Internship Report :

Innovative Multiprocessor System-on-Chip architecture for High Performance Packet Processing

Student :

XIE Licheng, Nanotech 2022

Company :

Huawei, 32 Rue Gustave Eiffel, Grenoble

Phelma Tutors :

ANGHEL Lorena, lorena.anghel@grenoble-inp.fr

PREJBEANU Liliana, liliana.buda@cea.fr

Company Supervisors:

MINGUEZ Benjamin, benjamin.minguez@huawei.com

JALIER Camille, camille.jalier@huawei.com

Confidential : Yes

Date : From 14 February 2022 to 12 August 2022

Abstract

In the era of information, network plays a critical role in the domain of communication. It provides a way to link the world. Especially for Internet of Things (IoT), devices are connected together within the network. High performance packet processing allows to deal with network, classifying the packet in order to prioritize the traffic.

The breakthrough of Machine learning (ML) gives rise to the possibility for classification of packet. However, high performance inference is required to achieve real-time processing, as well as high accuracy. Inside the network equipment, the CPU architecture is supposed to be well designed to meet the requirement.

Huawei Datacom team aims to propose the design of the next generation network System on Chip (SoC) of ARM CPU, optimizing the inference performance with the already-trained Convolutional Neural Network (CNN) model : ShuffleNetV2. The performance is measured by linux kernel perf tool or embedded benchmark. Besides, the firewall is necessary functionality of network equipment for filtering packets.

À l'ère de l'information, le réseau joue un rôle essentiel dans le domaine de communication. Il fournit un moyen de relier le monde. En particulier pour Internet of Things (l'IoT), les appareils sont connectés ensemble au sein du réseau. Le traitement de paquets haute performance permet de traiter le réseau, en classant les paquets afin de donner la priorité au trafic.

La percée du Machine learning (ML) donne lieu à la possibilité de classification des paquets. Cependant, une inférence de haute performance est nécessaire pour réaliser un traitement en temps réel, ainsi qu'une haute précision. Dans les équipements de réseau, l'architecture de l'unité centrale de traitement est censée être bien conçue pour répondre à cette exigence.

L'équipe de Huawei Datacom a pour objectif de proposer la conception de la prochaine génération de System on Chip (SoC) réseau de CPU ARM, en optimisant les performances d'inférence avec le modèle de réseau neuronal convolutionnel (CNN) déjà entraîné : ShuffleNetV2. La performance est mesurée par l'outil de perf du noyau linux ou par un benchmark embarqué. Par ailleurs, le pare-feu est une fonctionnalité nécessaire de l'équipement réseau pour filtrer les paquets.

Nell'era dell'informazione, la rete svolge un ruolo fondamentale nell'ambito della comunicazione. Fornisce un modo per collegare il mondo. Soprattutto per Internet of Things (l'IoT), i dispositivi sono collegati tra loro all'interno della rete. L'elaborazione dei pacchetti ad alte prestazioni consente di gestire la rete, classificando i pacchetti per dare priorità al traffico..

La scoperta del ML (Machine learning) offre la possibilità di classificare i pacchetti. Tuttavia, per ottenere un'elaborazione in tempo reale e un'elevata accuratezza, sono necessarie elevate prestazioni di inferenza. All'interno delle apparecchiature di rete, l'architettura della CPU deve essere ben progettata per soddisfare i requisiti.

Il team di Huawei Datacom intende proporre la progettazione di un System on Chip (SoC) di rete di prossima generazione con CPU ARM, ottimizzando le prestazioni di inferenza con il modello di rete neurale convoluzionale (CNN) già addestrato: ShuffleNetV2. Le prestazioni sono misurate con lo strumento linux kernel perf o con un benchmark embedded. Inoltre, il firewall è una funzionalità necessaria delle apparecchiature di rete per filtrare i pacchetti.

Contents

1 Introduction **1**

1.1 Internship Overview . . . 1

1.2 Company Description . . . 1

2 Technique Context **2**

2.1 Network . . . 2

2.1.1 Network Equipment . . . 3

2.1.2 Network Layer Protocol . . . 4

2.1.3 IP Packet . . . 4

2.2 Firewall . . . 5

2.2.1 L3 Forwarding . . . 6

2.2.2 Application Classification . . . 7

2.3 Machine Learning . . . 8

2.3.1 Convolutional Neural Network . . . 8

2.3.2 ML Framework . . . 9

2.3.3 ShuffleNetV2 . . . 11

2.3.4 Accelerated Linear Algebra (XLA) . . . 12

2.3.5 Quantization . . . 12

2.4 Hardware . . . 14

2.4.1 CPU . . . 14

2.4.2 Hardware Accelerator . . . 15

2.5 Software . . . 16

2.5.1 Operating System . . . 16

2.5.2 Simulator . . . 17

2.5.3 Perf Profiling . . . 17

3 Implementation **19**

3.1 L3 Forwarding . . . 19

3.1.1 Flow Table . . . 19

3.1.2 Packet Length Identification . . . 20

3.2 Performance with TensorflowLite . . . 22

3.3 Performance with MindsporeLite . . . 23

3.3.1 Benchmark Timeprofiling . . . 23

3.3.2 Benchmark Perf Profiling . . . 24

3.3.3 Quantizatoin performance 25
3.4 Hardware Accelerator Performance 26

4 Conclusion **29**

1 Introduction

1.1 Internship Overview

This report is a summary of the 6-months internship, which is a part of academic schedule of master degree in INP Phelma. As the final stage of master, this internship allows me to summarize the knowledge I have learnt and to better adapt the future carrier. In this summary, I will first introduce the technique context regarding the project, then the implementation as well.

During this internship, I have been involved in advanced RD topics in Hardware architecture targeting next-generation ARM/RISC-V based SoC for network equipment (firewall, access router etc). The working domain is computer network, processor architecture and data communication, AI is included as well. The working platform is linux OS, a server rent by Huawei, and programming languages python and C/C++ are required in addition.

1.2 Company Description

The company Huawei is a fast growing, leading global Information and Communications Technology (ICT) solutions provider, aiming to enrich life through communication. It also offers network infrastructure, cloud computing solutions and devices such as smart phones and tablet PCs with 3 business units : Carrier, Enterprise and Consumer.

The working location, as known as Amdahl Lab, Grenoble Research Center performs strategic research and cutting edge development for Huawei. The team I work with is team Datacom. Camille is the leader of team, while benjamin is my tutor in company.

2 Technique Context

As we step into 21st century, Internet becomes indispensable in our daily life. People are getting closer and closer to each other. Especially thanks to the emerging of 5G technology, the productivity has been significantly increased. As a result of explosion of information, the amount of data has been growing exponentially, leading to a higher and higher demand of data processing capability.

Data network gives a way to interconnect all the devices within the network, allowing the users to communicate with each other. As the most essential part for network, the router, it provides forwarding service for the computers, in which a firewall can be implemented in order to secure the data communication. Under this background, the purpose of Huawei Datacom team is to propose a hardware and software solution for high performance network equipment, such as Router, Gateway and Firewall.

2.1 Network

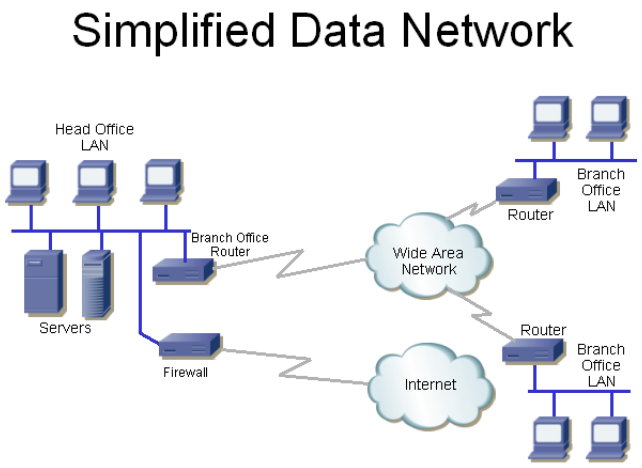

Figure 1: Simplified Data Network

[3]

As demonstrated in fig.1, this diagram shows the simplified abstraction of data network. The router provides a connection between LAN and WAN. The router consists in a basic architecture of data network. A example is company's internal network. The company's information is centrally stored in servers as a form of database. The clients, employees are able to either access the

data remotely through the Wide Area Network (WAN), or be connected through the Local Area Network (LAN). The access between LAN and public Internet is controlled by a firewall.

The basic way of data communication is the transmission of traffic flow. During a communication service, the traffic flow carrying data is exchanged between devices in network thanks to an Internet protocol. It is sent and received as a sequence of packet which is the minimum communication unity in network. The data, as known as payload is the content of traffic flow, containing the information to be transmitted. A typical application is Web such as google. At one moment, there are numerous requests arrive. In order to deal with the extreme case, a router is required to be very high performant, as well as a efficient algorithm. In terms of cybersecurity, firewall is also needed to prevent the network from being attacked, by filtering the packet that is not permitted to be forwarded, since the private information is getting more and more concerned and need to be protected. Moreover, due to the limited bandwidth of broadband, multiple applications that are running in parallel need to be prioritized to balance the resource usage. For instance, the traffic flow of stream media should have the higher priority than the real-time chat application to guarantee that video is played fluently.

2.1.1 Network Equipment

Figure 2: Huawei Unlocked E8372 4G/LTE Wi-Fi Dongle Router

[8]

A router is a device located between the LAN and WAN, aiming to interconnect both sides. It is basically a computer, but is customized to adapt the network packet processing and charged for receiving and forwarding the data flow at network layer. If we take the data packet as air plane, the router is thus the airport. The fig.2 shows the image of the router of Huawei. The router being a computer is supposed to have 5 components in general : Input, Output, Memory, Control and Datapath, while Control and Datapath are usually considered as processor. In the case of data communication, the data of traffic flow is fed in Memory by Input, then is read by register of

CPU, and manipulated. Finally the result of computation is stored in Memory to be used later or directly to Output.

2.1.2 Network Layer Protocol

The network layer at which router routes and forwards data packet is responsible for data transmission among devices. It also provides logical address for network devices according to common network layer protocol such as IPv4 and IPv6. At this layer, the packet whose header (see fig.4) carries the network layer address of the source and destination devices is sent from source host to destination host. A unique IP address is assigned to each communication device in order to identify themselves when IP is used as the network layer protocol. Once the packet is received and read by the router, the destination address in the header of packet helps to search the routing table. The forwarding of packet depends on the matching entry of the destination address in the routing table.

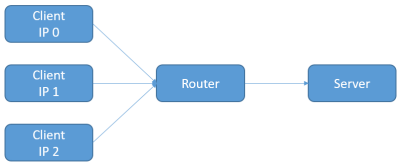

Figure 3: A basic Client-Router-Server model

The fig.3 shows a basic Client-Router-Server model in which the 3 clients communicate with the server through a router. An IP address stands for an identification of a node or an interface of a device in the network and is used for finding the destination of traffic flow. In order to connect a computer in Internet, the user must apply for an IP address for the computer. In addition, an IP address is assigned to an interface on the device, instead of the device itself.

2.1.3 IP Packet

As mentioned before, the data communication between client and server (host) is realized by transferring traffic flow. Due to the limited size of data transmission TCP/IP protocol, the traffic flow is thus splitted into multiple small IP packets which contains header and data (payload). The data payload is the information content to be conveyed, while the header segment includes some fields of identification such as IP address of source or destination. In general, the traffic flow is delivered as a sequence of packet, where the header will be added ahead of data payload

during packet assembly (encapsulating) when switching layer according to the TCP/IP protocol to facilitate the data content transmission, and data is usually a combination of letters or numbers.

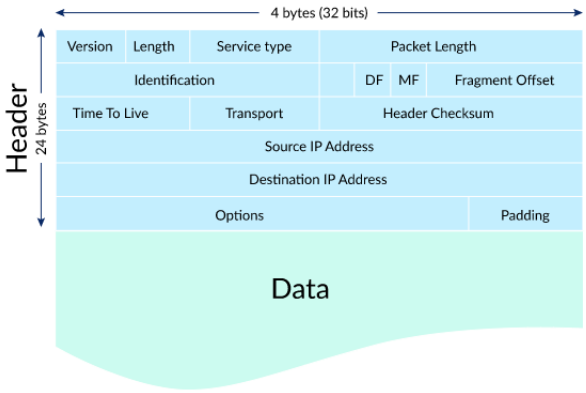

Figure 4: IP packet consisting of header and data (payload).

[5]

2.2 Firewall

The interaction within Internet is becoming necessary for personal and organization thanks to the network equipment. The inward and outward access give benefits to the Internet connection, but also bring invisible thread in terms of cybersecurity. For the purpose of internal network security, the firewall is required in order to control and monitor the traffic flow. The firewall which consists of software and hardware interface is the first front protection for cybersecurity. It is located at the edge of company group to limit the external user from accessing internal Internet, and to manage the authorisation for internal user to visit external Internet as well. Based on the defined rule, the traffic flow, as a sequence of packet will be dropped or passed. The firewall functions as a node to separate trusted and untrusted Internet by filtering the unsafe data. The firewall can also be an network equipment containing a SoC on which software interface is implemented.

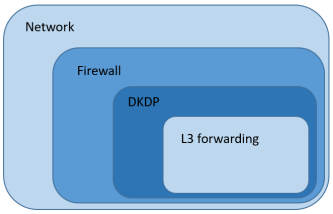

Figure 5: Block Diagram of Firewall

There are several types of firewall in terms of its way of functioning, where the packet-filtering

firewall is the one we mainly work on. As shown in fig.5, L3 forwarding is the principal application of firewall, and is implemented by Data Plane Development Kit (DPDK). It provides a security check on received packets without knowing if the packet is a part of streaming of traffic flow. The DPDK framework provides data plane libraries and network interface for the operating system kernel to run the packet processing. Given that the header of each IP packet contains the filed such as IP address of source and destination, the firewall can manipulate the packet according to the firewall rules. As shown in fig.6, the packets having a specific IP address or port number are blocked or passed. In our project, the firewall has 2 main functions to be researched:

1. L3 Forwarding : Decide if the packet is going to be dropped or forwarded.
2. Application Classification : By identifying the application label, prioritize the traffic flow.

	Source IP	Dest. IP	Source Port	Dest. Port	Action
1	192.168.21.0	--	--	--	deny
2	--	--	--	23	deny
3	--	192.168.21.3	--	--	deny
4	--	192.168.21.0	--	>1023	Allow

Sample Packet Filter Firewall Rule

Figure 6: Firewall Rule example, according to which the firewall decides to drop or forward the incoming packets.

[9]

2.2.1 L3 Forwarding

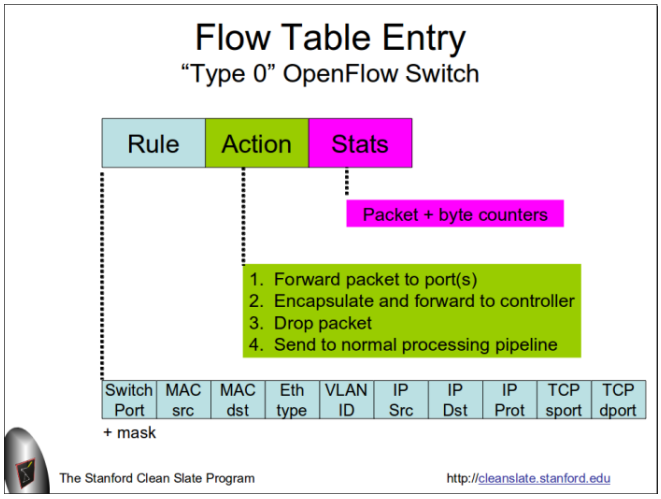

Figure 7: Flow Table Entry

[10]

Being the fundamental function of firewall, L3 forwarding is based on a particular data structure called flow table shown in fig.20. The hash lookup key is represented by a 5-tuple with the fields in the header of each IP packet. we take the IP address and port of source and destination respectively, as well as the protocol for the identification filed. Then a 32-bit hash key calculated the identification field is an efficient way to index the flow table to prevent doing a search in the table one by one. Thanks to this technique, a flow table can contain several millions of flows. There are 3 modules for this application : initialization of flow table, fast path (Fpath) and slow path (Spath).

2.2.2 Application Classification

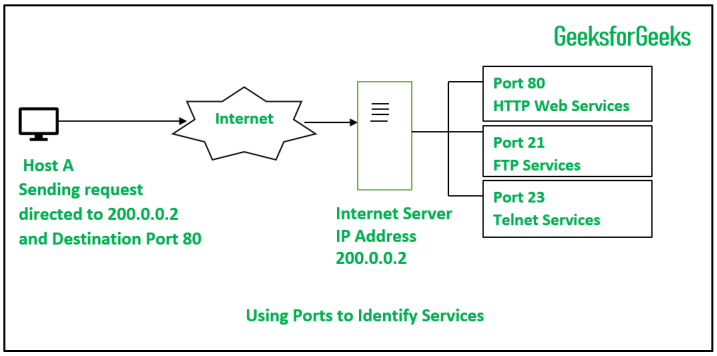

Figure 8: Application classification using IP address and port

[1]

The other main functionality called application identification is as important as the flow table for firewall. Since there may be several applications running at one moment in one device such as VOIP, streaming audio and video. For the same source and destination, the IP address stays the same. And the port number depends on the application. That is to say when watching video on youtube, the packets of stream traffic flow have the same IP address and port number. This length list showing some features of such label could be used for predicting by applying Machine Learning as demonstrated in fig.8. By doing a real-time inference for the length of packet payload, the applications are able to be classified and prioritized in order to optimized the performance.

2.3 Machine Learning

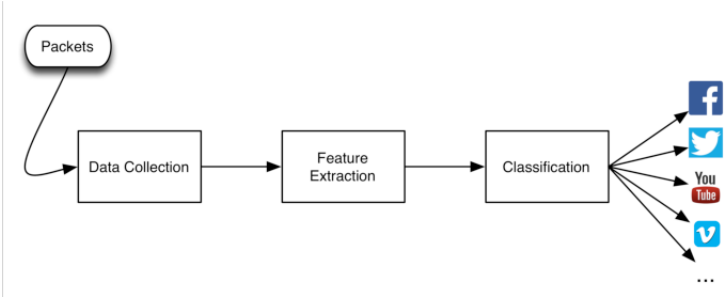

Figure 9: Packet Feature Extraction and Appliation Classification

[4]

In the domain of ML, Convolutional Neural Network (CNN) is often dedicated to Image Processing. As shown in fig.9, the CNN allows to extract the key features of the information in mathematic form, predicting the corresponding label. With enough training, the weight of such feature will be adjusted to adapt the input fed. This principle is to imitate the way human are learning. By applying ML, we can quickly know the application of packet and optimized the priority. The process of predicting the application is called inference.

2.3.1 Convolutional Neural Network

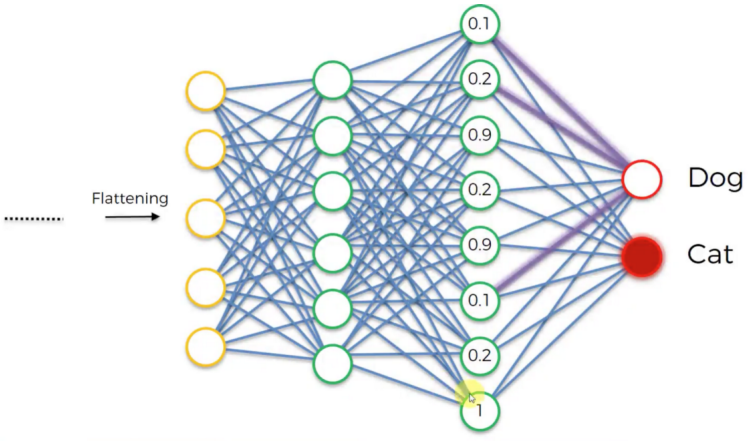

Figure 10: A part of Neural Network

[12]

The CNN is algorithm model which deals with the information with imitation of biological neural system. It can be used to extract the representation of input's content by adjusting the weight between the internal neurons. Consisting of certain convolutional layers, pooling layers and full connection layers, the CNN allows the tensors to propagate through layers, so that the weight of

each neuron node could be adaptive for the pre-defined label. The fig.10 shows a section of neural network example, where each node with a value of weight in its layer stands for a specific feature. After spreading through layers, the weight distribution of nodes gives rise to the prediction for the label "Dog".

As for our project, as shown in fig.11, a CNN is applied for application inference. The neural network receives a set of input data, then outputs a list of probability of each label. The maximum value of probability represents the most possible application predicted. Unlike image recognition, the input in our case is a vector instead of matrix, indicating that the network would not be large or complicated. In our case, per-packet information (the first N packets' payload length and direction) makes up a list to feed the input of CNN and the a distribution containing number of labels is output. Each number among the distribution represents the probability of the input flow belonging to a specific application. Thus the highest probability is chosen as the final predicted output label.

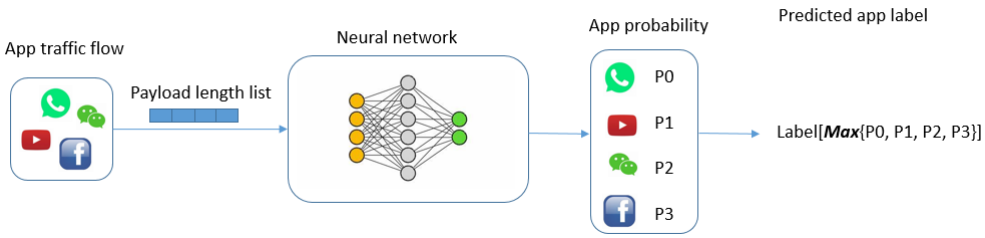

Figure 11: Application inference with neural network. A set of payload length list with app label are fed into neural network, outputting the most probable app label.

[6]

2.3.2 ML Framework

A framework is simply a software tool to be implemented for AI. For a CNN inference to be executed in a router, we need to first convert the trained network in an optimized executable, this is done offline by the AI framework. Being the most common framework for ML, Tensorflow is a flexible and high-performance serving system for machine learning models, designed for production environments. It provides a bunch of useful python interface with which we can easily do the training or inference. While keeping the same APIs, the other types of models can also be served with due to the easy extension of Tensorflow integration.

The Tensorflow is intrinsically a process of definition of graph, establishment of graph and execution of graph. Based on Data Flow Graph (DFG), Tensorflow allows to describe mathematic computation with nodes and edges. As shown in fig.12, the node represents the mathematic manipulation, while the edge, as known as tensor, stands for the multi-dimension data array which interconnects the nodes. In Tensorflow, a computation graph is at first constructed. According to the graph, it will create a session, in which the variable assignment and calculation will be done, then finally output the result.

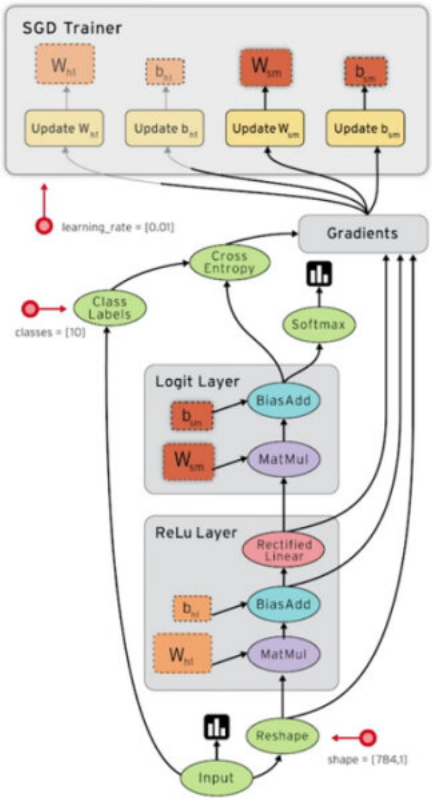

Figure 12: Data Flow Graph of Tensorflow

[11]

At the beginning of internship, a light Tensorflow model running on python is already trained by another team in Huawei. Considering our network equipment, a more optimized approach is to use TensorflowLite to generate a model to be integrated in a C/C++ application, instead of using the python interpreter. This is a light version of Tensorflow dedicated for edge deployment such as mobile device or embedded system. As well as Tensorflow, Mindspore is equally a ML framework developed by Huawei, but is more compatible on Huawei's network equipment. Its Lite version MindsporeLite is also available for edge AI deployment.

2.3.3 ShuffleNetV2

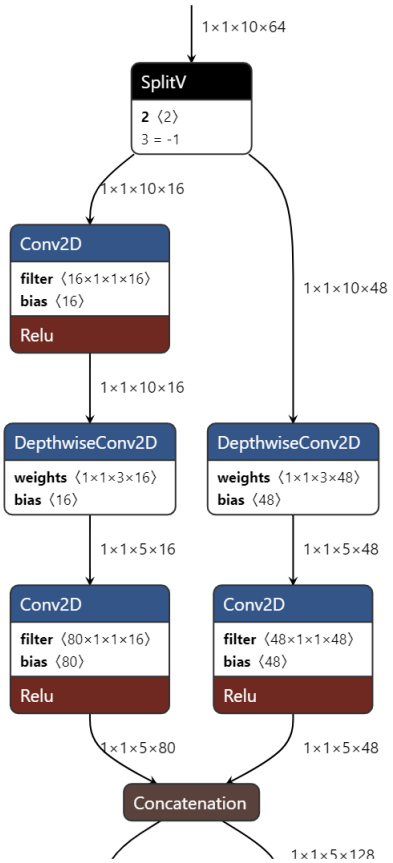

Figure 13: A part of ShufflenetV2 TensorflowLite model for our project

Despite the type of framework, the ML model is simply performed as a graph that contains a set of operators to be manipulated.The inference consists in going through and executing each layer in model sequentially. For instance, a 4-dimension input matrix is fed into the graph. By executing the operators, the graph finally outputs a vector representing the probability of labels. Fig.13 shows a demonstration of a part of our already trained ShuffleNetV2 model. The input is splitted into 2 paths along the its dimension, including Con2D and DepthwiseConv2D respectively. The layer Con2D contains a activation function Relu whereas DepthwiseConv2D does not. These 2 paths make up an independent convolution subpath to reduce the convolutional calculation cost, since this so-called group convolution helps to decrease the complexity of model, so as to reduce the parameter number. Moreover, this method gives a high degree of parallelism which can speed up the inference. However, another factor which influence the speed is the Memory Access Cost (MAC). Too much grouping leads to augmentation of MAC, thus slowing down the model.

2.3.4 Accelerated Linear Algebra (XLA)

Before the emergence of XLA, the execution of graph of Tensorflow is driven by runtime kernel which is responsible for graph definition, graph construction, graph optimization etc, which leads to time overhead in going though each node in graph and large size of binary file, since not only the graph, the runtime code is also needed. There 2 methods provided by XLA, Ahead Of Time (AOT) and Just In Time (JIT). Now thanks to XLA, another choice is that the graph is able to be compiled into executable binary which can be executed directly without runtime. This technique allows to save a lot of time since there is less overhead of running time in reading the graph, and reduce the binary file size.

2.3.5 Quantization

The quantization is an important technique allowing to speed up the inference without losing too much accuracy. As the development of ML, a large number of parameters have been brought to improve the model performance. Our ML model is trained with 32-bit float weight, which leads to high amount of calculation, as well as the number of parameters. Also, a large size of model is inconvenient to be deployed for edge AI despite the high accuracy. The weight is usually quantized, as well as activation output.

Performance	Parameters size	Calculation amount	Memory Usage	Inference Accuracy
Quantization	Large	Large	Huge	High
No quantization	Less	Less	Small	Low

Table 1: Performance Impact of quantization

The advantage of quantization is shown in table.1. Less parameters size has a direct impact for inference performance, since the calculation amount is decreased. Besides, less memory occupation can also speed up the inference. On the one hand, the bottleneck of inference depends on read and write most of the time, instead of calculation. The augmentation of calculation density has a huge optimization on performance. On the other hand, smaller model size results in less memory occupation, facilitating the software upgrading. With the help of quantization, the power consuming can also be reduced, since the power consuming is becoming the main figure of merit for edge device. What is more, the latency for floating point calculation is generally longer than integer for CPU.

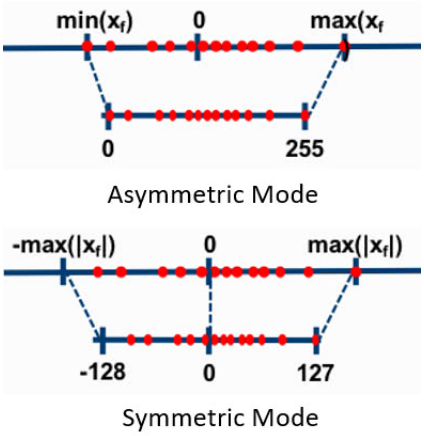

Figure 14: 2 modes for quantization

[7]

There are 2 quantization modes to convert 32-bit to 8-bit, asymmetric mode and symmetric mode. as shown in fig.14. In fact, the quantization is a process to convert the weight or activation output of trained model from high precision to low precision. A data reflection between fixed point and floating point is built. A continuous value is assigned to a discrete value, while losing more or less precision. The equation below shows how to construction the relation, where the Q is the quantized value, and the R is the value to be quantized. The zero point is the value of zero after quantized, it may not be at the middle value depending on the quantization mode. The scale factor demonstrates the proportion between floating point and integer. For instance, in asymmetric mode, $scalefactor = (255 - 0)/(max - min)$

$$Q = scalefactor * R + zeropoint \tag{1}$$

The Post Training Quantization (PTQ) and Quantization Aware Training (QAT) are 2 main ways of quantization. In my case, the PTQ is used to convert the 32-bit floating trained model into low bit quantized model. The 3 types of PTQ is shown in table.2.

Quantization Type	Description	Model Size	Accuracy loss
Dynamic range quantization	Weights are quantized to 8-bit for saving, and dequantized in 32-bit for inference	Reduced to 25%	Modest loss
Full integer quantization	Weights, input and activation output are all quantized from floating point to int8 for saving and inference	Reduced to 25%	Less loss
Float 16 quantization	Weights are quantized to 16-bit for saving, dequantized to 32-bit for inference	Reduced to 50%	Modest loss

Table 2: 3 types of Post Training Quantization and their performance

2.4 Hardware

Our project is to propose a solution for router, which is basically a computer. And the core of computer is the processor, as known as CPU. The CPU is integrated in SoC along with other peripherals interfaces to make up a computer. A CPU is supposed to dispose 3 fundamental components, the arithmetic–logic unit (ALU), processor registers and a control unit. The CPU can be classed into 2 types in terms of Instruction Set Architecture (ISA), "CISC" for intel x86 and "RISC" for ARM. As for our project, the ARM based CPU in the router having high performance is then required to be able to do data processing, such as inference, and other jobs. Hardware accelerator is also possible tobe applied depending on its performance improvement.

2.4.1 CPU

x86 is a family of ISA initially developed by Intel, dominated the PC era and dominates cloud computing. Started from intel 8086 microprocessor, a 16-bit processor, x86 family has evolved to 32-bit and then to 64-bit instruction set. Thanks to its nice compatibility, processor of x86 family has extended to any type of computer from desktop computer to laptop, server, supercomputer and notebook. In my internship, a x86 ISA server is provided as a platform to run some feasibility tests.

Unlike x86, ARM based processor is very suitable for mobile telecommunication domain due to its energy saving feature. Its multicore heteostructure allows to improve the effectiveness of Out-of-Order pipeline. With less instruction dependency, a high throughput of instruction can be

realized for high frequency. Our project is to research the performance of an ARM based processor.

2.4.2 Hardware Accelerator

In actual fact, most time passed for inference is on convolution calculation, meaning that the performance of inference depends mostly on the performance on the matrix calculation. Especially for large CNN network, the performance bottleneck is relevant to the convolution percentage. Given that the our network equipment is not a large device and its CPU size is also limited, a hardware accelerator may be proposed in order to improve the matrix calculation so as to have a better inference performance.

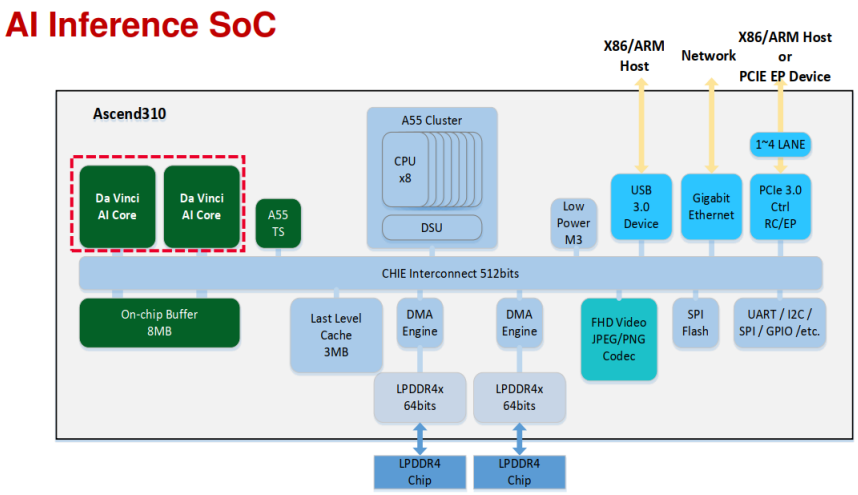

Figure 15: AI Inference SoC of Huawei Ascend310

[13]

The fig.15 shows the AI inference SoC architecture of Huawei Ascend310 where there are 2 AI Core embedded. The hardware accelerator called DaVinci AI core is generally a device hosted by CPU through a interconnect bus, dedicating for matrix calculation. The AI model is at first deployed on the accelerator. The host CPU then rearranges the input data in memory and sends the data to AI core via data bus. Once the AI core finished the calculating, the result will be sent back to CPU and ready for the next manipulation. However due to the transmission quality of Peripheral Component Interconnect Express (PCIE), the additional performance of hardware accelerator will be influenced depending on the weight of convolution calculation.

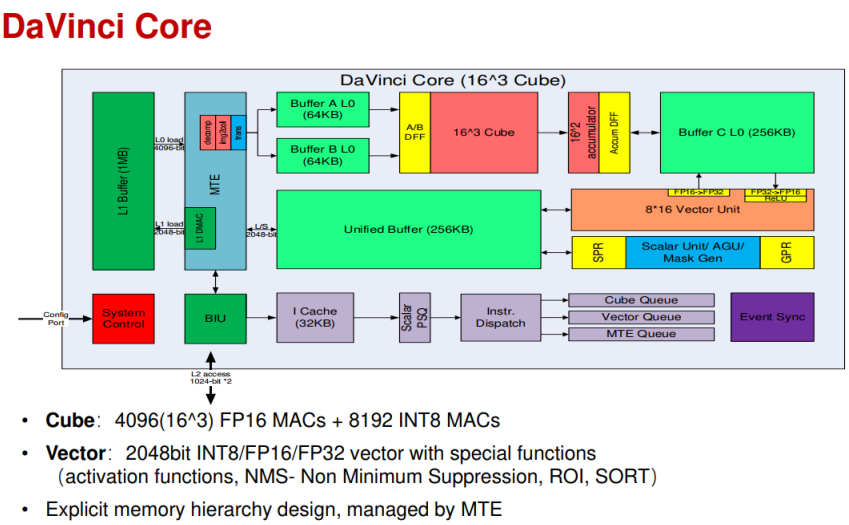

Figure 16: Illustration of DaVinci AI core

[13]

The fig.16 shows the architecture of DaVinci AI core. Thanks to the Cube unit, Vector unit and Scalar unit, it can deal with different types of data flexibly. Since matrix calculation is the main part of AI computation, the Cube unit allows to get the result much faster scalar calculation. For instance, a 16*16*16 cube matrix has 4096 operation. Thanks to its customized structure, the whole calculation can be finished within 1 instruction cycle. The drawback is that it can only do cube manipulation. This feature can speed up significantly the matrix calculation. Secondly, the instruction of Vector unit is dedicated to other vector manipulation other than cube matrix. And the Scalar unit can be functionally considered as a micro-CPU, it aims at data control, branch prediction and also the calculation of instruction address of Cube and Vector.

2.5 Software

2.5.1 Operating System

During this internship, I worked mainly on linux shell. All the data is stored in Huawei's server in order to secure the information. Unlike windows, a visual Operating System (OS), a skill of Command-Line Interface (CLI) is strongly required for linux OS. As shown in fig.17, it provides a interaction interface between human and machine. Being a command interpreter, shell is basically a User Interface (UI). It allows to receive the instruction and send it to linux kernel to execute. Except the basic command line operation, there are 2 other software tools are used, CPU simulators and linux perf tool.

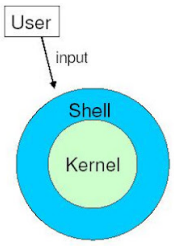

Figure 17: Illustration for bash shell

[2]

2.5.2 Simulator

In terms of R&D, especially for CPU architecture design, the simulation is needed before the manufacturing of real product. In this case, a simulator is utilized to evaluate the CPU performance and to optimize the SoC architecture to deliver the highest throughput. The parameters of simulator could be modified to adapt the configuration of the simulated CPU. Once simulation is finished, we get a trace file recording all the processes of program execution in CPU. With the help of the trace file generated by simulator, a call graph showing function calling hierarchy can be established. It would be much more convenient to analyse thanks to the call graph. We are interested in the performance of doing inference with Tensorflow Lite. In order to analyse the performance, it is necessary to know what the instructions are going on inside the CPU. As shown in fig.18, the behavior of CPU such as micro-instructions are recorded after the simulation.

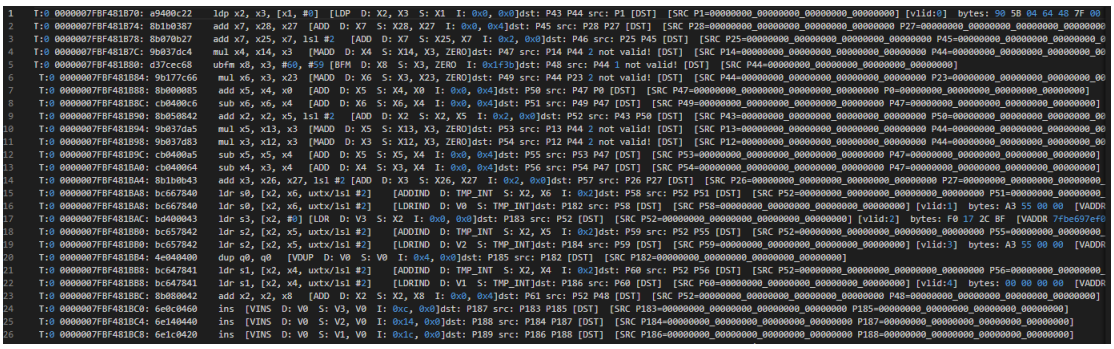

Figure 18: A section of core trace log for simulation of inference

2.5.3 Perf Profiling

Perf Profiling is a profile tool embedded in linux kernel to measure the performance of a program. Based on "*perf_event*", it allows to mark tracepoints with a certain frequency when executing a program. The perf tool supports a set of special measurable event from different sources. Soem events are purely kernel counter, while the others come from processor itself and it Perfor-

mance Monitoring Unit (PMU) which allows to measure the micro-architecture event such as cycle number, L1 cache miss or failed instruction. The *perf_event*" also provides a group of generic hardware event. In each processor, these events are mapped to a real happened CPU event. The "*perf_event*" is based on event sampling. It records the times of event happening during a period, instead of counter's counting. By default, the clock period event is used as the sampling event, which is a generic event mapped to particular PMU event by kernel. By collecting and analysing these events, a statistic report shows the time distribution of each function. The sub-functions help us to analyse the program, finding the bottleneck thus improving the performance.

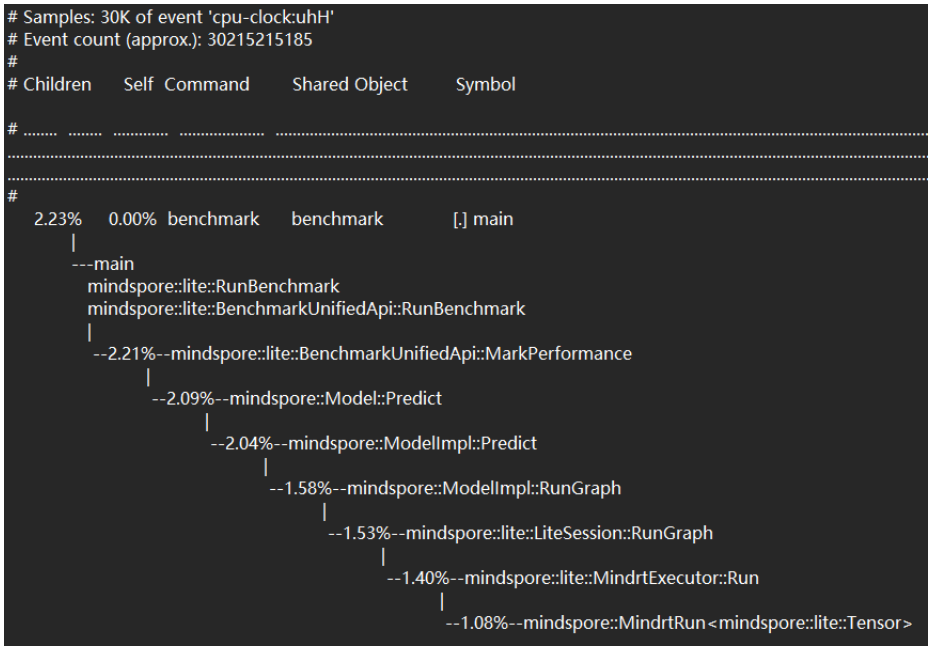

Figure 19: A demonstration of callgraph for inference benchmark

Another advantage of perf tool is that by profiling a program, it also generates a callgraph which introduce how functions are called. As shown in fig.19, this callgraph represents a tree of called functions and their hierarchy. The percentage of each functions shows the time distribution so we can easily know where the resources are used. For instance, in main function, a *RunBenchmark* is called, then followed a $MarkPerformance$. Next after *Predict*, *RunGraph* is called to construct the graph to go through. And finally the function *Run* is the core of this whole program. With this tool, it is much easier to locate the function wanted, and also the bottleneck that is critical for performance.

3 Implementation

3.1 L3 Forwarding

Based on L3 forwarding, the firewall allows to identify the received packet and decide to drop or forward it. With 5-tuple in the header of each packet, a particular data structure called flow table is adapted for our project to optimize the firewall. In addition, another technique called packet length identification using ML is utilized to predict the belonging application of packet, thus to prioritize the traffic flow. Thanks to this technique, the traffic flow of various applications will be equilibrated without being blocked, making the Internet surfing smoother and more fluent.

3.1.1 Flow Table

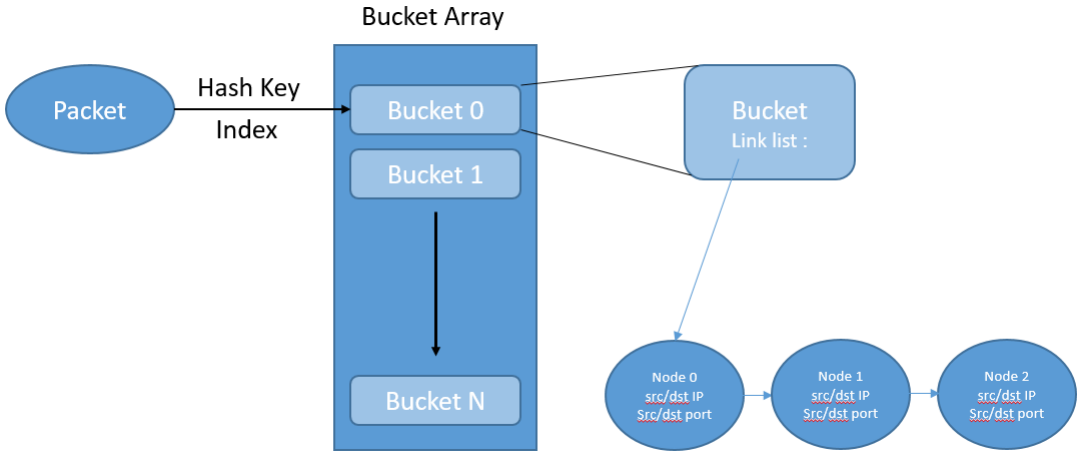

Figure 20: Flow Table for L3 Forwarding

The fig.20 shows the general architecture of flow table, in which 2 basic data structures are implemented, bucket and link list. The buckets are stored and arranged sequentially in the bucket array that is allocated depending on the size of index hash key. For instance, a 22-bit hash key can represent $2^{22} = 4M$ buckets. In each bucket, there is a link list allowing to store the 5-tuple of packet as a form of node.

- Initialization of flow table : A certain size of memory space necessitates being allocated initially to store the bucket. According to its hash index calculated by the 5-tuple identification field, a new packet which satisfies the rule will be added as a node containing the 5-tuple in the link list of the corresponding bucket in the bucket array. Otherwise it will be ignored. The new come node will be added to the head of such link list.

- Fast path : When receiving a new packet, its 5-tuple will be transformed to a particular hash key which allows to index the corresponding bucket. Then in the link list of the bucket, we go through each node in order to check the existence of such 5-tuple. If a packet exists already in the link list, it is to be said to be matched, and will be forwarded directly. This methods speed up so much the forwarding process, so it is called fast path.

- Slow path : On the contrary, if the new packet does not match in the link list, then we check the rule. If the rule (ex.fig6) satisfied, the link list is going to be renewed by adding the new node at the head of link list. Otherwise the packet will not be forwarded nor added in the flow table.

- Life span : Another critical feature of flow table as important as fpath and spath is the mechanism called up-to-date. A field called time-to-live (ttl) is stored in the node to mark the time difference between the absolute real time and the moment that the node is created. This ttl represents how long the node exists. By setting a time threshold, the "old" nodes that stay too long in the link list will be removed, since the flow table needs to keep fresh to adapt and react to the real-time environmental variation.

In the above way, the firewall allows to quickly check possibility of whether forwarding the packet or not. Depending on the number of flows, the size of bucket array is flexible to be adjusted. Thanks to the index hash key calculated by 5-tuple identification fields, it provides a fast way to index in the flow table without checking all the information. The spath allows the link list to be renewed according to the firewall rule. Also, the link list can be updated by removing the expired node to keep the flow table always in date. These functions consist in the basic conception of flow table in L3 forwarding.

3.1.2 Packet Length Identification

At the network layer, the firewall can decide to drop or pass the packet thanks to the flow table thus to optimize the forwarding. However, the couples consisting of one client and one application are numerous. In order to optimize the priority of application by classifying them, the technique called packet length identification is thus proposed. In terms of application classification, as mentioned before, the stream traffic flow is forwarded as a sequence of packets. Among a set of packet, the lengths of their payload make up a list called timeserie. This timeserie list performs some features of such application, and could be used for application prediction as shown in fig.9. In this way, we can use ML to extract the feature of each set of timeserie, thus to predict the

corresponding application.

app_nam	file_nam	train	length	duration	byte_up	byte_dow	time_up	time_dow	val_000	val_001	val_002	val_003	val_004	val_005	val_006	val_007	val_008	val_009	val_010	val_011	v
YY_Live	home_01	0	248	2.533008	44179	78	2.506077	0.026931	48	-30	860	-36	-36	31	47	193	218	193	214	203	
YY_Live	home_01	0	6391	596.0051	13598	1214033	297.0519	298.9533	30	-30	323	-39	45	-41	34	38	-202	-213	-212	-204	
YY_Live	home_01	0	635	60.9678	2170	118036	30.88348	30.08432	30	-30	323	-39	45	-41	34	38	-208	-206	-210	-208	
YY_Live	home_01	0	299	9.475469	46811	862	5.117453	4.358016	48	-30	860	-36	-36	31	47	193	206	193	217	205	
YY_Live	home_01	0	1433	125.6999	2655	290753	62.87963	62.82024	30	-30	331	-39	45	-41	34	38	-222	-226	-232	-235	

Figure 21: A demonstration of a set of timeseries of packet

The fig.21 shows a example of 5 groups of timeseries of the packets of communication between the client and an application. The first column *app_name* is the application label that we need to predict. The columns *file_name*, *length*, *duration*, *byte_up*, *byte_down*, *time_up* and *time_down* are all flow's aggregate metrics which we don't use. The column *train* is an indicator: if it equals to 1, then the flow should be used for training and validation, otherwise flow should be used for testing. Finally the columns *val_000* *val_099* are the flow's first 100 IP packets' size and direction where positive and negative sign stand for received and forwarded between the client and host respectively. Since the packet forwarding takes place in the network layer, the Maximum Transmission Unit (MTU) is 1480 Bytes while the header takes 20 Bytes. Next, in order to feed these timeseries data as the input data, they are then normalized for Tensorflow to deal with. As shown in fig.22, the timeseries data are normalized into a 2D matrix, with the value distribution from -1 to 1.

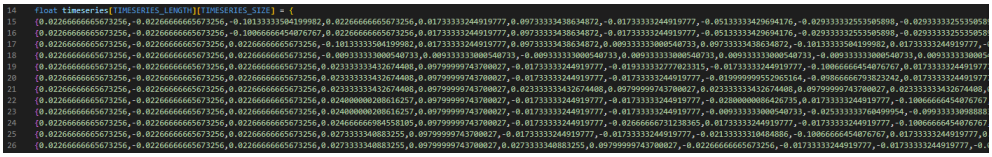

Figure 22: Normalized timeseries data

In terms of inference efficiency, which is also a very important indicator for our application classification, for real-time and limited computation ability. Moreover, although large batch size and more computation resources can make inference faster, this is not realistic in network traffic scenario, where we should process the packets flow immediately when it arrives to a low-power router, rather than waiting to gather a large batch. Thus, to better fit our special use case, we choose a small batch size of 20 for inference. After inference, a output vector containing the probability distribution of certain application is obtained. By comparing the most probable application with the expected one, we can get the accuracy of this ML model as a factor of ML performance.

3.2 Performance with TensorflowLite

The inference speed is the main factor of performance we concern. I thus run the inference for orignal model and its quantized model to profile the performance, including the accuracy and inference time. To measure the time, the tool Cprofile in python is used to get the time passed during a program. Due to the very fast inference speed, a million of inference are run to get the average inference time for each inference. There are 6 types of quantization including the original model : original, dynamic, float16, int16, int8 and uint8. For int8 and uint8 the input vector type is int8, whereas input type for the others is float32. The result is obtained in ARM environment.

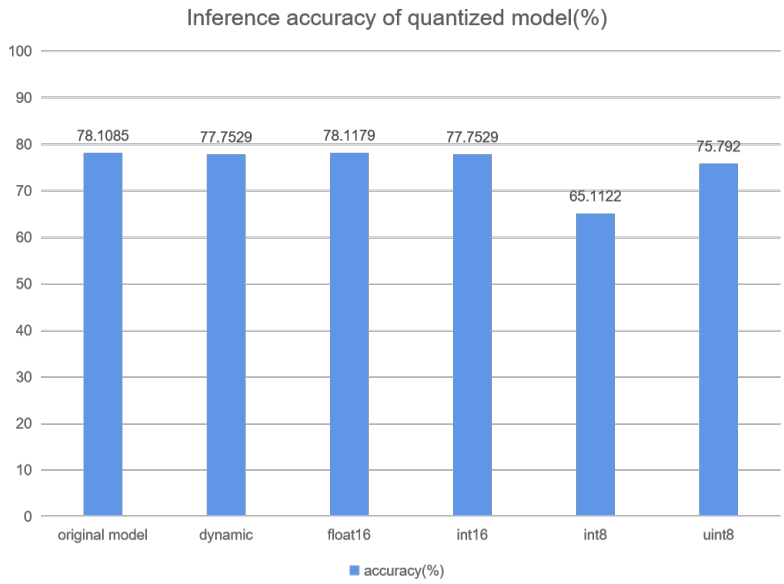

Figure 23: Accuracy for original and quantized ML model

As shown in fig.23, we observe that in terms of accuracy, there is nearly no difference among the original model and quantized models, which is reasonable since our model size is about 300KB and the number of parameters is not much. As a result, the quantization has less impact on inference accuracy. The accuracy for int8 quantized model is exceptional, the accuracy drops about 10%, but I cannot figure out the reason of such weird result. As for the inference time, a obvious improvement is performed thanks to the quantization as shown in fig.24. Time is reduced from 140us to 90us, around one third of original performance. Except the float16 quantized model, which nearly does not change anything, for which I do not know the reason neither. Due to the imperfection of quantization tool of TensorflowLite, it may because the incompatibility of our model.

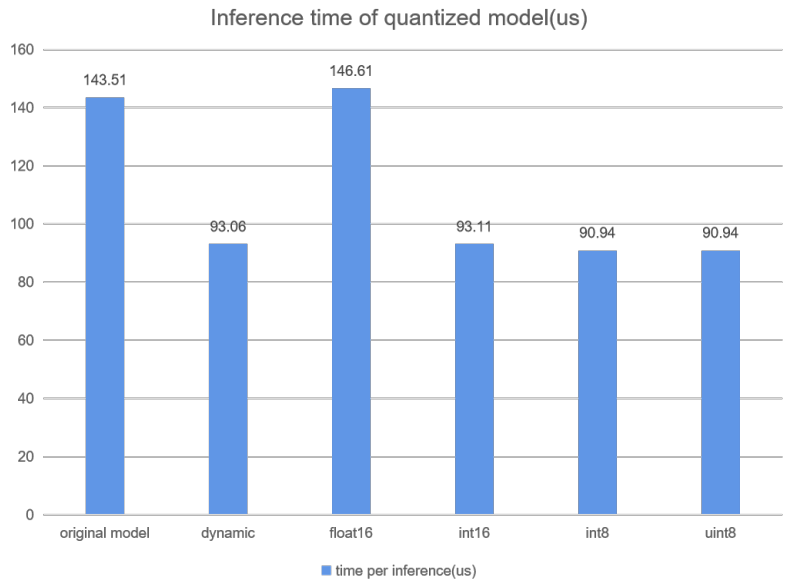

Figure 24: Inference time for original and quantized ML model

3.3 Performance with MindsporeLite

3.3.1 Benchmark Timeprofiling

Despite the unreasonable exceptional case for TensorflowLite, the quantization performs a significant improvement for inference speed. Given that our aim is to run the inference in Huawei's edge device, we have to focus on the performance of our model with MindsporeLite in ARM environment. Since the original model file is Tensorflow model, it is necessary to convert the model into MindsporeLite model by the converter tool integrated in MindsporeLite. Also, this framework has a benchmark tool allowing us to run the benchmark to profile the performance directly without obtaining the accuracy, which I prove equally than the result obtained with TensorflowLite.

Thanks to the benchmark tool provided by MindsporeLite, we easily get the inference time for each layer of our model. The fig.25 demonstrates the inference time distribution of our model. The total time passed is around 267us, which is longer than the one obtained from TensorflowLite. Besides, there are tens of layers representing different operators in the model. Among all the operators, the critical one is the $Conv2DFusion$. This layer taking nearly half inference time contains all the convolution function, including $ConvolutionCPUKernel$, $Convolution1x1CPUKernel$, $ConvolutionDepthwiseSWCPUKernel$ and $ConvolutionDepthwiseCPUKernel$. The other layers are individually less significant in terms of performance optimization.

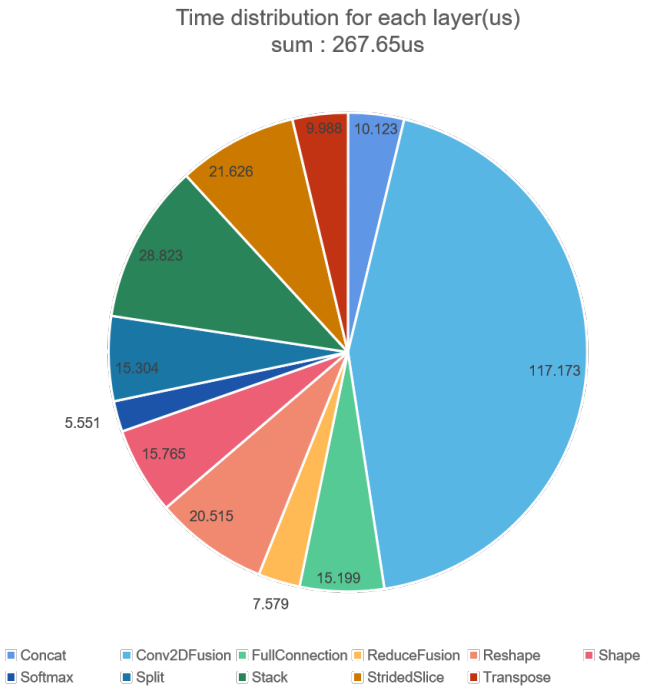

Figure 25: Time distribution for each layer of our model profiling MindsporeLite benchmark

3.3.2 Benchmark Perf Profiling

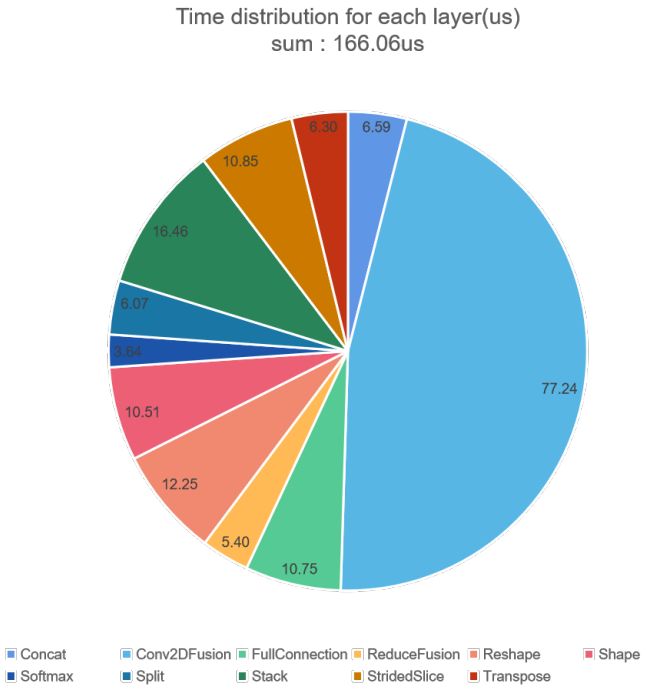

Figure 26: Time distribution for each layer of our model profiling MindsporeLite benchmark using perf tool

Knowing the time distribution of each operator, we wonder the real execution time in the kernel, since the benchmark takes some time to print the information which may cause a huge

perturbation when the inference time is comparable to the *print* functions in the kernel. The time spent other than in the kernel is the overhead of program, and should be excluded when measure the performance. In this case, I managed to use perf tool to profile the benchmark, in order to find out the real time distribution of each function in the kernel. To do this, I firstly modified the source code to print the function name and its corresponding operator name. Then by applying the perf tool, I can thus profile the real performance in the kernel. As shown in fig.26, the percentage of time distribution of each operator does not change much, but the total time becomes 166us. This result is more convincible since the perf tool profiles the function execution time in the kernel.

3.3.3 Quantizatoin performance

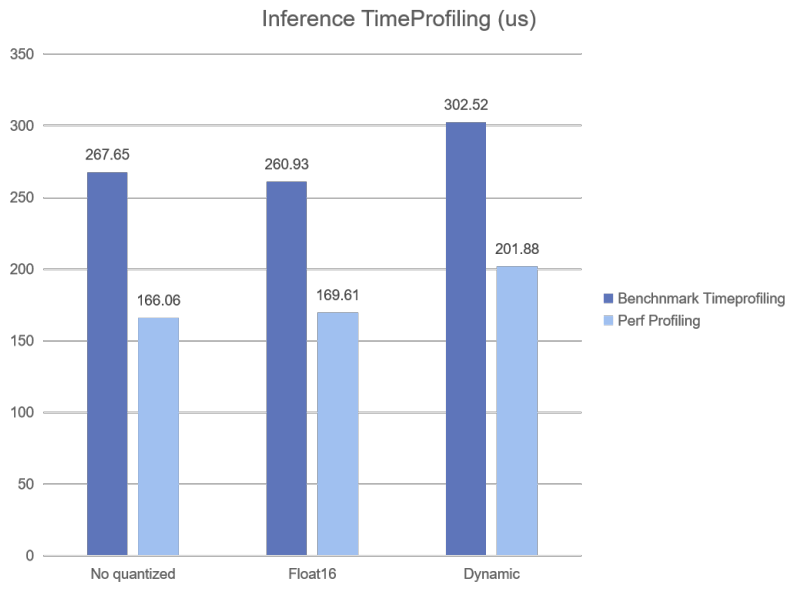

Figure 27: Different timeprofiling for different quantization

For MindsporeLite, quantization performance improvement is also worthy to be researched. The model is converted with the already quantized TensorflowLite model, since the MindsporeLite is on development and does not have a compatible quantization tool as TensorflowLite. As shown in fig.27, the total inference time is profiled both with benchmark's timeprofiling tool and perf profiling tool for 3 types model, no quantized, float16 and dynamic quantized model, since I met some problems converting the other quantized TensorflowLite model to MindsporeLite model. From which we observe that float16 quantized model has the same inference time as the no quantized one. Whereas the dynamic quantized model is slightly slower than the others for around one quarter.

The result shows that the quantization has nearly no optimization on inference speed, which is also odd. It may be the problem of the MindsporeLite benchmark tool, or the quantization is not compatible within these 2 frameworks. On the other hand, the quantization has some benefit on the size of model as shown in fig.28. As for TensorflowLite model, the no quantized model size is 300KB, and is decreased to one half and one third respectively for float16 and dynamic quantization. The MindsporeLite model is converted from the TensorflowLite model, it has the same quantization optimization as the TensorflowLite model except the float16 one. This may be the quantization compatibility reason as well.

ARM_dynamic		ARM_float16		ARM_No_quantized	
TFLite model size	119KB	TFLite model size	187KB	TFLite model size	333KB
MSLite model size	111KB	MSLite model size	320KB	MSLite model size	325KB

Figure 28: Quantized and no quantized model size for TensorflowLite and MindsporeLite

3.4 Hardware Accelerator Performance

We managed to research the performance improvement of a hardware accelerator, since this accelerator can drastically speed up the convolutional operator. To do this, I have the chance to get access to use the Ascend310 server. The Ascend310 is a CPU with several Da Vinci AI cores integrated. It is dedicated for AI computation, and only available with MindsporeLite. During the usage, I run benchmark timeprofiling for some classical neural network such as mobilenetv1, mobilenetv2 and mnasnet respectively with and without the participation of hardware accelerator as following.

mobilenet_v1

ACL=off

```
opType                  avg(ms)         percent         calledTimes     opTotalTime
AvgPoolFusion           0.039180        0.000388        100             3.918001
Conv2DFusion            100.996162      0.999480        2800            10099.616211
Reshape                 0.004660        0.000046        100             0.466000
Softmax                 0.008840        0.000087        100             0.884000
Model = mobilenet_v1_1.0_224.tflite.ms, NumThreads = 1, MinRunTime = 100.933998 ms, MaxRuntime = 104.627998 ms, AvgRunTime = 101.197998 ms
```

ACL=on

```
opName          avg(ms)         percent         calledTimes     opTotalTime
custom_0        1.440770        0.935602        100             144.076996
input_nh2nc     0.099170        0.064399        100             9.916997
-------------------------------------------------------------------------
opType          avg(ms)         percent         calledTimes     opTotalTime
Custom          1.440770        0.935602        100             144.076996
Transpose       0.099170        0.064399        100             9.916997
Model = mobilenet_v1_1.0_224_ACL.tflite.ms, NumThreads = 1, MinRunTime = 1.565000 ms, MaxRuntime = 1.615000 ms, AvgRunTime = 1.586000 ms
```

Figure 29: Performance enable and disable hardware accelerator for mobilenetv1

As we observe in fig.29, the above timeprofiling result shows that for each inference without hardware accelerator takes 101ms, during which the operator $Con2DFusion$ spends 100ms, nearly 99% of the time usage. Whereas the result in the bottom shows that all the operators are transformed to custom and transpose layer, which takes only 1.5ms in total. From 100ms to 1.5ms, the accelerator brings an extremely huge time gain to the inference performance.

mobilenetV2

NO_ACL

opType	avg(ms)	percent	calledTimes	opTotalTime
AddFusion	0.213210	0.004439	1000	21.320999
Conv2DFusion	46.149796	0.960758	5200	4614.979492
MatMulFusion	1.565500	0.032591	100	156.550018
ReduceFusion	0.054330	0.001131	100	5.432994
Softmax	0.008260	0.000172	100	0.826000
Transpose	0.043820	0.000912	100	4.381999

Model = mobilenetv2.mindir.ms, NumThreads = 1, MinRunTime = 48.188999 ms, MaxRuntime = 50.665001 ms, AvgRunTime = 48.319000 ms

converted ACL

opName	avg(ms)	percent	calledTimes	opTotalTime
custom_0	1.824650	0.896529	100	182.464981
x_nh2nc	0.210590	0.103472	100	21.059006

opType	avg(ms)	percent	calledTimes	opTotalTime
Custom	1.824650	0.896529	100	182.464981
Transpose	0.210590	0.103472	100	21.059006

Model = mobilenetv2_ACL.mindir.ms, NumThreads = 1, MinRunTime = 2.044000 ms, MaxRuntime = 2.096000 ms, AvgRunTime = 2.070000 ms

Figure 30: Performance enable and disable hardware accelerator for mobilenetv2

mnasnet

NO_ACL

opType	avg(ms)	percent	calledTimes	opTotalTime
AddFusion	0.054830	0.003014	1000	5.483019
Conv2DFusion	16.501850	0.906982	5200	1650.184937
MatMulFusion	1.614450	0.088734	100	161.444977
ReduceFusion	0.023000	0.001264	100	2.300000

Model = mnasnet_1.0_96.pb.ms, NumThreads = 1, MinRunTime = 18.058001 ms, MaxRuntime = 21.891001 ms, AvgRunTime = 18.437000 ms

ACL

opName	avg(ms)	percent	calledTimes	opTotalTime
custom_0	1.177270	1.000000	100	117.726997

opType	avg(ms)	percent	calledTimes	opTotalTime
Custom	1.177270	1.000000	100	117.726997

Model = mnasnet_1.0_96.pb_ACL.ms, NumThreads = 1, MinRunTime = 1.180000 ms, MaxRuntime = 1.236000 ms, AvgRunTime = 1.206000 ms

Figure 31: Performance enable and disable hardware accelerator for mnasnet

As well as mobilenetv1, the result shown in fig.30 and fig.31 respectively demonstrate the performance of models mobilenetv2 and mnasnet. From the results above we can conclude that the hardware accelerator plays a critical role for the inference since it dramatically reduce the time consumed in convolutional layer. The more convolutional layer the model has, the more benefits

the accelerator can bring. In general, the convolutional computation spends more than 99% in each inference. However for our project, the optimization is limited due to the relatively light size model. The improvement is not so obvious as expected. Moreover, an accelerator takes physical space in our network equipment, which is worthy to be discussed. Despite the large performance improvement, one may not be ignored is that the communication overhead between the CPU (host) and accelerator (device) is not negligible.

4 Conclusion

This 6-month internship is a very impressive working experience of my stay in France. During this 6 months, I have established a knowledge system of data communication. Along with that, the extensive reading about network allows me to know better the Internet world. By solving software problems, My programming skills have also been much improved. Despite being a little bit far away from my major, CPU architecture is still an interesting subject that I would like to work on after graduation. As a master student, I think a excellent engineer is supposed to be able to deal with various technique problem, instead of repeating the existing work.

On the other hand, I struggle to finish the hard job that they assign to me with of help of my tutors Benjamin and Camille. They 2 excellent engineers are so experienced, and patient to teach me. Not only they told me the solution of problem, but also the methodology on how to solve it. The basic idea is to split the problematic into several sub-problems which can be splitted into smaller individual problems. As a result now matter what problem is seen as a group of small problems which can be solved much easier.

Finally, this internship experience in company helps me a lot to adapt the working environment for my carrier in the future. Unlike school or laboratory, in the company there are different rules to respect. And a nice relationship with colleagues will much facilitate the job.

References

[1] "Application Classification". In: (). URL: https://media.geeksforgeeks.org/wp-content/uploads/20220522144232/portnumbers.png.

[2] "Are the Linux utilities parts of the kernel/shell". In: (). URL: https://askubuntu.com/questions/161511/are-the-linux-utilities-parts-of-the-kernel-shell.

[3] "cityinfrastructure". In: (). URL: http://cityinfrastructure.com/Data/Data.html.

[4] "Feature Extraction". In: (). URL: https://d3i71xaburhd42.cloudfront.net/4824ab867ccf635addf386280503-Figure1-1.png.

[5] Pamela Fox. "IP packets". In: (). URL: https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:routing-with-redundancy/a/ip-packets.

[6] "Handwritten digit recognition using convolutional neural networks". In: (). URL: https://blog.codecentric.de/en/2019/12/handwritten-digit-recognition-using-convolutional-neural-networks/.

[7] "How to run Keras model on RK3399Pro". In: (). URL: https://www.dlology.com/blog/how-to-run-keras-model-on-rk3399pro/.

[8] "Huawei Unlocked E8372 4G/LTE Wi-Fi Dongle - White". In: (). URL: https://www.amazon.de/HUAWEI-E8372-WiFi-WLAN-Modem/dp/B014WMEJ2Q/ref=asc_df_B014WMEJ2Q?tag=bingshoppin0b-21&linkCode=df0&hvadid=80264405710992&hvnetw=o&hvqmt=e&hvbmt=be&hvdev=c&hvlocint=&hvlocphy=&hvtargid=pla-458386398066912 7&th=1.

[9] "Introduction of Firewall in Computer Network". In: (). URL: https://www.geeksforgeeks.org/introduction-of-firewall-in-computer-network/.

[10] "Openflow-table-entry-med". In: (). URL: https://commons.wikimedia.org/wiki/File:Openflow-table-entry-med.png.

[11] "TensorFlowgraph". In: (). URL: https://sourcedexter.com/what-is-tensorflow/tensorflow-graph1/.

[12] "(CNN)". In: (). URL: https://blog.csdn.net/weixin_41863685/article/details/79871058.

[13] "". In: (). URL: https://www.cnblogs.com/wt-seu/p/12771894.html.