Mobile Machine Learning Hardware at ARM: A Systems-on-Chip (SoC) Perspective

Yuhao Zhu*
Department of Computer Science
University of Rochester
yzhu@rochester.edu

Matthew Mattina
Machine Learning & AI
ARM Research
matthew.mattina@arm.com

Paul Whatmough
Machine Learning & AI
ARM Research
paul.whatmough@arm.com

ABSTRACT

Machine learning is playing an increasingly significant role in emerging mobile application domains such as AR/VR, ADAS, etc. Accordingly, hardware architects have designed customized hardware for machine learning algorithms, especially neural networks, to improve compute efficiency. However, machine learning is typically just one processing stage in complex end-to-end applications, which involve multiple components in a mobile Systems-on-a-chip (SoC). Focusing on just ML accelerators loses bigger optimization opportunity at the system (SoC) level. This paper argues that hardware architects should expand the optimization scope to the entire SoC. We demonstrate one particular case-study in the domain of *continuous computer vision* where camera sensor, image signal processor (ISP), memory, and NN accelerator are synergistically co-designed to achieve optimal system-level efficiency.

1 INTRODUCTION

Mobile devices are the most prevalent computing platform of the present day, and are dominated by the ARM architecture. A huge number of emerging mobile application domains now rely heavily on machine learning; in particular, various forms of deep neural networks (DNNs) have been instrumental in driving progress on problems such as computer vision and natural language processing. On mobile platforms, DNN inference is currently typically executed in the cloud. However, the trend is to move the DNN execution from the cloud down to the mobile devices themselves. This shift is essential to remove the communication latency and privacy issues of the cloud offloading approach.

The increasing use of DNNs in mobile applications places massive compute requirements on the mobile System-on-chip (SoC), which must now process tens of billions of linear algebra operations per second under a tight energy budget. In response, there has been a huge effort expended on dedicated hardware to accelerate the computation of neural networks. This is borne out in a proliferation of papers describing architectures for DNN accelerators (NNX), which typically demonstrate high computational efficiency on the order of 0.4 – 3.8 TOPS/W on convolutional NN inference [12, 13, 15, 17]. This is several orders of magnitude more efficient than typical mobile CPU implementations.

Sadly, the efficiency benefits of hardware accelerators are largely a one-time improvement, and will likely saturate, while the compute requirement of DNNs keep increasing. Using computer vision as an example, today's convolutional neural network (CNN) accelerators are not able to perform object detection (e.g., YOLO[16]) in real time at 1080p/60fps. As the resolution, frame rate, and the need for stereoscopic vision grows with the emergence of AR/VR use cases, the compute requirement will continue to increase, while the power budget remains constant, leaving a large gap.

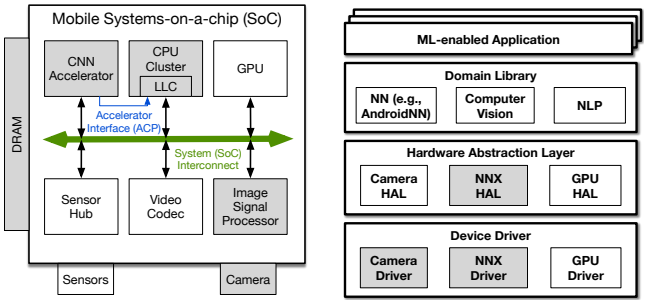

Figure 1: Mobile SoC (left) and software stack (right). Shaded components are used in continuous vision tasks.

Therefore, we must move from a narrow focus on hardware accelerators to begin to consider system-level optimizations for ML on mobile. Expanding the scope beyond just the CNN accelerator to consider the whole SoC, we emphasize three areas for optimization:

- **Accelerator Interfacing** : hardware accelerators must be efficiently interfaced with the whole system for full benefit.
- **Software Abstractions** : for cross-platform compatability, SoC details should be abstracted with a clear API.
- **System Optimizations** : Co-design of algorithms and the various hardware blocks in the system.

Section 2 will describe these optimizations further, with a case study in Section 3. We conclude in Section 4.

2 ML ON MOBILE SYSTEMS

We have already started to see changes in mobile systems in response to the computational demands of deep learning methods. Most notably, NNX components are now common in mobile SoCs, e.g., the Neural Engine in the iPhoneX [2] and the HPU CNN co-processor in the Microsoft HoloLens [7]. However, a significant challenge still remains as to how to integrate NNX components into the system. We identify three aspects from both hardware and software perspectives.

Accelerator Interfacing There are two main challenges in interfacing NNX IPs to the rest of the mobile hardware system: (1) providing an efficient path to offload an NN invocation task from the CPU to the NNX, and (2) providing sufficient memory bandwidth for weights and activation data. In particular, the cache/memory interface between the accelerator and the SoC is critical, since modern DNNs typically have very large parameter sets, which demand high memory bandwidth to feed many arithmetic units [10].

*Work done while a visiting researcher at ARM Research

Our key insight is to leverage the last level cache (LLC) in the CPU cluster as a bandwidth filter rather than directly interfacing the NNX with the DRAM as most state-of-the-art NNX designs currently do [11]. This is achieved through the ARM Accelerator Coherency Port (ACP) [3], which is available on ARM CPU clusters and allows attached accelerators to load and store data directly to and from the LLC. In this way, the NNX can also take advantage of LLC features in the CPU cluster such as prefetching and cache stashing. Furthermore, ACP is also low-latency, such that the CPUs and NNX can work together closely on data in the LLC.

Software Abstractions Mobile SoC hardware and ML algorithms are both evolving rapidly. Therefore, it is paramount to present a programming interface that minimizes disruption to application developers. This would make new hardware features easy to use, and provide compatibility across a range of mobile SoCs.

The key of such a programming interface is a clear abstraction that allows applications to execute DNN jobs efficiently on (one of many) hardware accelerators, or fall back to execution on a CPU or GPU. The AndroidNN API [1] provides an example of this principle, by abstracting common DNN kernels such as convolution, and scheduling execution through a hardware abstraction layer (HAL).

We built directly on top of AndroidNN to benefit the vast majority of mobile devices. Specifically, we provide optimized implementations for DNN kernels in the form of ARM Compute Library [4]. The library takes advantage of recent ARM ISA enhancement that provides new instructions for essential linear algebra operations behind DNNs, such as the new dot-product instructions in the Scalable Vector Extensions (SVE) [5]. Finally, we provide IP-specific drivers and HAL implementations to support the AndroidNN API.

System Optimizations While adding specialized NNX hardware IP to the SoC improves kernel-level performance and efficiency, the DNN invocation is typically only one stage in the larger end-to-end application pipeline. For instance in computer vision, many on/off-chip components such camera sensors, Image Signal Processors (ISP), DRAM memory, as well as the NNX have to collaborate together to deliver real time vision capabilities. The NNX IP itself constitutes at most half of the total power/energy consumption. Therefore, we must jointly optimize the whole system.

Once we expand our scope to the system level, we expose new optimization opportunities by exploiting functional synergies across different IP blocks; these optimizations are not obvious when considering the NNX in isolation. We will demonstrate this principle in the following case study.

3 CASE STUDY: CONTINUOUS VISION

Computer vision (CV) tasks such as object classification, localization, and tracking are key capabilities for a number of exciting new application domains on mobile devices, such as augmented reality (AR) and virtual reality (VR). However, the computational cost of modern CV CNNs far exceed the severely limited power budget of mobile devices, especially for real time (e.g., 1080p/60fps). This is true even with a dedicated CNN hardware accelerator [12, 13, 15].

To achieve real-time object detection with high accuracy on mobile devices, our key idea is to reduce the total number of expensive CNN inferences through system-level optimizations. This is done by harnessing the synergy between different hardware IPs

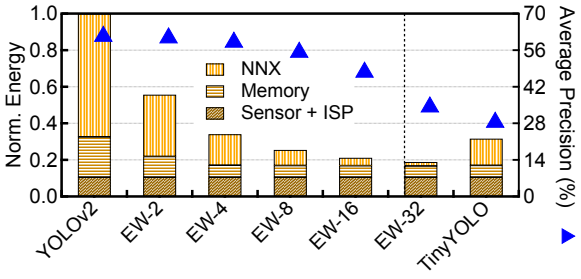

Figure 2: Cross-IP optimization of object detection on a mobile SoC allows over 40% reduction in energy (left) with less than 1% accuracy loss (right).

of the vision subsystem. In particular, we leverage the fact that the image signal processor (ISP) inherently calculates motion vectors (MV) for use in its temporal denoising algorithms. Usually MVs are discarded after de-noising, but we elect to expose them at the system level. Instead of using CNN inference on every frame to track the movement of objects, we reuse the MVs to extrapolate the movement of objects detected in the previous video frame, without calculating further CNNs for the current frame. As we increase the number of consecutively extrapolated frames (extrapolation window, or EW), the total number of CNN inferences is reduced, leading to performance and energy improvements.

We also leverage the ARM ACP interface [3] to use the LLC for inter-layer data reuse (e.g., feature maps), which would otherwise be spilled to the DRAM from the NN accelerator's local SRAM. A typical LLC size in mobile devices is about 2 MB [14] and ACP provides around 20 GB/s of bandwidth, which is sufficient to capture the reuse of most layers in today's object detection CNNs. This design greatly reduces DRAM and system power consumption.

Finally, we present software support that abstracts away the hardware implementation details. As Fig. 1 shows, the high-level CV libraries are unmodified, keeping the AndroidNN interface unchanged. We implement specific driver and HAL modifications that our hardware augmentation entails.

We evaluated the system-level optimizations on an in-house SoC simulator, which we calibrated with measurements on the Jetson TX2 development board [6]. We use commonly-used benchmarks such as VOT [9] and OTB [8] as well as our internal datasets. Results in Fig. 2 show that compared to state-of-the-art object detection frameworks such as YOLO [16] that execute an entire CNN for every frame, our system reduces the energy by over 40% with less than 1% accuracy loss at an extrapolation window (EW) size of two. The energy saving is greater as EW increases, while accuracy degrades. Compared to the conventional approach of reducing the compute intensity by down-scaling the network (e.g., TinyYOLO, which is $\sim 5\times$ simpler), our system achieves higher energy savings and higher accuracy.

4 CONCLUSION

Efficiently supporting demanding ML workloads on energy-constrained mobile devices requires careful attention to the overall system design. We emphasized three key research priorities: accelerator interfacing, software abstractions, and cross-IP optimizations.

REFERENCES

[1] [n. d.]. Android Neural Networks API. ([n. d.]). https://developer.android.com/ndk/guides/neuralnetworks/index.html
[2] [n. d.]. Apple's Neural Engine Infuses the iPhone with AI Smarts. ([n. d.]). https://www.wired.com/story/apples-neural-engine-infuses-the-iphone-with-ai-smarts/
[3] [n. d.]. ARM Accelerator Coherency Port. ([n. d.]). http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0434a/BABGHDHD.html
[4] [n. d.]. Arm Compute Library. ([n. d.]). https://github.com/ARM-software/ComputeLibrary
[5] [n. d.]. Arm Scalable Vector Extensions and application to Machine Learning. ([n. d.]). https://developer.arm.com/hpc/arm-scalable-vector-extensions-and-application-to-machine-learning
[6] [n. d.]. Jetson TX2 Module. ([n. d.]). http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html
[7] [n. d.]. Second Version of HoloLens HPU will Incorporate AI Coprocessor for Implementing DNNs. ([n. d.]). https://www.microsoft.com/en-us/research/blog/second-version-hololens-hpu-will-incorporate-ai-coprocessor-implementing-dnns/
[8] [n. d.]. Visual Tracker Benchmark. ([n. d.]). http://cvlab.hanyang.ac.kr/tracker_benchmark/datasets.html
[9] [n. d.]. VOT2014 Benchmark. ([n. d.]). http://www.votchallenge.net/vot2014/
[10] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning. In *Proc. of ASPLOS*.
[11] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-efficient Dataflow for Convolutional Neural Networks. In *Proc. of ISCA*.
[12] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. In *Proc. of ISSCC*.
[13] Giuseppe Desoli, Nitin Chawla, Thomas Boesch, Surinder-pal Singh, Elio Guidetti, Fabio De Ambroggi, Tommaso Majo, Paolo Zambotti, Manuj Ayodhyawasi, Harvinder Singh, et al. 2017. 14.1 A 2.9 TOPS/W Deep Convolutional Neural Network SoC in FD-SOI 28nm for Intelligent Embedded Systems. In *Proc. of ISSCC*.
[14] Matthew Halpern, Yuhao Zhu, and Vijay Janapa Reddi. 2016. Mobile CPU's Rise to Power: Quantifying the Impact of Generational Mobile CPU Design Trends on Performance, Energy, and User Satisfaction. In *Proc. of HPCA*.
[15] Bert Moons, Roel Uytterhoeven, Wim Dehaene, and Marian Verhelst. 2017. 14.5 Envision: A 0.26-to-10TOPS/W Subword-parallel Dynamic-Voltage-Accuracy-Frequency-Scalable Convolutional Neural Network Processor in 28nm FDSOI. In *Proc. of ISSCC*.
[16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. In *Proc. of CVPR*.
[17] P. N. Whatmough, S. K. Lee, H. Lee, S. Rama, D. Brooks, and G. Y. Wei. 2017. 14.3 A 28nm SoC with a 1.2GHz 568nJ/prediction sparse deep-neural-network engine with >0.1 timing error rate tolerance for IoT applications. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. 242–243. https://doi.org/10.1109/ISSCC.2017.7870351