Neural Network Inference on Mobile SoCs

Siqi Wang, Anuj Pathania, and Tulika Mitra
National University of Singapore

Editor's notes:
Mobile devices are increasingly being used to run machine-learning-based applications. This article provides a quantitative evaluation of machine learning inference capabilities of the different components on mobile SoCs and explores their performance limits.

—Sudeep Pasricha, Colorado State University

The tremendous popularity of neural-network (NN)-based machine learning (ML) applications in recent years has been fueled partly by the increased capability of the compute engines, in particular, the graphics processing units (GPUs). Traditionally, both the network training and inference were performed on the cloud with mobile devices only acting as user interfaces. However, enriched user experience and privacy concerns now demand inference to be performed on mobile devices themselves with high accuracy and throughput.

In this article, we look at NN-enabled vision applications on mobile devices. These applications extract high-level semantic information from real-time video streams and predominately use convolutional NNs (CNNs). They are important in many domains, such as advanced driver-assistance systems (ADASs), virtual reality (VR), and augmented reality (AR). Enabling these applications in power-constrained mobile devices is challenging owing to enormous computational and memory requirements.

Heterogeneous multiprocessor SoC enables the current state-of-the-art mobile devices. However, the presence of multiple vendors fragments the mobile SoCs. Accelerators (including GPU, FPGA, and dedicated neural accelerators) demonstrate great performance for inference. However, these high-performance components are present in only a small fraction of mobile devices. Morover, owing to market fragmentation, it is impossible to develop a mobile application with accelerators that can run across multiple devices. Instead, central processing units (CPUs) remain the common denominator among mobile SoCs and are a favored choice for inference [1].

Digital Object Identifier 10.1109/MDAT.2020.2968258
Date of publication: 20 January 2020; date of current version: 7 October 2020.

We embark on an exploration to quantitatively characterize and understand the inferencing capabilities of mobile SoCs given the diverse landscape. We portray the power–performance gap between the ubiquitous CPUs and high-performance accelerators in high-end devices and uncover the reasons behind the gap through the roofline models. Finally, we propose simultaneous engagement of all SoC components to greatly expand the promise of functional deployment of vision applications on mobile devices.

Inference on mobile SoCs

Heterogeneous multiprocessor SoCs

There are over 2,000 unique mobile SoCs in the mobile devices market. The diversity comes from the choice of different CPUs, GPUs, caches, memory controllers, and other application-specific accelerators. This fragmentation of the SoC market makes standard optimizations impossible. However, the similarity among these SoCs lies in the choice of one or more CPU core clusters.

1) ARM big.LITTLE: Multicores enable the state-of-the-art mobile SoCs. In 2019, 99.9% of the Android devices in the market had multiple cores [1]. Among these, about half of the SoCs implemented performance heterogeneity with at least two CPU clusters: a high-performance and an energy-efficient core cluster. ARM big. LITTLE architecture, one of the most popular architectures implementing this heterogeneity, is present in Hi-Silicon *Kirin*, Samsung *Exynos*, and Qualcomm Snapdragon series SoCs. Heterogeneous cores differ in power–performance–area characteristics but share the same instruction set architecture (ISA). Figure 1 shows an abstract block diagram of this architecture. The general availability of CPUs makes them a favorable choice for mobile inference and make device-agnostic optimizations feasible.

2) Accelerators: Existing architectures, including GPU and FPGA, have proven to be advantageous for ML workloads and are thus commonly used for deployment on certain devices. Both academic and commercial dedicated accelerators [Google Edge tensor processing unit (TPU), Intel Nervana neural network processor (NNP), Huawei network processing unit (NPU), Apple Neural Engine] offer exceptional runtime and energy efficiency. There are no standard neural accelerators for mobile SoCs, making horizontal application integration difficult. Limited availability even constraints the use of GPUs.

Mobile ML framework and optimizations

Tensorflow, PyTorch, and *MXNet* are some of the common ML development frameworks for all scenarios. *Tensorflow* Lite-like frameworks facilitate the compression of huge models to fit into resource-constrained mobile devices. Efficient libraries and application programming interfaces (APIs) bridge the gap between the frameworks and the underlying hardware, examples of which are Nvidia *cuDNN* for GPUs, ARM NN powered by Compute Library (*ARM-CL*) for ARM CPUs and GPUs, Facebook *NNPACK*, and *QNNPACK* for mobile CPUs. These libraries usually optimize with detailed architectural information. *ARM-CL* supports acceleration through *ARM* NEON vectorization and provides NEON assembly implementation for the most computationally intensive convolution kernels. Algorithmic optimizations [Winograd transform, fast Fourier transform (FFT), sparsity exploration] lower the computational complexity of convolution computations. Furthermore, quantization and network pruning are common techniques that bring down the processing requirement with the sacrifice of accuracy [2].

Even though most mobile inference workloads run on CPUs, optimization of ML workloads with accelerators hordes most of the attention. There is a lot of room for optimization of mobile CPUs to enable ML applications across different mobile platforms.

Characterizing inferencing on mobile SoC

We perform experiments across different technology nodes using two commonly used mobile SoCs: 28-nm *Exynos 5422* within the Odroid XU3 development platform and 10-nm *Kirin 970* within the *Hikey 970* development platform. Released in 2014 and 2017 respectively, these two SoCs show us the development of mobile SoCs over the years. Furthermore, these two SoCs roughly approximate the mid- and high-end mobile SoCs today.

In the experiments, both SoCs use *ARM-CL 18.05v*. *Kirin 970* NPU is supported by HiAI DDK (v100) for network deployment. For *Exynos 5422*, in-built power sensors, running at 200 Hz, measure the power of each component. For *Kirin 970*, because of the absence of any integrated on-chip power sensors, we approximate the power consumption by measuring the socket power with the help of a power measurement unit [3] running at 100 Hz.

Experimental setup

1) CPU: Both SoCs include the ARM big. LITTLE-based asymmetric multicore CPU. The *Kirin 970* CPU adopts the ARMv8-A architecture. It consists of a high-performance high-power out-of-order four-core *Cortex-A73* cluster (2.36 GHz) and a low-performance low-power four-core in-order *Cortex-A53* (1.8 GHz). *Exynos 5422* has a similar design but uses an older ARMv7-A architecture with *Cortex-A15* (2 GHz) and

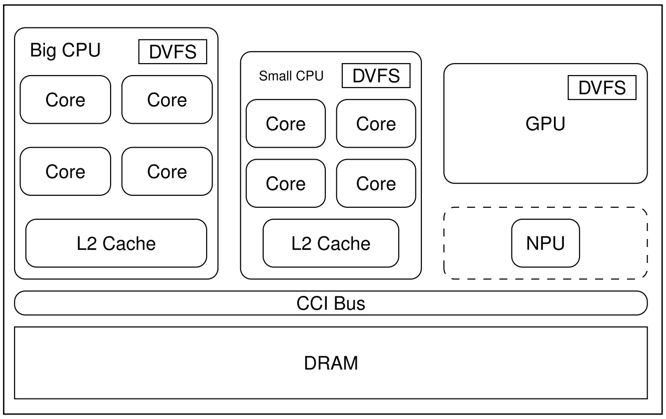

Figure 1. An abstract block diagram of a mobile SoC with an asymmetric multicore CPU, GPU, and NPU.

Table 1. Throughput of different networks on different mobile SoCs components running at their peak frequencies.

Network	*Exynos 5422* Throughput (Imgs/s)			*Kirin 970* Throughput (Imgs/s)			
	A7	*A15*	*T628*	*A53*	*A73*	*G72*	NPU
AlexNet	1.1	3.1	7.8	2.2	7.6	32.5	32.5
GoogLeNet	0.9	3.4	5.2	3.0	7.1	19.9	34.4
MobileNet	1.5	5.7	8.5	6.5	17.7	29.1	*Not Supported*
ResNet50	0.2	1.3	2.1	1.5	2.8	8.4	21.9
SqueezeNet	1.5	5.0	8.0	6.8	15.7	43.0	49.3

Cortex-A7 (1.4 GHz) cores. All CPU cores support NEON advanced single instruction multiple data (SIMD) operations, which allows for four 32-bit floating-point operations per cycle.

2) GPU: *Kirin 970* adopts the *ARM Mali G72 MP12 GPU* (850 MHz), implementing the second-generation *Bifrost* architecture. It has 12 shader cores with three execution engines each. Each engine is capable of eight FP32 operations per cycle, giving a total peak compute capability of 244.8 GFLOPS/s for *G72*. *Exynos 5422* includes an *ARM Mali T628 MP6* GPU (600 MHz). It adopts an older *Midgard* architecture with six shader cores implementing *Tripipe* design with two arithmetic pipelines. Each pipeline is capable of performing eight FP32 operations per cycle, providing a total peak compute capability of 57.6 GFLOPS/s for *T628*.

3) NPU: *Kirin 970* includes a *Huawei* NPU purpose-built for ML. It has a peak performance of 1.92 TFLOPS/s with FP16. The accompanying *HiAi DDK* API enables the deployment of networks on NPU but only works with *Android*. *Exynos 5422* does not have any ML accelerator.

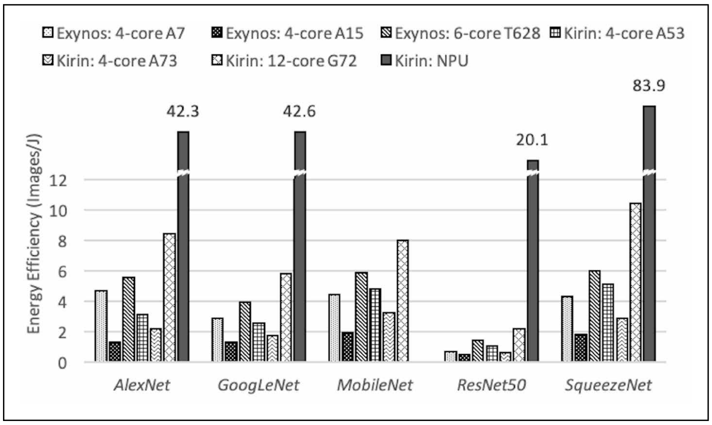

Figure 2. Energy efficiency of different components while running at their peak frequencies.

4) Network Structure: We experiment with several popular networks introduced in recent years—*AlexNet* [4], *GoogleNet* [5], *MobileNet* [6], *ResNet50* [7], and *SqueezeNet* [8].

Individual heterogeneous components

We first study each component in isolation by running inferencing of multiple images in a stream on a single component. Both *Big* and *Small* clusters are self-sufficient for inferencing. GPU and NPU require the support of a *Small* cluster for inferencing.

1) Throughput: Table 1 shows the throughput of each component on both our SoCs. All components in *Kirin 970* outperform their respective counterparts in older *Exynos 5422*. *Big A73* cluster, *Small A53* cluster, and *G72* GPU outperform *Big A15* cluster, *Small A7 cluster*, and *T628* GPU on average by a factor of 4.4×, 2.6×, and 4.2×, respectively. The performance gap between the *Big* and *Small* clusters has reduced from 4× to 2.5× with a decrease in *Big* to *Small* power consumption ratio from 10× to 4×. Furthermore, the performance gap between GPU and CPU clusters is only about 2× to 3× for both SoCs.

For NPU, we were unable to deploy *MobileNet* due to incompatible operators. On average, NPU is only 1.6× better than the high-end *G72* GPU. On the other hand, the portability of applications across different platforms remains a challenge for dedicated accelerators. The proprietary development kit makes the general optimization a difficult endeavor.

2) Energy efficiency: We measure the average active power consumption of inferencing on different components and calculate the energy efficiency, as shown in Figure 2. For *Exynos 5422*, power sensors for individual components measure the power consumption of each component separately. For *Kirin 970*, we calculate active power values by subtracting the idle power (measured when no workload is running) from socket power measurement taken during inferencing. Therefore, the power measurements for *Kirin* are slightly higher, as memory power cannot be separated.

NPU is the most energy-efficient among all components, which we expect, given its custom design for inference. GPUs are the second-most energy-efficient component. *Small* clusters also show good energy efficiency. However, Table 1 shows that their performance in terms of absolute throughput is too low to be ever useful alone.

Comparing across two platforms, the energy efficiency of each component has improved for the newer SoC. However, the improvement is minimal and even negative for the *Small* CPU cluster. Compared to its predecessor *A7*, *A53* is more complex and area hungry with 64-bit, complex branch prediction, and larger translation lookaside buffer (TLB). It achieves greater performance but at the cost of even greater power consumption.

3) Impact of technology scaling versus architectural innovations: *Exynos 5422* and *Kirin 970* use the 28- and 10-nm technology nodes, respectively. In moving from 28-nm *Exynos 5422* to 10-nm *Kirin 970*, the maximum frequency of the *Big* cluster has only changed from 2 GHz (*A15*) to 2.36 GHz (*A73*), whereas the *Small* cluster changes from 1.4 GHz (*A7*) to 1.8 GHz (*A53*). So the frequency scaling is 1.18× for the *Big* cluster and 1.29× for the *Small* cluster for these two platforms. On the other hand, we get 4.4× and 2.6× throughput improvement across technology generations (Table 1) for the *Big* cluster and *Small* cluster, respectively. This improvement in performance is achieved through smart designs such as microarchitectural improvements (improved branch predictor, cache data prefetchers, etc.), larger caches, and 64-bit support, leading to improved NEON processing, among others.

However, in the case of the *Small* cluster, with an increased area, the microarchitectural changes give an increase in power that cannot be offset by technology scaling. Indeed, the *Small A53* cluster consumes roughly twice the power of the *Small A7* cluster. Thus, the energy-efficiency improvement is limited for the *Small* cluster for some networks as we move from *A7* to *A53*. In contrast, between the two *Big* clusters, *A73* is more power-efficient than *A15*; the energy efficiency improves from *A15* to *A73* cluster. As mentioned earlier, the power measurements for *A7* and *A15* are quite accurate, while the measured power for *A53* and *A73* are higher as it includes the memory power that could not be separated.

4) Insights: We observe that NPU provides unmatched energy efficiency for inferences. It is the optimal choice to perform network inferences on the platforms with such dedicated accelerators. However, a developer needs to put in substantial effort to port their application with proprietary API to execute on NPU, and the effort would not bear any fruits on mobile devices lacking this very-specific NPU. NPU, as a black-box, also causes inflexibility in development and optimizations. Furthermore, NPU is compatible with only a limited set of network designs. These extra requirements could make it quickly obsolete for future networks.

On the other hand, high-end GPUs can provide performance comparable to NPU at satisfactory energy efficiency. GPUs are capable of running general-purpose (GPGPU) applications written in *OpenCL*, which is easily portable to a large variety of GPUs and even CPUs supporting *OpenCL*. This generality makes it a good candidate to use when high performance is a major consideration.

CPUs provide both the worst energy efficiency and the worst throughput among all components. Still, they are critical for inferencing because they are commonly present across all mobile devices. Low-end mobile SoCs would lack accelerators like NPU. They may contain a low-end GPU, but maybe missing *OpenCL* support, thereby lacking any inferencing capability. Network inference on the CPU is inevitable and demands optimization considerations.

Our analysis shows that any component alone on both platforms can barely support the increasing performance requirement for network inferencing. The "Coexecution of multiple components" section presents the coexecution methodology that can mitigate the performance issue to some extent. Still, we must continue to look into the networks themselves in search of further optimization opportunities.

Roofline analysis

To understand the execution behaviors of the networks on each SoC components, we perform a roofline analysis.

Roofline analysis [9] is a widely applied methodology that can classify an application as memory- or compute-bound on a given hardware. It gives insights into developers for improving their application design to cater to the computation and memory capability of the underlying processing devices. The horizontal "Ceiling" and the "Roof" construct a "Roofline" that bounds the maximum performance of an application (measured in GOPS/s) under a hardware-determined compute- or memory-bound, respectively. Operational intensity (OI) of application (measured in FLOPS/byte) determines whether its peak performance is bounded by the memory bandwidth (measured in GB/s) or compute capability (measured in GOP/s) of the hardware. Both *Exynos 5422* and *Kirin 970* show similar behavior for the CPU core clusters and GPU. Therefore, we only present here the analysis for *Exynos 5422*.

Construction of a roofline model

Hardware specifications provide the peak pure compute performance. Microbenchmarking [10] provides the peak (sustainable) memory bandwidth. Specifications claim the peak memory bandwidth of the memory bus to be 14.9 GB/s. However, we observe the actual component-wise peak bandwidth to be 3.44, 0.49, and 6.15 GB/s for *A15* cluster, *A7* cluster, and *T628* GPU, respectively.

Many variations of the roofline model are constructed to adapt to different use cases. In this analysis, we defined two operational intensities that are theoretical OI (OI_t) and empirical OI (OI_e) defined in the following equations:

$$\mathrm{OI}_t = \mathrm{GOPS/Mem\_Access} \tag{1}$$

$$\mathrm{OI}_e = \mathrm{GOPS/DRAM\_Access} \tag{2}$$

We calculate OI_t by analyzing the code.The memory accesses include all the data required in the computation. During actual executions, multiple levels of caches within components improve the memory access performance. Caches make it difficult for OI_t to correlate with the actual performance on the components.Therefore, we introduce empirical OI (OI_e). We calculate OI_e using the actual DRAM accesses on the bus, which models the presence of multilevel memory hierarchy. It is more informative and has a better correlation with the actual performance on the component than OI_t. We use application-specific performance counters obtained from *ARM Streamline DS5* at run time for the calculation of OI_e (CPU: *L2_data_refill*, GPU: *Mali L2 cache external read/write bytes*). Figure 3a shows the roofline points of major layers in *AlexNet* on the *A15* cluster for both OI_t and OI_e.

Theoretical and empirical OI

Figure 3a plots the OI_t (squares) and OI_e (diamonds) values of several *AlexNet* major layers, marked with different colors. Black marks the whole network OI_t and OI_e of *AlexNet*. The intersection points of the OI_t values with the "Roofline" represent the theoretical maximum performance for the code-based theoretical operational intensities, which fall in the memory-bound region on the "Roof." The corresponding points for OI_e are actual achieved performance in GOPS/s, which are always below the "Roofline."

The presence of cache reduces the memory accesses going to the DRAM during execution, and thus increases the OI. Therefore, for all layers, OI_e points are on the right of OI_t points, indicating higher performance. For layers with low OI_t [fully connected (FC)], the points move along the "Roofline," achieving the theoretical maximum performance. For layers with higher OI_t [convolutional (CONV)], the points cross the boundary of memory-bound and become compute-bound. The performance gain is not as significant, and we explain this with the underutilization due to insufficient or imperfect parallelization. Overall, OI_e is a better indicator of real-world performance. Therefore, we only plot values of OI_e going forward.

Across different components

Figure 3b shows the performance of different networks on different components on *Exynos 5422*. The color of the points corresponds to the respective component. We can observe that memory severely bottlenecks the performance of both the *A7* cluster and *T628* GPU. The performance of the *A15* cluster falls in both compute- and memory-bound regions depending upon the network.

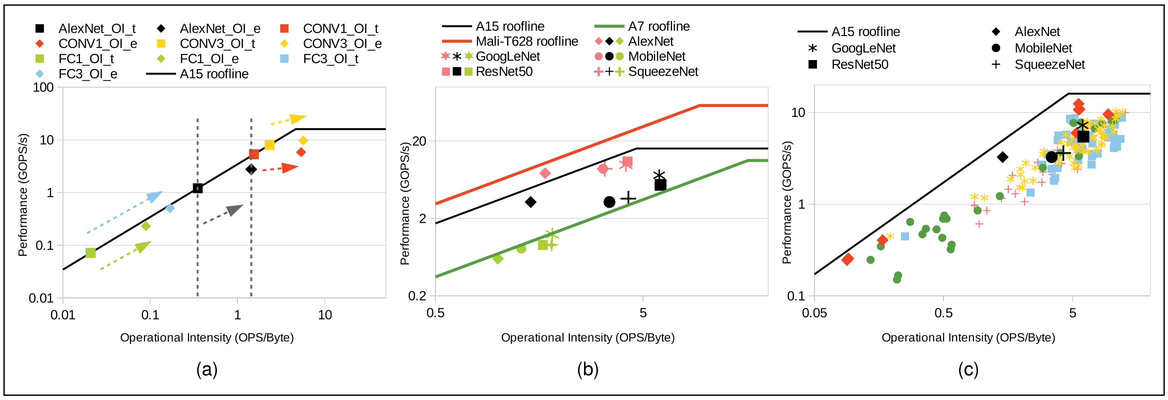

Figure 3. Roofline plot for inference workloads and major layer information on multiple processors in *Exynos 5422*.

The OI_e values are different because of the different memory hierarchies for different components. The *Big* core cluster with a larger cache size (L2: 2 MB) derives higher benefits from memory hierarchy than GPU (L2: 128 KB). However, *AlexNet* that is notorious for huge parameter sizes caches will get flushed regardless of the cache sizes, resulting in a smaller benefit from the memory hierarchy. On the other hand, small filter sizes lead to suboptimal parallelization (underutilization). This observation holds more starkly for newer networks with smaller filter size than older networks. The observation explains the significant deviation in the empirical performance of networks on the components from the "Roofline."

Major layers in inference

We do a deeper layer-level analysis to explain the behavior of the networks. Both convolutional and FC layers dominate the total execution time of networks, and thus both are considered as major layers worthy of examination. We limit our analysis to the *Big* cluster because networks there show both memory- and compute-bound behavior. Figure 3c shows that different layers in *AlexNet* (and also other networks to a lesser extent) exhibit different empirical OIs. Convolutional layers at the start of *AlexNet* perform compute-intensive convolution on large inputs, thereby having relatively higher OIs. On the other hand, FC layers perform memory-intensive operations on large-size parameters, thereby having relatively lower OIs. Convolutional and FC layers of *AlexNet* fall in the compute- and memory-bound regions of the roofline model, respectively. Overall, *AlexNet* falls somewhere in the middle of both. In general, we observe that layers of a network are scattered in both compute- and memory-bound regions. This difference comes from the choice of the size of the input tensors and filters. Vast differences in OI_e for different layers within a network motivate layer-level optimizations such as per-layer dynamic voltage and frequency scaling (DVFS) for power management. Furthermore, the variation within a network motivates fine-grain layer-level co-executions, which improve the overall chip utilization [11].

Effect of quantization

Quantization is a commonly applied technique that reduces the memory and computation requirements of a network while reducing accuracy. However, the quality of its implementation primarily determines the benefits it provides. In the implementation of quantized MobileNet in *ARM-CL* (18.05v), the QASYMM8 model with 8-bit weights is used. This implementation fails to improve the overall performance of the network. Deeper analysis reveals that the latencies of convolutional layers are indeed reduced, but the overheads from extensive dequantization and requantization overshadow any benefit.

Quantization reduces the total operations and memory access required near proportionally. Reduction in memory accesses results in a slightly higher empirical OI (OI_e). Therefore, the roofline analysis of a quantized network nearly overlaps with that of its nonquantized counterpart, and quantization does not improve the memory behavior of the layers. Lower operation requirements under quantization predominately contribute to the reduction in the execution time of the convolutional layers.

Glimpse of NPU

NPU, due to its novelty and dedicated ML processing design, garners a lot of attention. However, most of the details are kept confidential. We are unaware of its architectural and integration details. Therefore, we can only attempt to reverse engineer its behavior to gain some insights.

We implement a kernel module that enables counting of traffic on the cache coherent interconnect (CCI) bus. We attribute the traffic on the CCI bus that goes to DRAM during the engagement of NPU to the main memory activity of NPU. The maximum observed memory bandwidth of executing several networks and the peak performance of 1.92 TOPS

Table 2. Throughput improvement on *Exynos 5422* and *Hikey 970* by coexecution over the best throughput with a single component (*T628* and *G72* GPU).

Network	*Exynos 5422* Throughput (Imgs/s)			*Kirin 970* Throughput (Imgs/s)		
	T628	Co-execution	Gain	*G72*	Co-execution	Gain
AlexNet	7.8	10.3	32.4%	32.5	33.4	2.8%
GoogLeNet	5.2	8.7	66.3%	19.9	28.4	42.8%
MobileNet	8.5	14.9	76.7%	29.1	51.5	77.1%
ResNet50	2.1	2.9	38.6%	8.4	12.3	46.3%
SqueezeNet	8.0	13.8	73.9%	43.0	54.5	26.7%

Table 3. Throughput improvement on *Kirin 970* by coexecution over the best throughput with a single component (NPU).

Network	Throughput (Images/s)		Gain (%)	Image Frames Composition (%)			
	NPU	Co-execution		*A73*	*A53*	*G72*	NPU
AlexNet	32.5	63.7	96.0	1.90	0.95	47.47	49.68
GoogleNet	34.4	59.3	72.4	3.06	1.70	33.33	61.90
ResNet50	21.9	30.9	40.9	2.63	1.32	26.97	69.08
SqueezeNet	49.3	95.1	92.9	3.18	1.69	43.43	51.69

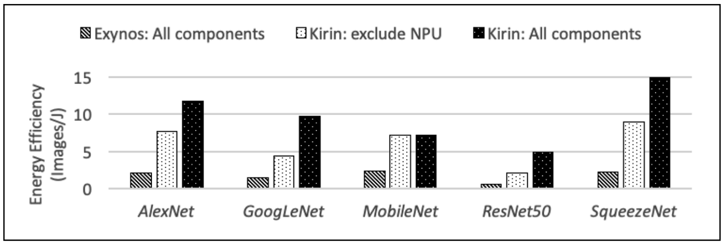

Figure 4. Energy efficiency of coexecution on *Exynos 5422* with all components, on *Kirin 970* with CPU and GPU (excluding NPU) and all components (including NPU).

from the specification construct the “Roof” and “Ceiling” of the NPU roofline. We observe that the performance of NPU is significantly bounded by the memory for the networks tested. This observation shows a significant scope for optimization to achieve the full processing potential of NPU.

Improving the performance

Coexecution of multiple components

Stream processing, depending on the application, requires 10–40-images/seconds throughput. Some applications even require multiple inferences to run at the same time. Table 1 shows that the high-end *Kirin 970* SoC can barely sustain such requirement, whereas the midend *Exynos 5422* cannot. We previously observed that the peak bandwidth consumed by any individual component is far below the total bandwidth supported by the bus. This observation supports the claim that inferencing through multiple components together will not make individual components more memory-constrained than their isolated inferencing. Therefore, we use *ARM-CL* to create an infrastructure, wherein multiple components process images from a single unified stream in parallel using a work-stealing mechanism. The infrastructure uses a buffer to reorder the out-of-sync output from different components. Coexecution obtains significantly higher throughput than the highest throughput component in isolated execution.

Table 2 shows the peak coexecution throughput on both mobile SoCs with the *ARM big.LITTLE* CPU core cluster and GPU. We include the best individual component executions, which are GPU for both platforms, for comparison. On average, the coexecution gives 50% throughput improvement over GPU-only execution. Furthermore, Table 2 shows *Exynos 5422’s* obsolescence. Even with the coexecution, *Exynos 5422* shows very low absolute throughput.

Coexecution with NPU

The performance of NPU is unbeatable. Table 3 shows that *Kirin 970*, with the coexecution of all on-chip components, gives exceptionally high throughput. In practice, we can execute NPU and GPU in parallel toward one application that demands very high performance or to perform multiple inferences simultaneously with multiple applications.

Coexecution energy efficiency

Synergistic coexecution engages multiple components simultaneously to improve performance at the cost of higher power consumption. Therefore, the energy efficiency of the coexecution is the average energy efficiency of engaged components. Figure 4 shows the energy efficiency of the execution that engages all the components on *Exynos 5422*, the CPU clusters and GPU on *Kirin 970* (exclude NPU), and all the components on *Kirin 970* (include NPU). Overall, the coexecution energy efficiency is always better than the *Big* CPU cluster. In *Kirin 970* SoC, as the GPU is much more energy-efficient than the CPU clusters, the coexecution provides better energy efficiency than the power-efficient *Small* CPU cluster.

Mobile inferencing is now ubiquitous. In this work, we examine the power–performance characteristics of inferencing through several prominent NNs on different components available within a mobile SoC. We also perform roofline analysis of networks on components to unveil the further optimization scope. We show that the network throughput can increase by up to 2× using coexecution that engages all the components in inferencing simultaneously. ■

Acknowledgement

This work was supported in part by the Singapore Ministry of Education Academic Research Fund TI 251RES1905.

References

[1] C.-J. Wu et al., “Machine learning at Facebook: Understanding inference at the edge,” in *Proc. IEEE Int. Symp. High Performance Comput. Archit. (HPCA)*, 2019, pp. 331–344.

[2] M. Wess, S. M. P. Dinakarrao, and A. Jantsch, “Weighted quantization-regularization in DNNs for weight memory minimization toward HW implementation,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.,* vol. 37, no. 11, pp. 2929–2939, 2018.

[3] “Keysight Technologies B2900 Series Precision Source/Measure Unit.” Accessed: 5 February 2020. [Online]. Available: https://goo.gl/U4HMbu

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. Adv. Neural Info. Process. Syst.*, 2012, pp. 1097–1105.

[5] C. Szegedy et al., “Going deeper with convolutions,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.,* 2015, pp. 1–9.

[6] A. G. Howard et al., “MobileNets: Efficient convolutional neural networks for mobile vision applications,” 2017, *arXiv preprint:1704.04861*.

[7] K. He et al., “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[8] F. N. Iandola et al., “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 0.5 MB model size,” 2016, *arXiv preprint :1602.07360*.

[9] S. Williams, A. Waterman, and D. Patterson, “Roofline: An insightful visual performance model for floating-point programs and multicore architectures,” Lawrence Berkeley National Lab. (LBNL), Berkeley, CA, USA, Tech. Rep., 2009.

[10] S. Siamashka, “Tinymembench.” Accessed: 5 February 2020. [Online]. Available: https://github.com/ssvb/tinymembench

[11] S. Wang et al., “High-throughput CNN inference on embedded arm big.LITTLE multi-core processors,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.,* 2019. Accessed: 5 February 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8852739

Siqi Wang is currently a Research Assistant and is currently pursuing a PhD at the School of Computing, National University of Singapore, Singapore. Her current research interests include performance optimization, task scheduling, general-purpose GPUs, and deep learning on heterogeneous multiprocessor systems.

Anuj Pathania is currently a Research Fellow at the School of Computing, National University of Singapore, Singapore. His research interests include resource management algorithms with an emphasis on performance-, power- and thermal-efficiency in embedded systems. Pathania has a PhD from the Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany (2018).

Tulika Mitra is a Professor of computer science at the School of Computing, National University of Singapore, Singapore. Her research interests span various aspects of the design automation of embedded real-time systems, cyber-physical systems, and Internet-of-Things. Mitra has a PhD in computer science from the State University of New York at Stony Brook, Stony Brook, NY (2000).

■ Direct questions and comments about this article to Tulika Mitra, Department of Computer Science, School of Computing, National University of Singapore, Singapore 117417; tulika@comp.nus.edu.sg.