Perceptron Hardware Accelerator

**ABSTRACT**

This paper describes the implementation procedure for learning based neural network acceleration. Neural network is trained closely to approximate the original code for a particular transformation. We implement this design on a Xilinx Virtex 5 FPGA and contrast the results obtained through the hardware accelerator with the results obtained through software run on the processor. The hardware accelerator shows 1.77x speedup and 1.71x energy savings.

**Categories and Subject Descriptors**

C.1.3 [**Processor Architecture**]: Other Architecture Styles – *neural nets*; B.2.2 **[Arithmetic and Logic Structures]**: Performance Analysis and Design Aids

**General Terms**

Algorithms, Measurement, Performance, Design, Experimentation, Verification.

**Keywords**

Neural network, neural processing unit, multi-layer perceptron, accelerator, computer architecture, approximate computing, FPGA.

# INTRODUCTION

The cessation of Denard scaling has reduced technological strides on performance and energy efficiency, which has motivated the search for seeking solutions through architectural innovation. There has always been a trade-off between efficiency and programmability. Using custom logic such as ASICs for rapidly changing applications is impractical and this has led the researchers to look at programmable accelerators such as the FPGAs. One way to do this would be through approximate computing for applications that can tolerate quality degradation in return for performance and energy efficiency. Many applications such as signal processing, augmented reality, robotics and speech processing, can tolerate inexact values for most of their execution and this trade-off is leveraged for a boost in performance and energy gains. Second way is to do this is to have dedicated logic in form of accelerators where the flexibility is compromised for lesser hardware demand. Fusion of these 2 techniques leads to better improvement in efficiency. Commercial SoCs incorporating large amount of programmable logic for energy efficiency, are beginning to appear on the market and the On-chip FPGAs are utilized to offload work from CPU which in turn would lead to energy efficiency.

This paper exploits the idea of utilizing FPGA to accelerate approximate programs for better performance without the need for implementing the program strictly as per the algorithm. This is achieved by instantiating a robust, flexible and high performance neural network on the FPGA. This enables programmers to run a spectrum of applications by just initializing new weights into the accelerator without the need to reconfigure the hardware design. The fundamental idea is to learn how the original region of the code that is about to be approximated behaves and replace the original code with an efficient model of the learned model.

The hardware accelerator can be configured through compiler’s workflow by training the logical neural network to behave like regions of approximate code. Better efficiency is obtained because once the neural network is trained, the system discontinues executing the original code and instead starts operating the neural network model on the Neural Processing Unit (NPU). The reconfigurable accelerator has an adaptive neural network design which is advantageous in comparison custom logic for each region of code to be accelerated. First, the neural network training framework helps to eliminate the need for the programmer to design the logic. Second, a large spectrum of code can be accelerated with the same circuit thereby avoiding the need to reconfigure the FPGA each time which can be expensive.

This paper aims at presenting a new technique for utilizing hardware neural network accelerator for general purpose computations. It shows that replacing regions of the original code using the trained a neural network is practical and advantageous by experimenting with Sobel filter algorithm and inversek2j algorithm.

\*\*\*\*\* Some thing about benchmarks \*\*\*\*\*\*\*\*\*

The rest of this paper is organized as follows. In Section II, we discuss about the implementation in detail. In Section III, we discuss about the --. In Section IV, we discuss about the --. Finally, conclusions are given in Section V.

# DETAILS OF DESIGN

In this section, we present the architecture of each of the modules that have been implemented in the design.

## Neural processing unit

## CPU architecture

The CPU developed for this project is based on the MIPS32 architecture, in other words, it is a 5-stage pipelined processor, with the following stages: Instruction Fetch, Instruction Decode, Execution, Memory and Write Back. It is a 32-bit architecture but for practical reasons the memory controller operates only with 28 bits for addressing space. For this processor the instruction set has 30 instructions in which 3 of them are NPU related instructions. This system is capable to execute integer and floating point operations and explicitly flush cache with a dedicated instruction.

All data transfer between CPU and NPU is done through two dedicated 26-bit buses, one to send data and another one to receive data. The communication between the CPU and NPU is based on three instructions. First one is ENQC, which enqueues immediate configuration data to NPU, the second is ENQD which enqueues the data from a register to the neural unit. The last instruction, i.e., DEQD, dequeues data from NPU and writes back into a CPU register position.

Processor itself handles only one side of the communication FIFO. Processor keeps track of the FIFO signals to stall in case NPU is not ready to receive or transmit data. From the CPU perspective, the data must be made available on the correct bus and the proper signals should be asserted, such as, memory communication. NPU takes care of where exactly this data should go and from where it should be read.

## Cache controller

D-cache is implemented as a 2 way set associative cache with write-back, write allocate policy. Cache has a size of 64KB. It has been implemented with flush functionality. Cache address is 28 bit wide, the data bus between CPU and cache is 32 bits wide, and the data bus between cache and memory is 256 bits wide.

## Memory controller and Interface

CPU interacts with the off-chip DDR2-SDRAM, data cache and DVI interface through arbiter logic. Processor receives the image data from computer through serial communication and saves them to DDR2-SDRAM. CPU later gets these data from memory unit and depending on the instruction either computes them in execution unit or sends them to NPU for computation. The resultant image is outputted on screen using DVI interface.

## Implementation platform

The project was implemented on Xilinx Virtex 5 XUPV5-LX110T FPGA. Coding was done in Verilog HDL and the design was synthesized using Xilinx ISE 14.7. Power was estimated using Xilinx Xpower Analyser