University of Pennsylvania
Department of Electrical and System Engineering
System-on-a-Chip Architecture

ESE532, Fall 2018 HW7: Restructuring for Accelerator Wednesday, October 17

Due: Friday, October 26, 5:00PM

So far, we have accelerated small kernels with straightforward mappings to hardware. In this assignment, we will look at two applications that that require more effort, namely the compression pipeline from homeworks 2, 3, and 4 and a digit recognition benchmark. You can find the sources for this homework on the course website. We made a few small modifications, but the code should look familiar. The data stream is here.

Collaboration

In this assignment, you work with partners that we assigned. You can find the assignment on Canvas in the *Partners* map under the *Files* section. In the event that the partner assignment does not work out, contact the instructor or TA as soon as possible. Partners may share code and results and discuss analysis, but each writeup should be prepared independently. Outside the assigned groups, only sharing of tool knowledge is allowed. See the course policies on the course web page `http://www.seas.upenn.edu/~ese532` for full details of our policies for this course.

Homework Submission

1. **Warmup**

   As we have seen so far, the `Filter_SW` function is by far the function that consumes most of the clock cycles. However, its acceleration is also more complex than the other functions, so we will start our endeavor more gently by mapping `Differentiate_SW` first.

   (a) Create a new Vivado HLS project and add the provided source files. Use a clock period of 7 ns. Write a new function that invokes the hardware implementation that you will write later, `Differentiate_HW`, and exits your program with a value of 1 if the output is not correct. Verify that your test function works. Include the function in your report.

   (b) How many times does `Differentiate_SW` load each pixel on average? (3 lines)

   (c) Can we use streaming in `Differentiate_SW` to handle arbitrary large frames? Assume that we do not change the code except for adding pragmas and changing the dimensions of the data. Motivate your answer. (1 line)

(d) We could store pixels that are used multiple times in a buffer that is mapped to a local memory. Assuming we still produce the output pixels in the same order, what is the smallest buffer that we can use? Motivate your answer. (3 lines)

(e) In some iterations, we must write a value to the local memory and read multiple values. An array is typically mapped on a BRAM, which has only two ports. Consequently, we need more bandwidth than the BRAM offers. Give two ways in which we could resolve this issue. (4 lines)

(f) Implement the function `Differentiate_HW` such that it loads the input pixels only once and sequentially. Verify your code using your test function. Include the `Differentiate_HW` function in your report.

(g) Pipeline the loop body of your implementation with an II of 1. What is the latency that Vivado HLS predicts? You can ignore whether Vivado HLS meets the clock period or not for now.

(h) On a microprocessor, branches are generally undesirable because they introduce delays when they are predicted wrong. Why is this not a problem in an accelerator?

2. **Accelerating the Filter**

In this part, we will accelerate `Filter_SW`.

(a) Does `Filter_horizontal` offer any opportunities for data reuse? Motivate your answer. (3 lines)

(b) What is the optimal order for traversing the input data (column-wise or row-wise)? Assume that the input and output are stored in a BRAM. Motivate your answer. (3 lines)

(c) Create a function `Filter_horizontal_HW` that is a version of `Filter_horizontal_SW` that you modified based on the insights from the previous two questions. You don't have to use the streams at this point. Include the code in your report.

(d) Pipeline the loop body of `Filter_horizontal_HW`. Verify your code using the test function that you wrote. What is the latency that Vivado HLS predicts? You can ignore whether Vivado HLS meets the clock period or not for now. (1 line)

(e) Let's continue with accelerating `Filter_vertical_HW`. We could store pixels that are used multiple times in a buffer that is mapped to a local memory. Assuming we still produce the output pixels in the same order, what is the smallest buffer that we can use? Motivate your answer. (3 lines)

(f) What is the optimal order for traversing the input data (column-wise or row-wise) with respect to FPGA on-chip memory usage? Assume that the input and output data are stored in a BRAM. Motivate your answer. (3 lines)

(g) Create a function `Filter_vertical_HW` that is a version of `Filter_vertical_SW` that you modified based on the insights from the previous two questions. You don't have to use the streams yet. Include the code in your report.

(h) Pipeline the loop body of `Filter_vertical_HW`. Verify your code using the test function that you wrote. What is the latency that Vivado HLS predicts? You can ignore whether Vivado HLS meets the clock period or not for now. (1 line)

(i) Write a verification function for `Filter_SW`, similar to the one in question 1a. Verify that your test function works. Include the function in your report.

(j) Create a function `Filter_HW` that connects both parts of the filter together. Store the intermediate results in a local array. Include `Filter_HW` in your report. Use the default data movers.

(k) What is the expected latency of `Filter_HW`? (1 line)

(l) We could replace the local array in `Filter_HW` with a stream. Assume that the stream requires no resources for buffering. What impact do you expect that will have on the resource consumption? Quantify your answer. (3 lines)

(m) Replace the local array with an `hls::stream` object and insert a `dataflow` pragma into `Filter_HW`. The `hls::stream` class is declared in `hls_stream.h` Modify the remaining functions as necessary. Note that you don't have to inline `Filter_horizontal_HW` and `Filter_vertical_HW` explicitly. The tool typically inlines them automatically, or you can use the `inline` pragma to obtain the same result. Include `Filter_HW` and any other significant changes in your report.

(n) What is the predicted latency of `Filter_HW` now? Make sure you verify your code. (1 line)

(o) Import your code into SDx. Set the optimization level of the SDS++ compiler to `-O3`. Add `Differentiate_HW` and `Filter_HW` as hardware functions and equip both functions with `access_pattern` pragmas to inform the compiler that they access data sequentially. Comment out the invocations of the hardware functions that you used for verification because each invocation may result in an additional accelerator instance. Note that commenting out a function that calls a hardware function is not sufficient. Report the speedup of both accelerators (together).

(p) What is the speedup of the entire application?

The speedup of the acceleration is still poor with respect to the results of the vectorized implementation. We haven't exhausted all the restructuring and optimizations we could potentially perform. It is worthwhile to think about what else you could do to achieve a greater speedup.