TDT4200: Parallel computing

Assignment 5: Pthreads and OpenMP

Name: Martin Rebne Farstad

Task 1

(a) Baseline measurement from a lab computer with parameters RES=4096 and TRADITIONAL=0 (Escape-Time) gives a wall-time of 19.56 seconds, while TRADITIONAL=1 (Mariani-Silver) gives a wall-time of 6.83 seconds.

(b) The runtimes differ for figure 1 and figure 2. These pictures are taken from running the program on my own computer. The reason for this can be explained as follows. It was mentioned in the assignment text that "Any shape in the rendering area having the same dwell value on all border pixels, can be filled using this dwell value". This has consequences for the runtimes of the two images, as the images show which blocks need to be calculated (the red blocks) and which are instantly filled with the dwell values of the borders (the black blocks).

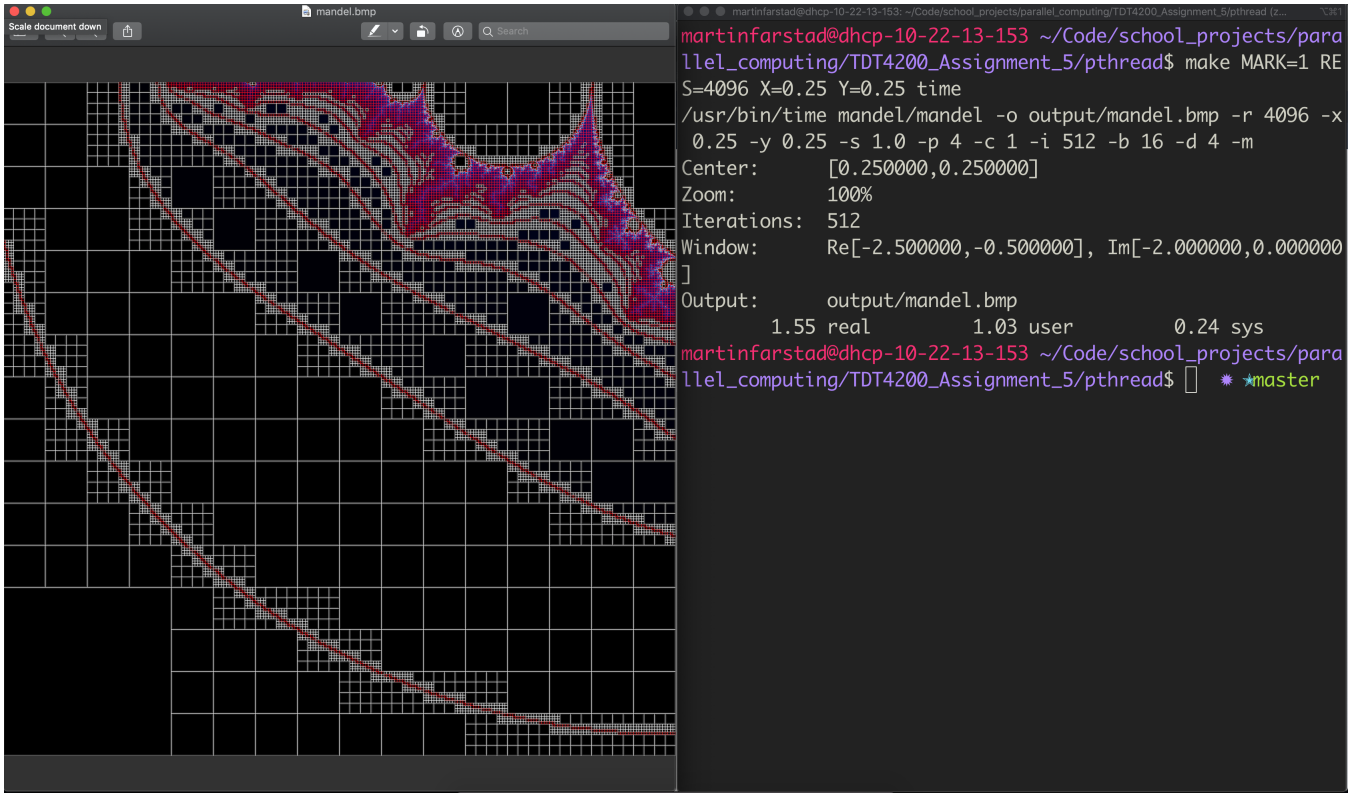

Figure 1: Parameters X=0.25 Y=0.25

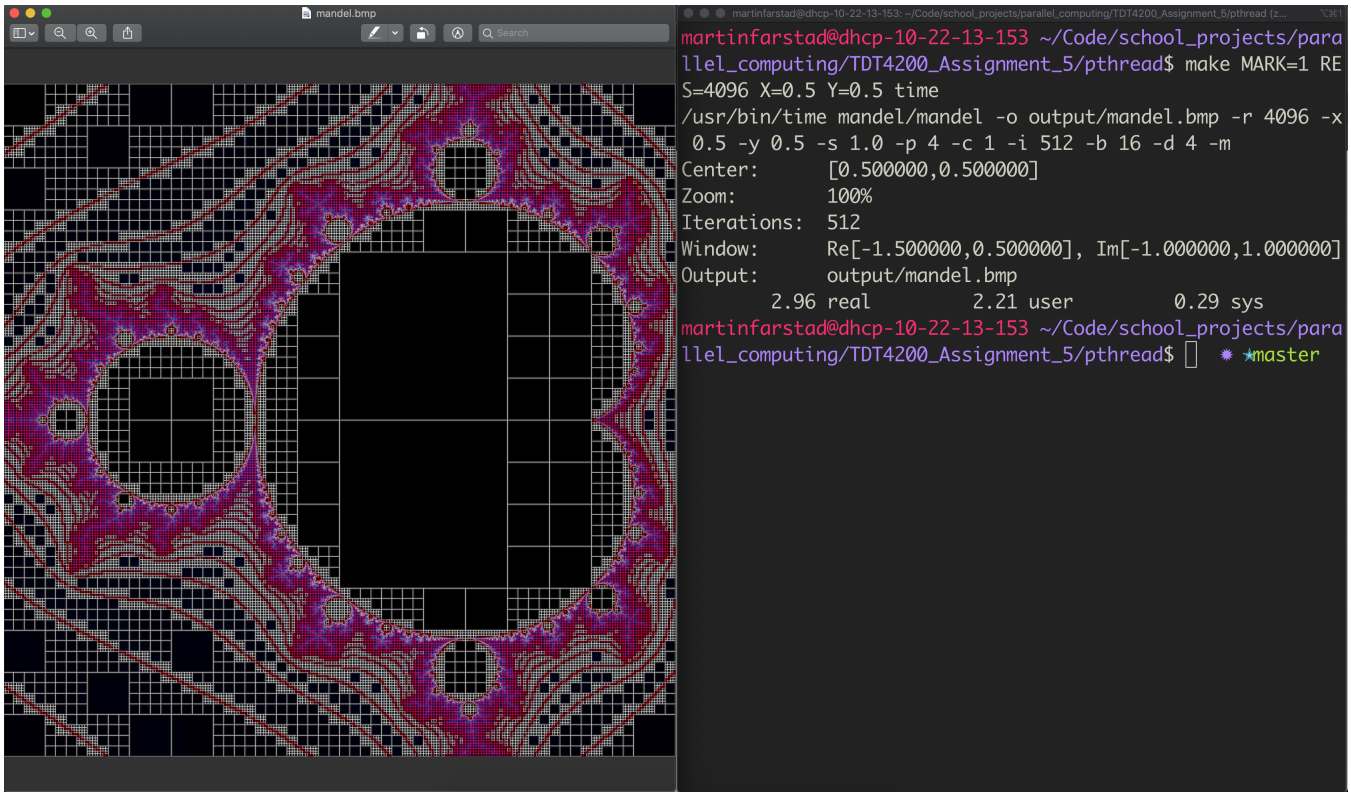

Figure 2: Parameters X=0.5 Y=0.5 (default)

Task 2

(b)

Threads	*Runtime (seconds)*	*Speedup*
2	6.03	1.13
4	5.62	1.22
8	5.48	1.25

Table 1: Mariani-Silver

Threads	*Runtime (seconds)*	*Speedup*
2	12.28	1.59
4	9.09	2.15
8	7.80	2.51

Table 2: Escape-Time

By doubling the amount of threads, the program gets a small but noticeable speedup which scales better for Escape-Time than Mariani-Silver.

Task 3

Using omp_get_wtime() I measured serial_mxm() with a runtime of 0.3527 seconds. I inserted the timing functions before and after the switch clause that decide which function to use (serial, omp or blas). This is because I only want to measure the runtime of these functions and not the rest of the code. Also, by setting the measurement functions before and after the switch clause I only had to write them once.

Task 4

The runtime using OpenMP with 8 threads is 0.0568 seconds. That is a speedup of 6.21, which is close to the amount of threads running the program. The overhead of spawning additional threads and synchronizing them

makes this speedup not equal to the amount of threads.

Task 5

The runtime using BLAS is 0.0503 seconds. This gives a speedup of 7.01 compared to the serial version and a speedup of 1.13 compared to OpenMP.

By looking at the source code of dgemm, it is shown that each value from the B matrix is only accessed $n * k$ times, while each value in A is accessed $n * k * m$ times for a total of $n * k * (1 + m)$ accesses. In the OpenMP program, every value of A and B is accessed $m * n * k$ times for a total of $2 * n * k * m$ accesses. These accesses are split between the eight threads and therefore gives a total of $(n * k * m)/4$ accesses per thread. Since OpenMP uses shared memory each thread may have to synchronise to write parts of the memory. For the default parameters, it seems like the threads can't write the result concurrently and therefore the memory becomes a bottleneck for the OpenMP function. As a result, the BLAS function has a better runtime for the default parameters.