Vivek Sivakumar
303652195

Lab 5 Report

Briefly explain the parallel strategies you applied in this lab. How is it parallelized? What is the expected communication overhead?

Before starting the lab I looked into both development flows to see which one I wanted to utilize. I decided to use the Merlin compiler because it provided many of the suggested optimizations (kernel bit-width, bust memory access, etc.) automatically without me having to write them up myself. Given the timeline of the project I wanted to incorporate most of these suggested optimizations but I was skeptical of my ability to master them all to the degree I'd need to. Once I went with Merlin I just had to figure out how to pipeline the loops in my kernel to increase performance the most. First i had to swap the p and q loops with the h and w loops to remove the data dependency (so Merlin could unroll the innermost loop), then I added pipeline pragmas to the second to last loop to maximize pipeline speedup. Looking at the optimizations in the generated OpenCL code from the Merlin compiler, it seems to use 512 bit width's for the kernel arguments, it does burst transfers of all arrays into their local copies, it partitions the memory cyclically across different blocks to allow for multiple pipe stages to access data without collision, and finally it tries to automatically unroll the innermost loops.

The expected communication overhead is all in the data transfer onto the memory blocks of the FPGA chip, since we are not running multiple threads but instead one pipelined datapath we only need to worry about getting the data into the pipeline and off it. With the Merlin optimizations this means getting data from the host program to FPGA global memory, then from global to local memory when applicable, and finally from global back to the host memory for verification.

Compare your program in terms of the execution time and energy efficiency with your lab 3 and lab 4.

My program for this lab is significantly slower than my lab3 and lab4 implementations because it relies on pipelining the sequential version for most of its speedup whereas lab3 was able to distribute the computation to 32 different threads which executed simultaneously; my lab4 implementation distributed the problem to even more threads making it the fastest of the 3.

My lab 5 code is likely the most energy efficient out of the 3 and significantly so, for lab5 I am running the program on a customized architecture for that program. This means that the datapath is optimized specifically for my algorithm with no extraneous control flow to decipher between lanes in the datapath. The bit width of all arguments to the kernel, the burst memory access, and the pipelining also help to provide power savings. Compare this to a CPU (or a GPU) which has an ALU that can execute many different operations and needs large control flow logic to determine which operation to do, the CPU is also running multiple processes at once on multiple cores meaning greater power consumption.

Discuss how much memory is used at private, local, and global levels of your best configuration.

Cin, Cout, weight, and bias are all held in global memory at their full size, each array has a local copy that is used for faster memory access, this local copy may or may not be the size of the original array. Cout_buf_local only has one row of Cout at a time, weight_local also has only one row of weight at a time, and Cin_local also has only one row of Cin at a time. Meanwhile bias is only a 1-D array so all of bias is held in local_buf.

<u>Challenges you encountered and how you solved them.</u>

My first challenge was understanding the FPGA as I had never encountered one before, I had to go through the lecture notes and some outside reading to get a good grasp of exactly what I was programming. Second I had some problem just setting up Merlin, understand the directory structure, the log files, the different commands, as well as what exactly it was doing. Once I understood Merlin I was able to create a flow and do HLS, but I ran into my most time consuming bug: the simulation. For whatever reason whenever I ran the simulation on my account it would fail and complain that it could not find the Xilinx tools, in the end a friend graciously let me use his account to run the simulation and ensure program correctness. Finally generating the bit stream was a challenge simply because of the time that it took.