Parallelization Strategies

At the suggestion of some posts on campuswire, I added pragmas from the inner loops outward. I started by tiling the i, j, h, and w loops, and adding `#pragma ACCEL parallel` to the inner loops for those as well as the p and q loops. This tiling lets me cache chunks of the weight, input, and C arrays. I used a `#pragma ACCEL pipeline` on the w loop since that was where I started caching the chunks.

Optimization Evaluation

Loop tiling is good because then the chunks of each array that I need for a tile can fit entirely in cached memory. The parallelization on the inner loops lets me do computations in parallel. The pipelining outside of where I cache my chunks means that those data transfers are done in parallel with computation.

Differences from Lab 3

In lab 3, I was doing a “top-down” parallel approach where I got rid of outer loops and each thread executed the inner loops. Here, I parallelize the inner loops, and each component is going to be reused for the iterations of the outer loops. This is because parallelizing the outer loops would make the circuit level logic more complicated; each flattened iteration of the outer loop would have to internally keep track of its inner loops, but if I flatten the inner loop, the logic for the outer loop only needs to be on the FPGA once. Also, this approach uses coarse-grain parallelism to overlap data transfers with computation, whereas in lab 3 this was done by swapping threads.

FPGA resource Utilization

Cycles - 250297830 (1001.191ms)
LUT - 223309 (18%)
FF - 321260 (13%)
BRAM - 2142 (49%)
DSP - 2081 (30%)
URAM - 25 (2%)

The most used resource is BRAM