Accelerating convolutions using 1-directional dataflow accelerator

February 10, 2020

Abstract

Custom silicon is proving to be a suitable way to accelerate neural network execution and deploy them to resource-constrained devices. One of the ways to utilize custom accelerators is the use of systolic array accelerators to accelerate artificial neural network inference. Within systolic array accelerators, there is a huge design space for the dataflow and various networks are better suited to different dataflow networks. In this paper, a synthesizable 1-d dataflow network is presented and the improvement over a baseline is discussed.

1 Introduction

As the amount of energy required for data movement fast exceeds the energy required for the actual compute, there has been a search for methods of computation on silicon devices which attempts to minimize the data movement.
One such design is the systolic array, proposed by [1]. While the idea has been around for a while, applications successfully exploiting the computation patterns amenable to systolic arrays, have only arisen recently. Mainly, the regular and predictable compute requirements of artificial neural network computations has been proven to benefit from systolic array accelerators.
Recent works such as [2] and [3], have shown that systolic array accelerators do actually realize significant accelerations for neural network computations. These networks are widely used in image recognition, recommender systems, classification and other machine learning tasks at scale. Research has shown that these networks spend most of their time in the convolution passes and accelerating these parts can lead to massive gains.
To work around the challenges posed by the current designs, namely large energy consumption and memory bottleneck, systolic array designs aim to make maximum re-use of the input feature maps and the weights. Given that these elements can be reused over millions of times, depending on the network structure, efficient re-use can lead to large savings in energy and inference bandwidth.
In this work, we present a synthesizable systolic array accelator written in Bluespec System Verilog. It implements a distinct dataflow scheme on a 2D systolic array of multiply and accumulate units. The scheduler circuit allows arbitrary convolutions to be implemented by breaking up the convolution in smaller sections. The accelerator has an AXI4 interface which allows easy integration with a processor and the open-source design allows it to be easily integrated into experiments. The entire design is parametrizable in various regards, which will be discussed. We run tests on the software simulator by integrating it with an open source RISC-V processor.

2 Dataflow

The choice of the dataflow plays a significant role in the performance of different neural networks. A crucial insight that dataflow circuits rely on is the reuse of elements. For a given set of input feature maps and weights, there are multiple avenues of reuse of elements without realoding them from memory. Existing efforts have significantly explored mainly three types of dataflows - Weight Stationary (WS) [][], Output Stationary [][] and No Local Data Reuse (NLR) [][].

Given this, we have decided to explore a distinct method dataflow, which can be considered as a 1-dimensional dataflow with the weights being stationary. Instead of passing partial sums and input elements from the X and Y axes in a dataflow circuit, we pass the data along a single direction and keep the weights stationary.

For the purpose of discussion in this section, we will consider a 4x4 square array of Processing Elements (PE's). These PE's perform 32-bit fixed-point multiply and accumulate operations. The convolution we consider is of a 4x4 filter on a 16x16 image with stride of 1. Arbitrary convolutions can be mapped to this scheme, by breaking the convolution into smaller convolutions.

2.1 1-D Dataflow

Figure 1 details the dataflow design in our case. The blue boxes represent the PE elements. The yellow bubbles represent delay elements. The arrows indicate the flow of information. The grey boxes denote the values entering for one cycle of exeuction (64 cycles).

As shown in Figure 1, the ofmap partial sum (psum) inputs flow in a single direction within the PE array, as compared to the input in X and Y directions in [3]. The reuse of a psum elements in this case, is of a maximum of 4 times in 13+3 = 16 cycles. The shape helps in maintaining a balance between ifmap data reuse and reducing redundant PE calculations of corner elements.

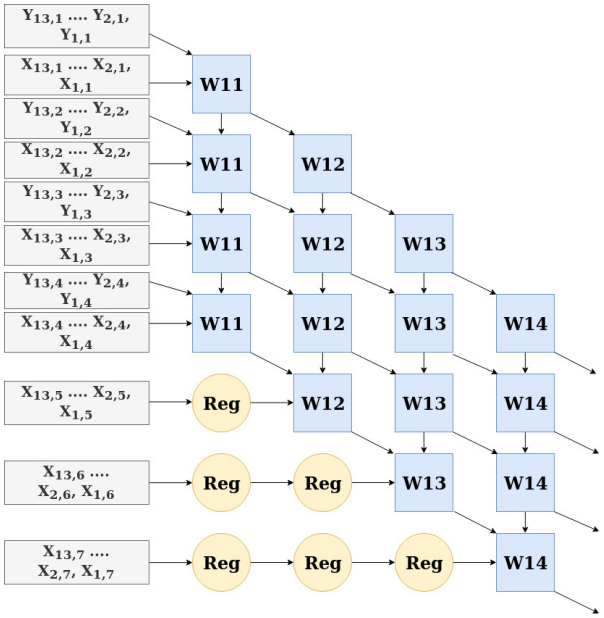

Figure 1: Systolic Array

2.2 Weight Matrix Loading

The loading of the weight matrix decides the weight reuse. The options here are to load each PE with a different weight or to duplicate the weights. According to our analysis, loading 4 weights per row leads to maximum re-use and least idle PE's.

2.3 On-chip Memory and Scheduling

The on-chip memory for the accelerators are designed using BRAM memories in Bluespec. There are BRAM's for storing the input elements and the partial sum output.
The scheduling is maintained by using FIFO's which are fed by the processor bus. Every 16 cycles, 4 weights are loaded using this scheduling scheme and every 64 cycles, one-fourth of the considered convolution is computed.

2.4 Input stream

The diagnal dataflow has 2 input streams: ifmap and weight filter. The weight filter input channel is used to load the weights every 16 cycles. The ifmap stream feeds the input ifmap elements every cycle.
The equations 1 2 are used to govern the the ifmap input streams.

After 16 cycles

$$X_{row,column} -> X_{row+1,column} \tag{1}$$

After 16*4 cycles

$$X_{row} -> X_{row,initial} \tag{2}$$

$$X_{row,column} -> X_{row,column+4} \tag{3}$$

3 Implementation and Design

In this section, we discuss the various decisions made regarding the implementation and the implications. We also discuss the various parametrizations, strategies available for our accelerator.

3.1 Implementation language

The systolic array accelerator is written in Bluespec SystemVerilog which generates the software simulator as well as the Verilog used for synthesis.
The main reason for choosing Bluespec as the language of implementation is the power of customization. Given the large number of parameters that can be customized, being able to parametrize the design and define relations between these parameters can allow rapid design space exploration. Further, the scheduling and rule analysis of the rules can make designing the control circuitry much easier and faster.

3.2 Processing Elements

The processing elements are bit-width parametrized multiply-accumulate units. These PE's can be mapped to high performance DSP circuits in FPGA's or ASIS's.

3.3 On-chip Memory and Scheduling

Small on-chip memories which is instantiated as BRAM's in FPGA's are used to hold the input elements and partial sums. These memories are small, in the order of 64 elements to store the input elements and partial sums. FIFOs are used in this design as an interface betweeen the BRAMs and the systolic array to facilitate a constant input stream of data in the required pattern.

3.4 AMBA AXI4 Interface

The accelerator uses the standard AMBA AXI4 bus interface. This makes the accelerator highly portable i.e. it can be interfaced with any device with AMBA AXI4 support. We interfaced the accelerator to an open source RISC-V processor using the AMBA AXI4 interface. An 8-bit status register in the processor memory is used to control the inputs, weight loading and fetch output from the accelerator. Once the data is written into X and W BRAMs, the computation starts asynchronously and the final results are fetched after completion. The completion of the convolution can be mapped as an interrupt or can be polled from the status register.

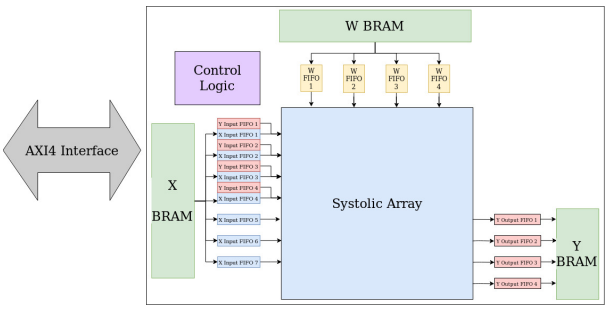

Figure 2: Accelerator Design

4 Results

For an initial evaluation of our design, we run a simple convolution on the software simulators for the system. The comparison is between the convolution code running sequentially on the core and being accelerated by the systolic array system.
The C code performs a convolution with a 4x4 filter on a 16x16 image with a stride of 1. The C code manages initialization of the input and weight matrices and the actual computation. For the case of the sequential code, simple multiplication code carries out the convolution. For the accelerator, the code makes memory mapped requests to the accelerator for reading and writing the data.
Our methodology has been to count the number of cycles required for the computation part of the convolution, on the software simulator. This can be used to benchmark the performance, once post-synthesis frequency is available.
Based on the above experiment, our seqeuntial C code took 57967 cycles for the given computation. Our systolic array accelerator finished the computation in 5941 cycles. Based on this, we claim that for the given computation we received a speed-up of 9.75x.

5 Future Directions

The first and foremost task after this is the synthesis of this circuit for FPGA's and comparing it to available designs like the NVDLA.
Another area to explore is to check the type of neural networks which work better with this 1-d dataflow design and come up with a methodology for mapping networks to hardware.
Thus in conclusion, we have presented a fully-synthesizable, parametrizable dataflow circuit with a distinct dataflow.