Tampere University

Forename Surname

Zero to DLA: Building Software Support For Custom Hardware To Run Complex Neural Networks

Faculty of Information Technology and Communication Sciences (ITC)
Master's thesis
December 2024

Abstract

Forename Surname: Zero to DLA: Building Software Support For Custom Hardware To Run Complex Neural Networks
Master's thesis
Tampere University
Master's Degree Programme in Signal Processing and Machine Learning
December 2024

Lorem ipsum

Keywords: DLA, Deep-Learning, SoC, Virtual Prototype.

The originality of this thesis has been checked using the Turnitin Originality Check service.

Contents

1 Introduction

In recent years neural network based application have become more and more prominent in our everyday-life. The large driver for this has been the adoption of efficient accelerators in mobiled device, that have enabled running neural network applications of mobile devices, such as smart phones.

The goal of this project was to build software support for the Deep-Learning Accelerator in the upcoming Headsail SoC from SocHub using a Renode based virtual prototype as the development board.

2 Background

2.1 Deep-learning accelerators

In desktop applications and data center workloads neural networks have been accelerated with GPUs, due to their ability to perform linear-algebra operations like matrix multiplication with high amount of parallellity.

In recent times there has been a growing need to also run neural networks in mobile devices. Traditional GPUs are too power-hungry to efficiently accelerate these networks, so a need for more efficient accelerators for these workloads has been born. Companies such as Apple and Qualcomm now include multiple mobile DLA's in their SoCs to run applications like face recognition on their phones.

2.2 Headsail

Headsail is the third Soc build by the SocHub research group. Headsail has two RISCV CPUs, one 32-bit meant for booting up the system called Sysctrl and one 64-bit on called HPC, meant for running the actual applications. Headsail includes a wide variety of different peripherals, one of which is a custom build the Deep Learning accelerator.

2.2.1 DLA

Headsail's DLA is a MAC array based accelerator, which provides the following operations: Conv2D, Bias, ReLU.

Conv2D is the non-flipped convolution operation or cross-correlation defined as (Goodfellow, Bengio, and Courville 2016)

$$Conv2D(I, K) = (K \star I)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n) \tag{2.1}$$

as opposed to flipped convolution, headsail vp follows the TVM implementation of using non-flipped convolution instead of flipped convolution. Main purpose of convolution in DNNs is feature extection from inputs (I) with different weight kernels (K).

Bias is defined as such

$$y = xA^T + b \tag{2.2}$$

where if xA^T is the non-biased output of a particular channel in layer, b is the

bias applied to the whole channel as a constant value. In DNNs bias is used to determined importance between different channels i.e. extracted features when used in convolutional layers.

Rectified linear unit is a common activation function in DNNs defined as

$$ReLU(x) = \begin{cases} 0, & \text{for} \leq 0 \\ x, & \text{otherwise} \end{cases} \tag{2.3}$$

In DNNs, activation functions are used to determined if a particular input is interesting in terms of decisions making. Main benefit of ReLU in comparision to other activation fuctions, likea Sigmoid is that ReLU is computationally light operation.

During one layer cycle the operations need to be executed in the following order due to the pipelined design of the DLA: Conv2d, Bias, ReLU. This is the most commonly found order in modern neural networks so it suits most use cases.

$$ReLU(Conv2D(I, K) + b) \tag{2.4}$$

Bias and ReLU can be skipped in the case either or both of the aren't needed in the given layer. In this case Conv2D output is used directly and is capped to fit the 8-bit width of the output.

2.3 TVM

TVM is a machine learning compiler framework by Apache. Among other features TVM includes, multiple runtimes, accelerator backends, optimizers, and a machine learning library for building and training models. The variety of features allows for TVM to be used to implement a complete machine learning workflow, or TVM can be used to implement part of the workflow with other tools.

TVM has it's own graph representation for neural networks called Relay IR. Like the traditional graph representation Relay IR represents network layers as nodes in a abstract syntax tree, where the data flow of the networks is shown as the relationship between parent and child nodes, where parent nodes output is the input of the child node.

TVM is able to be extended to support additional hardware accelerators by implementing a custom code generation module for the target hardware. In principle the developer defines external C symbols that provide the operation implementations which TVM then injects into the Relay IR models. During runtime TVM then calls these external symbols instead of the default operations provided by the TVM Relay library.

It's possible to generate Relay IR models from other graph formats with TVM.

For example common formats like Tensorflow, Torch and Onnx models are officially supported by TVM. This allows developers to build and train their models with tools they might prefer over TVM, and use TVM as a compiler/runtime.

During model compilation TVM is able to optimize the graph and allocate accelerateable nodes to suitable acclerators.

2.3.1 TVM on baremetal

TVM also provides a tool to run TVM models on baremetal platforms called microTVM. MicroTVM is only dependant on the C standard library and thus can be used in any baremetal system that has a working C-toolchain.

MicroTVM works by generating platform independent C-source code from Relay IR-models, which can then be integrated with the microTVM c-runtime to produce executable binaries to run the network.

With custom codegeneration it's also possible to define baremetal compatible accelerator nodes, which the TVM runtime is able to assign layers for during the C source code generation.

3 Methodology

3.1 Renode

Renode is a software development framework, which enables developers to use principles of continuous integration when writing hardware dependent code. In essence Renode is a hardware emulator which allows the user to specify exactly which kind of hardware they want to target, down to the implementation of specific peripherals and memory addresses. This streamlines the process of HW/SW integration, since hardware and software can be developed in parallel, which in return reduces the total production time for products.

Renode models a wide variety of different processors and peripherals, but it also expandable with custom components that are either baked directly into the binary (source code extensions in C#) or with dynamicly loaded python peripherals. Python peripherals are more limited when compared to the C# peripherals. This project implements the DLA hardware design as a dynamic python peripheral.

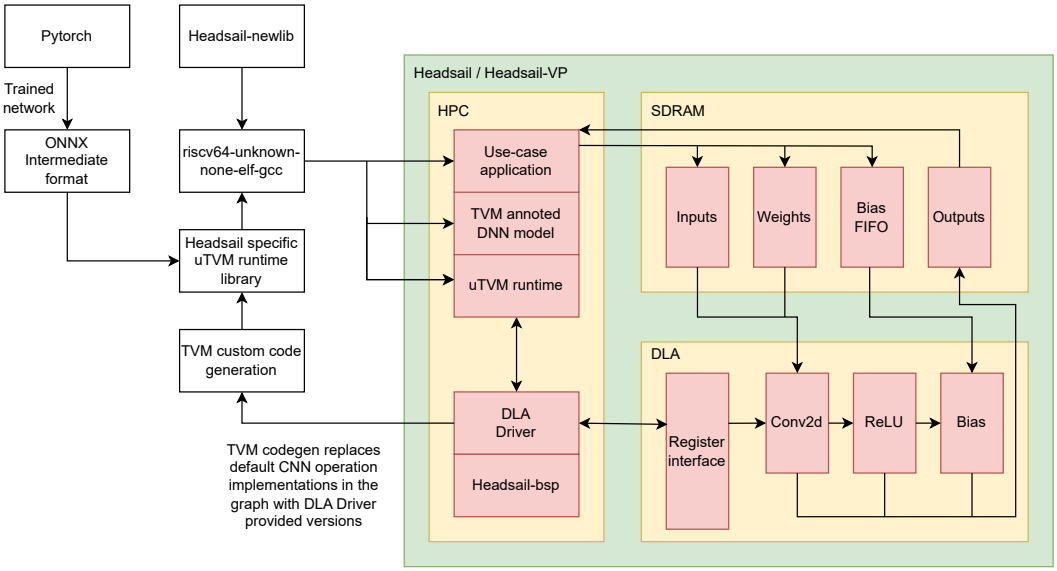

Figure 3.1 *Architecture of accelerated DLA flow in Headsail with TVM runtime and a Pytorch model*

4 Implementation

4.1 Software support

Even though Headsail is the third SoCHub Soc, it had very little existing software support for C. Previous SoCs had only support for Riscv-rt in rust. So a major part of this project involved setting up a Headsail compatible C toolchain. Since Headsail uses includes RISCV CPUs we could use an already existing riscv-gnu-toolchain for the compiler, but we still had to set up a C standard library for the chip with custom version of newlib libgloss. Also due to specific memory addressing decisions in the hardware, we needed to use medany code model compatible compiler and standard library when targetting the 64-bit processor.

5 Conclusions

References

Goodfellow, Ian J., Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. `http://www.deeplearningbook.org`. Cambridge, MA, USA: MIT Press.

APPENDIX A. Something extra

Appendices are purely optional. All appendices must be referred to in the body text

APPENDIX B. Something completely different

You can append to your thesis, for example, lengthy mathematical derivations, an important algorithm in a programming language, input and output listings, an extract of a standard relating to your thesis, a user manual, empirical knowledge produced while preparing the thesis, the results of a survey, lists, pictures, drawings, maps, complex charts (conceptual schema, circuit diagrams, structure charts) and so on.