A Portable Tensor Processing Unit

Wesly Tonks
UC Davis
Davis, United States
watonks@ucdavis.edu

Cameron Shinn
UC Davis
Davis, United States
ctshinn@ucdavis.edu

Alan Qin
UC Davis
Davis, United States
aqin@ucdavis.edu

Skye Develasco
UC Davis
Davis, United States
sdevelasco@ucdavis.edu

Abstract

Trends in computing have led to a proliferation of Neural Network applications. Unfortunately, todays general purpose processors are not well suited for the class of computations these applications require, creating demand for a new class of processors - Tensor Processing Units (TPU). These hardware accelerators are designed with Neural Networks in mind, and allow host CPUs to offload computationally expensive tensor operations to them. We implement our own, low-power, scalable TPU intended for embedded and mobile applications, and evaluate its performance using a simulated fully connected Neural Network layer.

Index Terms

Neural Networks, Machine Learning, TPU, Hardware Acceleration

I. Introduction

With the diminishing of Moore's Law and the exponential increase in data, computing needs are out-pacing computing capabilities in general purpose processors in both client and server applications. Today's solution is the design of specialized hardware co-processors, which specialize in one or a few tasks. These hardware accelerators perform complex computations much faster (on the order of 10 - 100 times faster), and with much lower power consumption. With the increasing expense of power in all parts of the computing spectrum, it is no wonder that hardware accelerators are becoming ubiquitous.

With the creation of data accelerating at an exponential rate, algorithms to extrapolate useful information - namely Neural Networks, have become common, and are found in embedded, mobile, and data center applications. These algorithms rely heavily on matrix multiplication and convolution, both of which place heavy load on today's general purpose processors, creating the need for a Tensor Processing Unit (TPU), first commercialized by Google for data center use.

Current TPU's aim to implement the aforementioned matrix multiplication and convolution, along with common activation functions used in todays Neural Networks. This is typically done via a highly pipelined data path known as a systolic array, which can complete an NxN matrix multiplication in just $2N$ cycles. As with most hardware accelerators, TPU's must communicate large amounts of data between itself and the host, introducing undesired overhead. TPU's also have large on chip memory bandwidth, as the systolic array requires large input and output bandwidth for sustained computation.

We present a scaled down, low-power TPU implementation providing performance gains for embedded and mobile neural network applications. We aim to increase performance of fully connected layers in an application running on an ARM A9 processor. The TPU is memory mapped, and communication between the two devices is done via an Avalon bus on an Altera DE-1 development kit. The TPU implements a 16 x 16 systolic array which runs at a max clock speed of 115 MHz, yielding a noticeable speedup in fully connected neural network layers, when compared to a CPU only.

This implementation allows useful Neural Network applications such as image processing, object detection, speech-to-text, and more to be run in power and performance limited devices such as smart phones and embedded systems. The design is also scalable, allowing for a range of options per application.

II. Background

A. Neural Networks

B. Tensor Processing Units

III. Design and Implementation

A. System Architecture

We give credit to Google's TPU as inspiration for our own architecture. Given the level of design we are at, resources describing Google's TPU proved very helpful in implementing our own.

A basic overview of our architecture is shown below. Commands and data flow in from the host interface, an Avalon AXI bus. These commands are decoded into one of the following functions:

- *Write Weight Memory* - Data on the bus is written into a specified location in Weight Memory space
- *Write Input Memory* - Data on the bus is written into a specified location in Input Memory space
- *Fill Weight FIFO's* - A set of weights is read from weight memory into the weight FIFO's
- *Drain Weight FIFO's* - The set of weights currently held in the weight FIFOs is loaded into the systolic array
- *Matrix Multiply* - A set of inputs is piped into the systolic array and multiplied with the set of weights currently held in the array.
- *Read Output Memory* - A specified word from memory is read to the host.

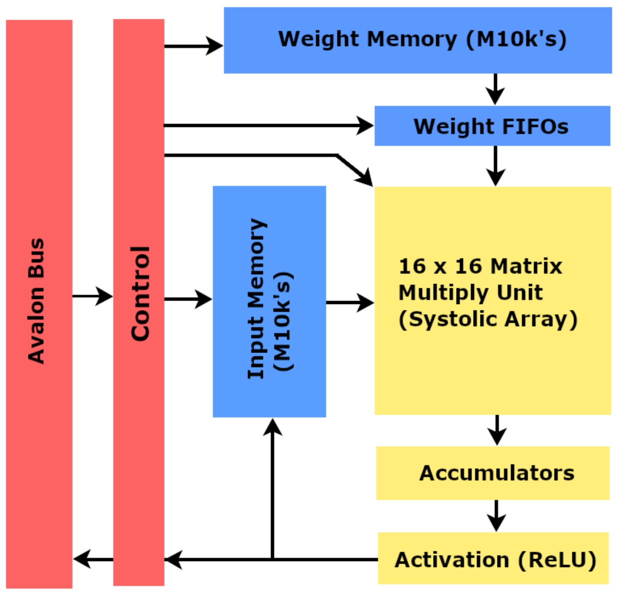

Fig. 1. High level view of system Architecture

B. Systolic Array

Our TPU features a weight stationary systolic array, meaning a set of weights may be loaded in once but used for many operations. The array is fully pipelined, performing a 16 x 16 matrix multiply in just 32 cycles. It is composed of many processing elements (PEs), which contain a small amount of memory and control logic, and a single multiply accumulate data path.

A complete Matrix Multiply starts at the top left corner of the Systolic array, and is piped diagonally downward. In the first cycle of a multiply, input memory supplies data for only the top left PE. After one cycle, the first PE activates its neighbors below and to the right, creating the diagonally downward piping.

Each PE holds one element of the input matrix in any given cycle, and passes that element to its right neighbor every cycle. The multiply accumulate result is passed downward to the neighbor below every cycle. Each PE then multiplies its input element with the weight element stored in it, then adds that value to the sum being supplied from its above neighbor.

C. Software

IV. Results

A. Final Implementation

The final implementation of our TPU is quite simple. It has three memory modules, one each for input, weight, and output matrices, and can perform a single 16 x 16 multiply at a time. Surprisingly, this simple implementation was enough to see noticeable performance gains when compared to the typical $O(n^3)$ matrix multiply algorithm, even with compiler optimizations!

V. Future Work

There remains a plethora of future work to be done. Our resulting implementation is extremely bare bones, as we had issues controlling more complex features including accumulation (for larger matrix operations) and ReLU activation.

Given the amount of FPGA fabric unused by our design, it would also be beneficial to create a separate convolution data path, allowing the TPU to be useful for convolutional neural networks.

An idea that came to us late in development was to include a small set of registers within each PE, rather than just a single one bit register. This would enable dynamic switching of weight sets in a single cycle, reducing the overhead of loading weights into the array.

References

References

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, “Title of paper if known,” unpublished.

[5] R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.