Workshop fixes

msp exercise

For the java implementation of the MSP algorithm it has been parallelised using the equivalent of concurrent futures. This means that an executorservice has been defined such that processes can be given to this executer and executed while the main process continues. The result from futures can be accessed at any time but if the get function is called it will wait until a result is generated.

dct exercise

Parallel

jit

First jit had to be investigated, apparently the numpy arrays being worked on was of different datatypes preventing jit from enabling nopython mode, going into object mode.

The fix was to go through every operation that should be run with jit and changing the datatypes of the numpy arrays, while making non-numpy arrays into numpy arrays. this fixed our issue with jit not going into nopython mode.

cuda

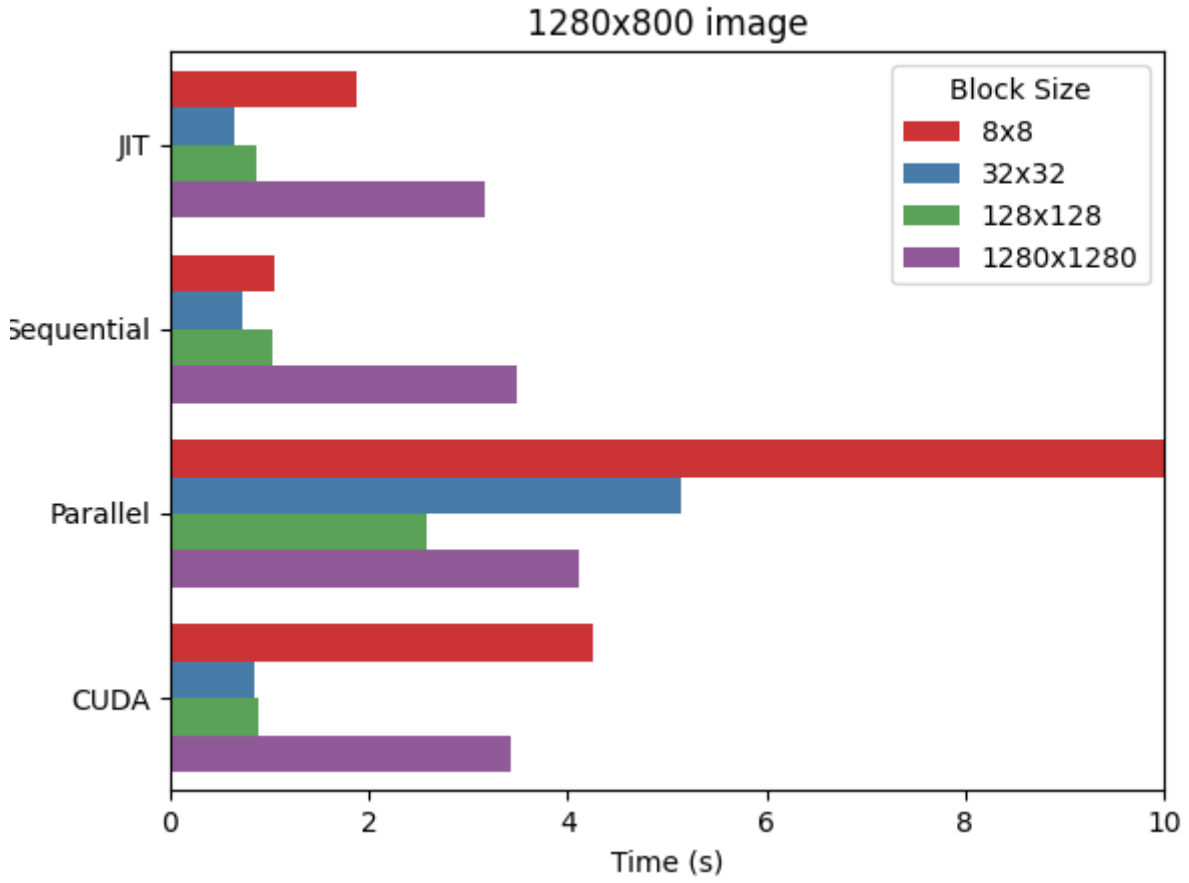
Then cuda was also an issue because it was weirdly fast, the suspicion here was that there should be so much communication cost that it should be significant overhead.

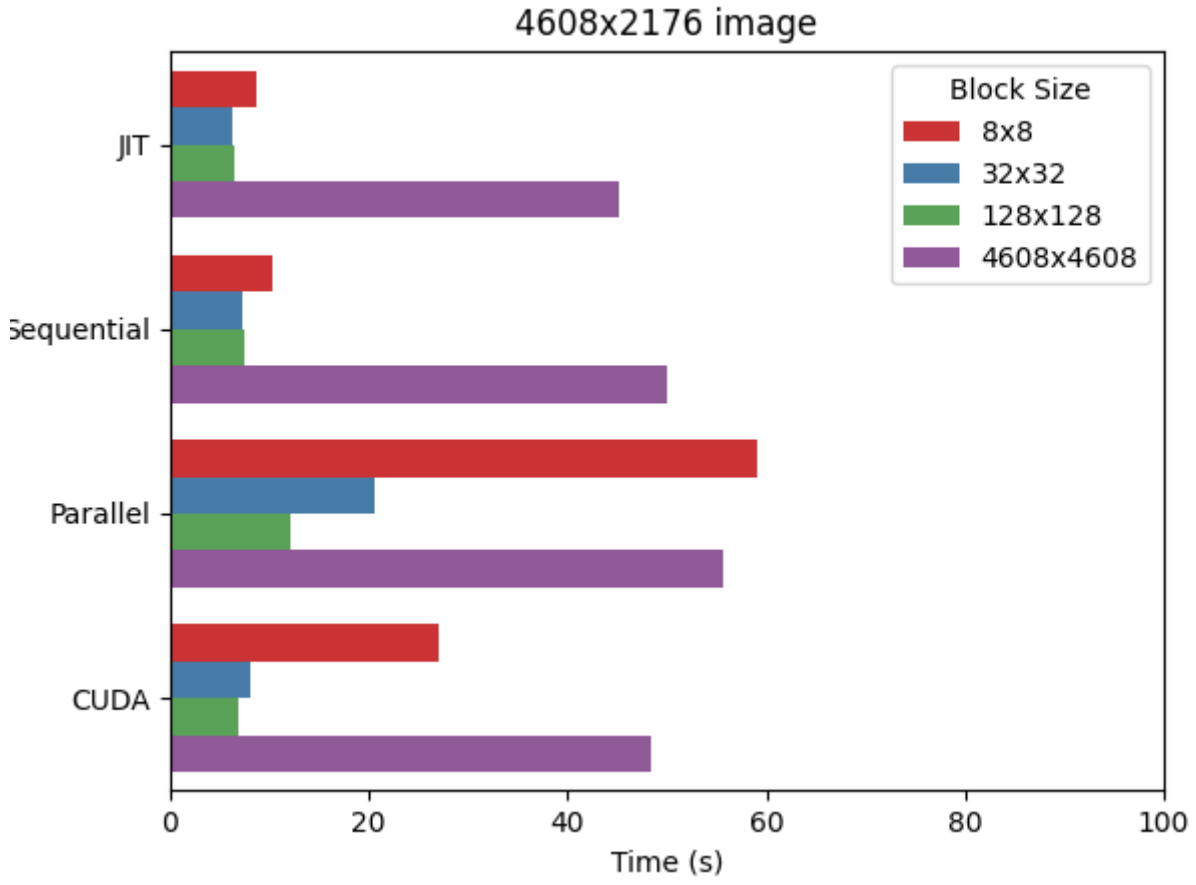
The cuda implementation was tested and apparently the first time it was tested it was not utilizing the GPU, rather the program was running on a cpu. fixing it was a matter of two things;

first every single tensor should have a device assigned to it

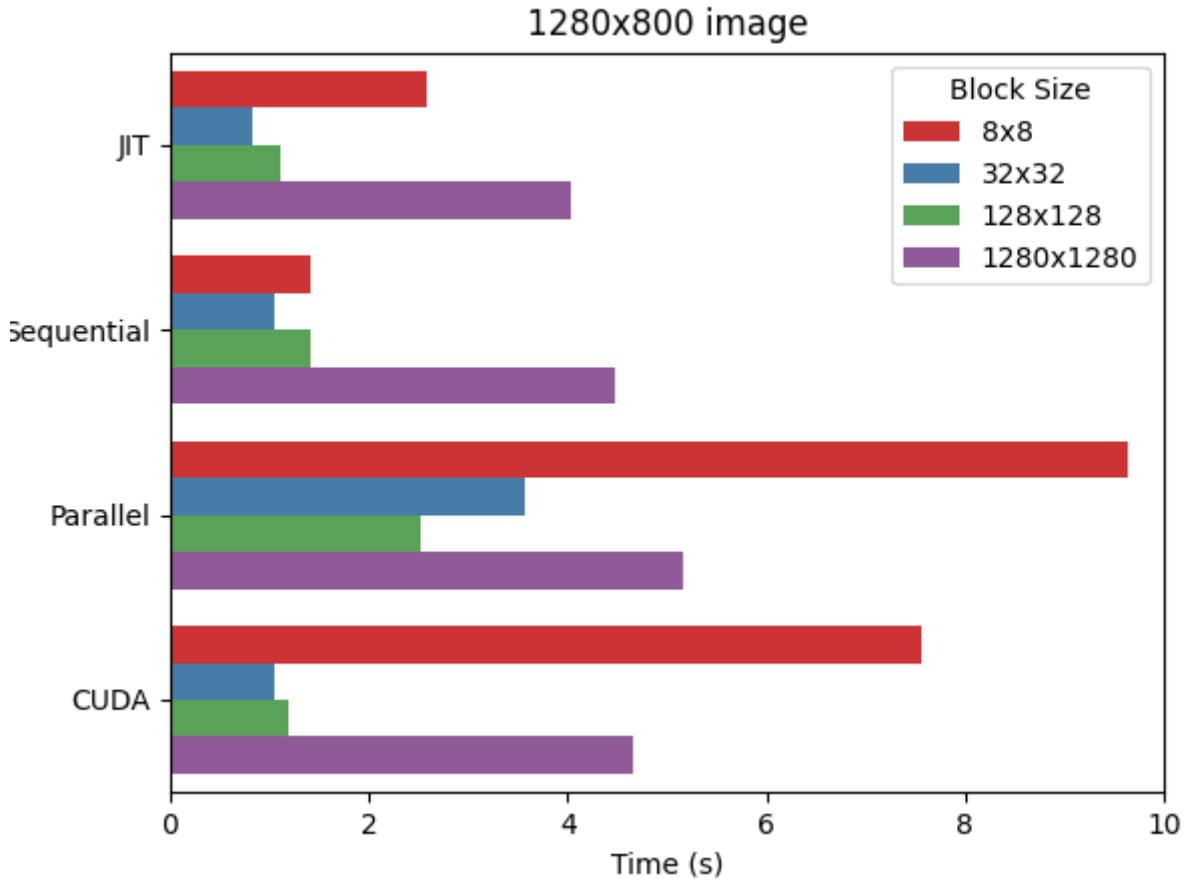
Second the tensors were defined in the multiply and hadamard functions which means that for every loop in the function, the program would create new tensors for the dct and the quantization matrices. these tensors could have been defined once at the beginning of the program, thus generating a significant performance improvement.

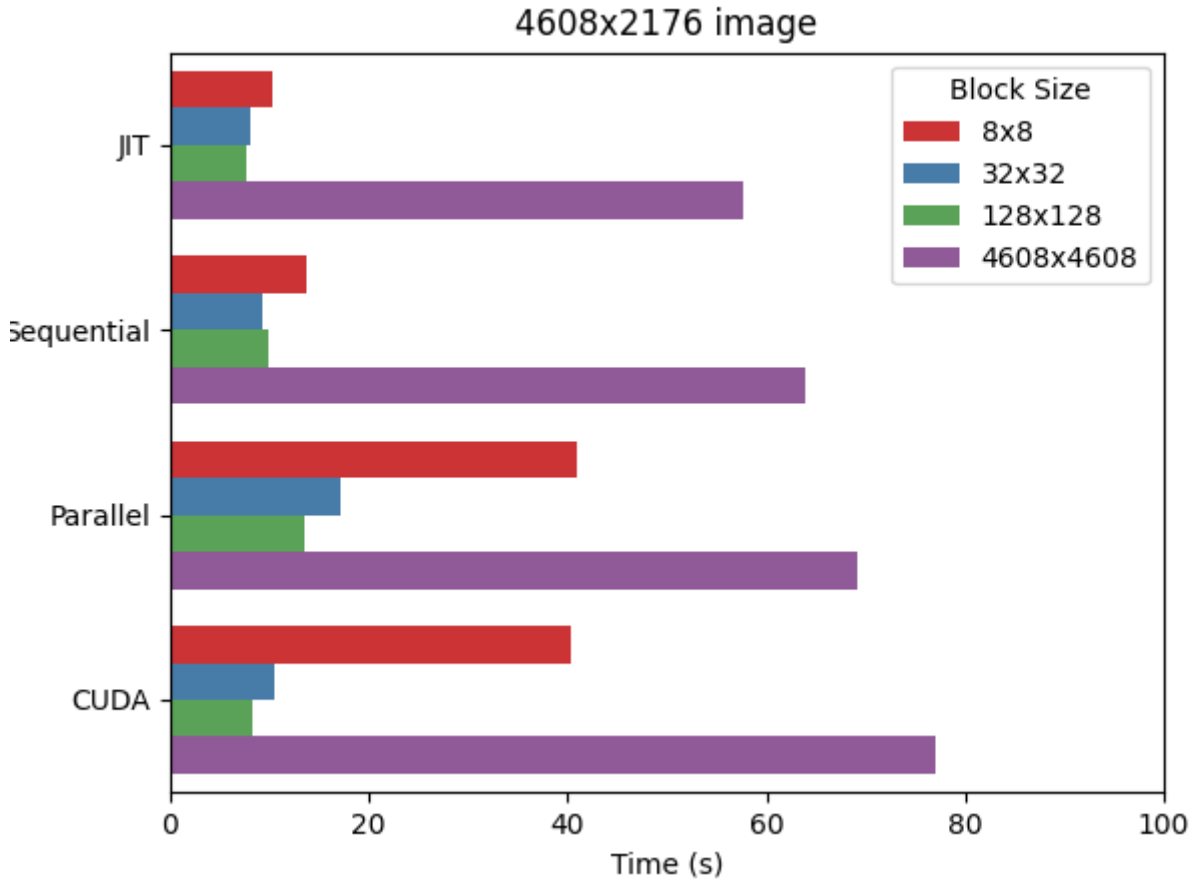
Results with a Geforce GTX 1080 GPU + 6 core cpu





Results with a Nvidia Quadro m3000m integrated GPU + 4 core cpu


1280x800 image
Block Size
8x8
32x32
128x128
1280x1280
JIT
Sequential
Parallel
CUDA
0
2
4
6
8
10
Time (s)


4608x2176 image
Block Size
8x8
32x32
128x128
4608x4608
JIT
Sequential
Parallel
CUDA
0
20
40
60
80
100
Time (s)