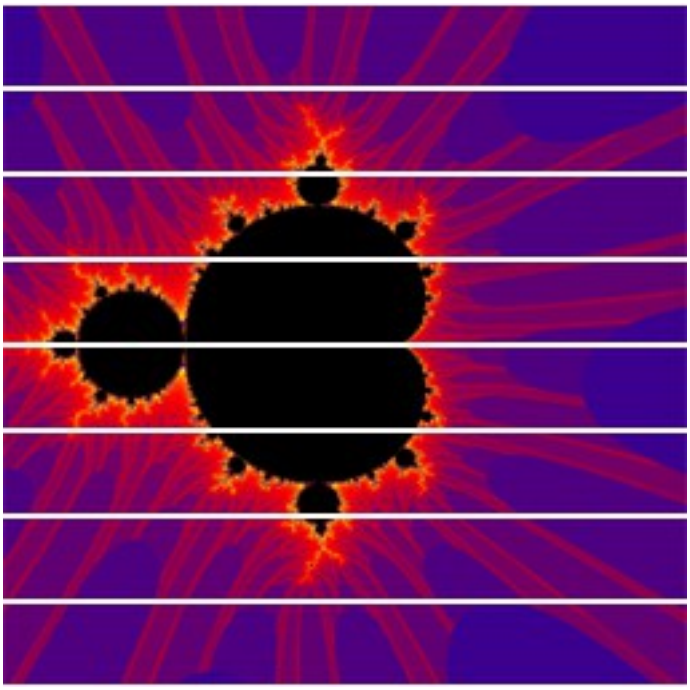
Assingnment 2

Mattias Lunderot & Jonas Andersson

20/12 – 2015

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.

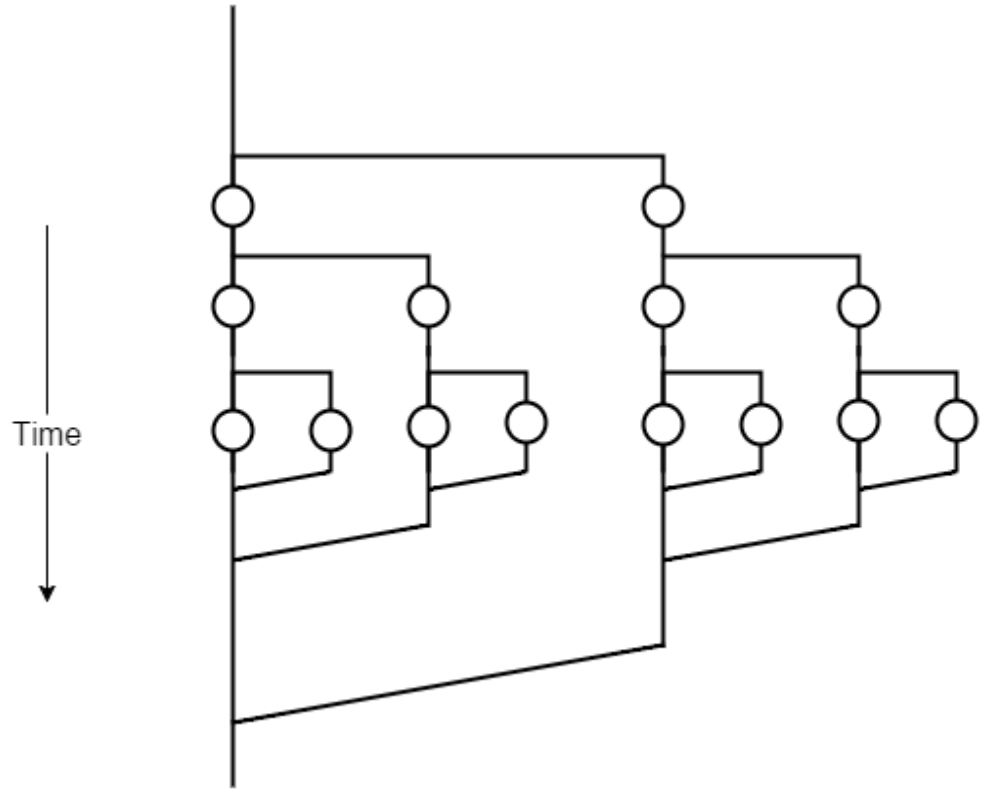
$$\begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} \begin{bmatrix} A_2 & B_2 \\ C_2 & D_2 \end{bmatrix} = \begin{bmatrix} A_1 A_2 + B_1 C_2 & A_1 B_2 + B_1 D_2 \\ C_1 A_2 + D_1 C_2 & C_1 B_2 + D_1 D_2 \end{bmatrix}$$

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

For our parallel quicksort implementation we decided to create one thread per level.



In the picture below, each circle represent a sequntial part of the quicksort algorithm. The bold leftmost line represent the main thread. The program splits half of the array and hands it of to the other thread. The main thread then continues on with the other half.

This happens recursivly until a maximum depth of three is reached and then the algorithm is only executed sequentially. The threads then join back up with their parent thread until the main thread has the sorted array.

The parallel implementation with 8 threads gave as speedup of 3.34x faster than the sequential version.

The parallel implementation running in one thread ran at 99% speed compared to the sequential implementation.