Deep Learning Inference frameworks for ARM CPU

Vanessa Courville — Huawei Noah's Ark Lab, Canada
Vahid Partovi Nia — Huawei Noah's Ark Lab, Canada

Abstract

The deep learning community focuses on training networks for a better accuracy on GPU servers. However, bringing this technology to consumer products requires inference adaptation of such networks for low-energy, small-memory, and computationally constrained edge devices. ARM CPU is one of the important components of edge devices, but a clear comparison between the existing inference frameworks is missing. We provide minimal preliminaries about ARM CPU architecture and briefly mention the difference between the existing inference frameworks to evaluate them based on performance versus usability trade-offs.

1 Introduction

Real-time performance of deep model inference models is crucial for most end-user applications. As neural network architectures continue to become deeper and more complex, how these models are deployed on hardware devices has a direct impact on response time. As a result, optimized and architecture-dependent implementations are typically required to ensure maximum performance and usability.

Many open-source inference frameworks exists that target different types of hardware devices. These devices include server-grade CPUs, such as Intel x86 architectures, smaller embedded CPUs such as ARM, domain-specific ASICs, or even reprogrammable FPGAs. In all cases, the devices chosen are directly dependent on the application.

One of the most common and readily-available hardware architectures used for neural network inference is the ARM CPU which are based on Reduced Instruction Set Computer, ARM architectures are designed to support a reduced set of instructions, resulting in a smaller die size, reduced energy requirements and allowing it to run at higher frequencies. As a result, ARM processors are typically the go-to architecture for most small consumer electronics.

2 ARM Architecture

The ARM CPU architecture is typically made up of a multi-core design wherein each core contains a 32-bit/64-bit CPU, a NEON SIMD accelerator, a floating point unit, and a series of cache, as shown in Figure 1

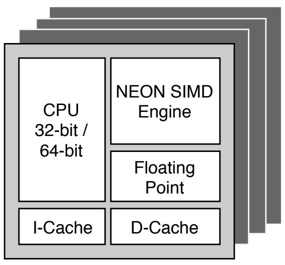

Fig. 1: Example of typical ARM architecture.

2.1 NEON SIMD Engine

Single instruction, multiple data (SIMD) programming is a popular form of parallel processing supported by most modern CPU architectures. Based on the concept of vectorization, a parallel programming paradigm where an algorithm is modified to run a single instruction on multiple different sets of data points concurrently. SIMD processors are typically represented as specialized hardware accelerator components of a CPU. For ARM architectures, the SIMD accelerators is known as NEON [1]. The NEON accelerator can be programmed using either C-intrinsics or assembly programming. Figure 2 shows the flow of a simple SIMD processor.

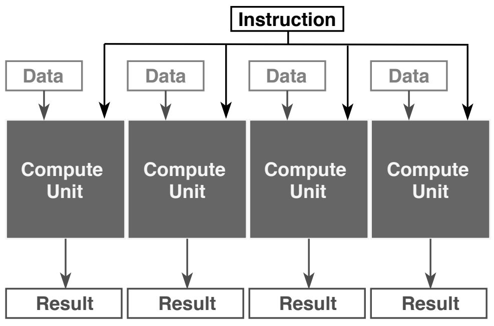

Fig. 2: Example of typical SIMD architecture.

The NEON SIMD accelerator is a key component of ARM architectures and is vital for any performance-driven applications. It allows for an other-wise small, inefficient processor to maximize compute capability and significantly increase the number of concurrent operations possible.

The computations required in most neural network operations can be treated as an embarrassingly parallel workload as a very large number of independent operations take place. This makes it very straight-forward to vectorize most neural networks operations, making it a perfect application for SIMD acceleration.

2.2 big.LITTLE

ARM systems with big.LITTLE architectures [2] are heterogeneous systems made up of multi-core processors of different compute capabilities. Typically, the system will contain both smaller, power-saving processors (LITTLE) as well as larger, more compute-intensive processors (big). Typically used for consumer electronics, these architectures are designed to maximize both power savings and performance by allowing the designer to pick and choose which processor to use depending on the application requirements. An example of a typical big.LITTLE architecture is shown in Figure 3

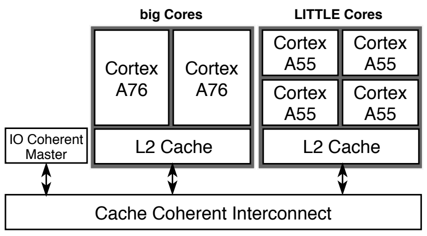

Fig. 3: Example of an ARM big.LITTLE architecture.

These types of architectures are typically found in most embedded devices where performance and power-efficiency are imperative. Many mobile phones currently available in the market use the ARM big.LITTLE architecture as a means of splitting up tasks on the device. Performance-critical applications will run on the 'big' processors whereas background applications will run on the 'LITTLE' processors.

As many neural network applications run on mobile devices, it is vital that any inference framework designed to target mobile platforms must be able to efficiently take advantage of this architecture.

3 Quantization

As neural networks continue to evolve, the number of applications for typical consumer electronics continues to increase. Applications such as speech recognition, object character recognition and model personalization all require large, complex neural networks to provide high accuracy. These models require significant data storage and compute capability which are constrained on edge devices.

In order to run large neural network models on these edge devices, the models must be compressed to not only reduce the overall storage size, but also reduce the number of computations needed to ensure that results can be achieved within a reasonable delay and save energy consumption.

The most common technique of model compression is quantization. Quantization involves reducing precision, of either weights, activations, or both in a typical neural network [3]. Say a full precision model contains weights and activations that are each 32-bit floating point values. A quantized version of this model could involve replacing the 32-bit float weights/activations with quantized 8-bit integers. This not only reduced storage by a factor of 4 times, but also simplifies the compute required as there are no longer any floating-point multiply-accumulate operations taking place. Instead these can be replaced by simple 8-bit integer multiplications, effectively simplifying the hardware, reducing the compute latency, and saving energy.

In order to successfully support these compressed models, inference frameworks targeting edge devices must be able to support these quantization techniques. Therefore, an inference framework must support different quantized data precisions for both weights and activations.

4 Existing Inference Frameworks

There is currently no shortage of available open-source inference frameworks targeting ARM architectures. These frameworks vary in terms of performance, flexibility, usability, and data-precision support.

4.1 ncnn

Tencent's neural network inference framework (ncnn) [4] is highly optimized to target mobile platforms using ARM CPUs. The framework targets the NEON accelerator using a combination of C intrinsics and low-level assembly instructions to maximize performance. Written entirely in `C++`, it contains a big.LITTLE multi-core scheduler, and does not depend on any third party library. The framework can also support 8-bit quantization, including 8-bit compute operations and half precision storage. It can also be extended to support GPU offload. However, currently it does not support very low-bit precision such as binary networks.

4.2 mnn

Alibaba's Mobile Neural Network (mnn) [5] inference framework strength is in its versatility. It can be integrated into multiple high-level frameworks such as TensorFlow, ONNX, and Caffe. Much like ncnn, it targets performance using hand-coded NEON assembly code, and supports heterogeneous environments by offloading certain operations to the GPU. MNN does not support 8-bit quantization, but does support full half-precision compute (using ARM v8.2 new features) differentiating it from ncnn.

4.3 NeoCPU

Amazon Web Services' NeoCPU [6]was designed to be generic and does not target a specific hardware architecture. Instead, the framework supports multiple hardware devices including Intel, AMD and ARM CPUs because the framework is built upon a TVM deep learning compiler stack. The code does not use any intrinsic of assembly code for performance optimization, but instead focuses on algorithmic improvements such as transformation techniques, layout optimizations and search space optimization. Although versatile, it does not currently support low-bit precision networks.

4.4 BMXNet-v2

Hasso Platner Institute's BMXNet-v2 [7] is an extension of the popular MXNet framework targeting binary and quantized networks. This framework differs from the other frameworks by the fact that it supports a wide array of quantization schemes including, 2 to 31 bits. Although it does not target performance, its wide range of support is intended to allow researchers to evaluate different quantization schemes.

4.5 dabnn

JD AI Research Group's dabnn [8] is a binary neural network inference framework targeting mobile platforms. The framework is designed for highly optimized binary convolutions using ARM assembly code. It integrates easily into ONNX by providing a conversion tool. Although optimized for binary networks, DABNN does not support 8-bit or half precision data formats.

4.6 ARM Compute Library

ARM's neural network compute library [9] is a set of pre-built binary functions for common neural network functions. Provided by ARM directly, the library is highly-optimized and takes full advantage of the NEON SIMD architecture. It supports some low-precision operations such as 8-bit quantized convolutions but currently does not support binary or half-precision networks. It is packaged as a stand-alone library and does not have a pre-defined integration into high-level frameworks.

4.7 Inference Framework Comparison

Each of the inference frameworks evaluated have varying sets of trade-offs. Table 1 summarizes the features of all networks evaluated.

Feature	ncnn	mnn	Neo	BMXNet	dabnn	CL
NEON	✓	✓	x	✓	✓	✓
b.L.	✓	x	x	x	x	x
GPU	✓	✓	x	x	x	✓
half	Storage	✓	x	x	x	x
8-bit	✓	x	x	✓	x	✓
binary	x	x	x	✓	✓	x
Android	✓	✓	x	✓	✓	✓
TF/ONNX	x	✓	✓	x	✓	✓
Opt	x	x	✓	x	x	x

Table 1: Inference framework feature support comparison

5 Discussion

Many companies and academics have attempted to design performance-optimized inference frameworks. These frameworks vary as some target versatility, with the intent of being able to use the same framework for different hardware devices, while others target performance by hand-tuning the implementation using assembly code, or by focusing on supporting low-bit quantized networks.

However, what is lacking is a comprehensive framework whose sole focus is performance targeting resource constrained edge devices. In order to get optimal performance on these devices, a framework must be able to take full advantage of the hardware resources available to it, including being able to maximize performance on the SIMD accelerator, be able to schedule jobs across a heterogeneous architecture including a multi-core big.LITTLE architecture or GPU offload. It must have full support or model compression techniques like quantization with multiple different supported precision types. In addition to these essentials, the framework should also take advantage of algorithmic optimization techniques as seen by the NeoCPU framework.

Industry and academia alike would significantly benefit from such a framework as it would ensure that all neural networks models can be tested using realistic consumer-driven hardware and conditions.

6 Acknowledgement

We appreciate fruitful technical discussions with Huawei Cloud Core Shanghai colleagues Gang Chi and Pengcheng Tang. We also thank Yan Zhou and Yanhui Geng for their support throughout the project.

References

[1] ARM. Introducing neon development article. Technical report, 2009.

[2] ARM. big.little technology: The future of mobile. Technical report, 2013.

[3] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

[4] Tencent. ncnn. `https://github.com/Tencent/ncnn`, 2019.

[5] Alibaba. mnn. `https://github.com/alibaba/MNN`, 2019.

[6] Yu Li Shaarma Wang Liu, Wang. Optimizing cnn model inference on cpus. *USENIX Annual Technical Conference 2019 1025-1040*, 2019.

[7] Hasso Platner Institute. Bmxnet-v2. `https://github.com/hpi-xnor/BMXNet-v2`, 2019.

[8] JD AI Research Group. dabnn. `https://github.com/JDAI-CV/dabnn`, 2019.

[9] ARM-Software. Arm compute library. `https://github.com/ARM-software/ComputeLibrary`, 2019.