ARADA: Adaptive Resource Allocation for Improving Energy Efficiency in Deep Learning Accelerators

Citation for the original published paper (version of record):
Azhar, M., Zouzoula, S., Petersen Moura Trancoso, P. (2023). ARADA: Adaptive Resource Allocation for Improving Energy Efficiency in Deep Learning
Accelerators. Proceedings of the 20th ACM International Conference on Computing Frontiers 2023, CF 2023: 63-72. http://dx.doi.org/10.1145/3587135.3592207

ARADA: Adaptive Resource Allocation for Improving Energy Efficiency in Deep Learning Accelerators

Muhammad Waqar Azhar
waqarm@chalmers.se
Chalmers University of Technology
Göteborg, Sweden

Stavroula Zouzoula
zouzoula@chalmers.se
Chalmers University of Technology
Göteborg, Sweden

Pedro Trancoso
ppedro@chalmers.se
Chalmers University of Technology
Göteborg, Sweden

ABSTRACT

Deep Learning (DL) applications are entering every part of our life given their ability to solve complex problems. Nevertheless, energy efficiency is still a major concern due to the large computational and memory requirements. State-of-the-art accelerators strive to address this issue by optimizing the architecture to the compute requirements of DL algorithms. However, there is always a mismatch between compute and memory requirements and what is offered by a particular design. A way to close this gap is by providing run-time adaptation or resource allocation to improve efficiency.

This paper proposes an adaptive resource allocation for deep learning applications (ARADA) with the goal of improving energy efficiency for deep learning accelerators. This is leveraged by having a layer-by-layer resource allocation. The rationale is that each layer in the DL model has a unique compute and memory bandwidth requirement and allocating fixed resources to all layers leads to inefficiencies. This can be achieved by means of resource allocation (e.g., voltage-frequency, memory bandwidth) to save energy without sacrificing performance. Experimental results show that applying ARADA to the execution of 9 state-of-the-art CNN models results in an energy savings of 38% on average compared to race-to-idle for an Edge TPU coupled with LPDDR4 off-chip memory.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Artificial intelligence**; • **Hardware** → **Power and energy**; • **Computer systems organization** → *Embedded systems*; **Systolic arrays**; Multicore architectures; **Neural networks**; *Heterogeneous (hybrid) systems.*

KEYWORDS

CNNs, Energy Efficiency, Resource Allocation, Accelerators

ACM Reference Format:
Muhammad Waqar Azhar, Stavroula Zouzoula, and Pedro Trancoso. 2023. ARADA: Adaptive Resource Allocation for Improving Energy Efficiency in Deep Learning Accelerators. In *20th ACM International Conference on Computing Frontiers (CF '23), May 9–11, 2023, Bologna, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3587135.3592207

CF '23, May 9–11, 2023, Bologna, Italy
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0140-5/23/05.
https://doi.org/10.1145/3587135.3592207

1 INTRODUCTION

Deep Learning (DL) applications are entering every aspect of society, given their popularity in solving complex problems. However, these applications pose high computational and memory requirements at both ends of the compute continuum - edge to the cloud. In this context, improving the energy efficiency of the underlined computing systems is a major concern. One approach to address this is to design dedicated domain-specific accelerators.

Several DL accelerators have emerged as both academic research and commercial products [5, 10, 26]. The most efficient accelerator designs are those where the architecture is custom designed to meet the requirements of several DL models or algorithms. These accelerators are typically designed as application-specific integrated circuits (ASIC) and would only be able to solve underlined models. Consequently, they would have limited application and could not evolve to new or different models. Alternatively, FPGA-based accelerator designs for a single model that can change over time could help future proofing but are costly [4, 7].

Another popular approach is providing dedicated hardware modules for accelerating one or more of the dominant computational kernels, such as the general matrix-matrix multiplication (GEMM) and general matrix-vector multiplication (GEMV) operations. These hardware modules are included in most of the current DL accelerators, giving them the capability of accelerating current and future models of different characteristics. Examples of such architectures include Google's Tensor Processing Units (TPUs) [12], Graphics Processing Units (GPUs) [10], NVIDIA's NVDLA [26] and several other Neural Processing Units (NPUs) integrated with general-purpose CPUs (e.g., Apple M1 [22]). At the core of most of these architectures are several Multiply-and-Accumulate (MAC) (often termed as Processing Elements - PEs) units grouped together in the required form. For example, GEMM can be effectively accelerated by a systolic array (SA) of MAC units. SAs can also be used to accelerate other kernels, such as convolution, by applying transformations such as image to column (im2col), thus enhancing their applicability [6]. Obviously, this flexibility comes with a tradeoff in terms of efficiency for these more generic domain accelerators. This paper addresses this efficiency gap by improving the resource allocation at a finer grain during the execution of the DL algorithms.

DL models are composed of several layers of various sizes and types (e.g., depth-wise convolution, point-wise convolution, fully connected), resulting in a varying degree of computation and memory requirements over the course of the application's execution. The most common case, when using the domain-specific accelerators mentioned above, is for the execution to be performed layer-by-layer due to data dependency between layers. As such, the accelerator processes all operations for the first layer and then

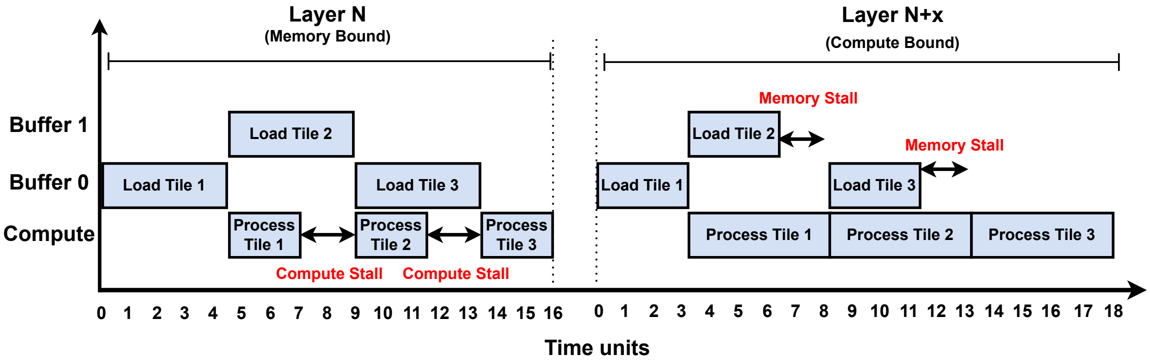


Figure 1: A scenario depicting interleaved loading and computation and inherent stalls.

proceeds to the second layer, and so forth. Furthermore, due to the limited computational units and internal buffer space, the execution granularity is further refined to smaller partitions of the data into what is known as tiles. Such an execution pattern leads to having two clear phases: data loading and computation. Whenever possible, in order to hide the latency of data loading, double or ping-pong buffers are used to overlap execution with the loading of the data as shown in Figure 1. Nevertheless, given the diverse characteristics of the different layers of a model, the data loading and computation do not necessarily take the same time. For example, in Figure 1, we depict the execution of two layers of a DL model when using double buffering and observe that for the first layer, the computation is smaller than the loading of the data for the next layer. In contrast, for the second layer, the data loading for the third layer is larger than the computation. This clearly illustrates the mismatch between computation and memory. Thus, fine-grain (e.g., layer-by-layer) resource allocation can improve overall efficiency.

The impact of resource allocation (RA), (i.e., voltage-frequency (VF), power gating, batch size, PE count) on the performance and energy efficiency of DL application's execution is extensively studied [27, 33], where the focus is mainly on finding the best RA setting for the complete execution of a specific DL model on hardware (CPU and GPU). Jiang et al. [17] have proposed the use of a power-down state using Dynamic Voltage Frequency Scaling (DVFS) to save energy while in an idle period after processing an input image and waiting for the following image on the FPGA. Jiang et al. [16] propose an accelerator design and VFS policy similar to power gating (PG). Liu et al. [21] proposed a mechanism based on sparsity detection to control individual PEs by employing DVFS and power gating to save energy. Nabavinejad et al. [24] have proposed using batch size to enhance the DVFS capability and improve performance under a power cap. Similarly, Yao et al. [35] proposes a model for optimal batch size and frequency setting. Morteza et al. [23] proposes a method for precision control and DVFS to reduce power under QoS of response time in server-based CNN inference. Yu et al. [36] proposes a heuristic to manage the frequency of the inference server based on requests in the queue to satisfy response time.

There are two main shortcomings in the above-mentioned works. First, the proposed techniques try to find the optimal setting for the complete model, failing to exploit the opportunity offered by inter- and intra-layer type variations. Second, these works assume a DL model and underlined hardware as black-box and don't provide insights into the interplay of parameters (e.g., layer size and type) and hardware architectural parameters (e.g., on-chip memory size and bandwidth). In particular, the impact of a particular RA knob depends if a particular layer is memory or compute-bound. Resource provisioning can be reduced (or increased in the case of compute-bound execution) depending on the relative distance from the roofline performance [15, 34].

In this context, this paper argues to analyze each model at the layer granularity and proposes resource allocation for each layer with the objective of improving energy efficiency. This paper specifically considers voltage-frequency (V-F) scaling and memory bandwidth as resources to improve efficiency but the technique can be applied to cases where new knobs, e.g., PE count, etc are controllable. We evaluated the proposed scheme using SCALE-Sim [30], a DL accelerator simulator, and proposed an implementation framework for a few commodity hardware platforms. We have evaluated ARADA for two design points, namely Edge TPU and HPC TPU, for nine state-of-the-art CNN models of different sizes and characteristics. Simulations have shown that we improve energy efficiency by 38% (for Edge TPU coupled with LPDDR4) and 26% (for HPC TPU coupled with DDR5-4800) on average over race-to-idle (i.e., fixed maximum allocation of compute resources and memory bandwidth) without any performance degradation. Moreover, we show a memory bandwidth reduction potential of 6.5% for Edge TPU coupled with DDR5-4800 and 9% for HPC TPU coupled with HBM2 that can translate into additional energy savings by applying DVFS on memory.

The contributions of this paper are as follows:

- An analysis of computational and memory bandwidth requirements of state-of-the-art CNNs to establish the inter- and intra-layer variability
- An adaptive resource allocation technique *ARADA* applicable to a variety of hardware platforms
- A proof of concept implementation of *ARADA* for an SA-based accelerator architecture and evaluation of energy efficiency and bandwidth reduction

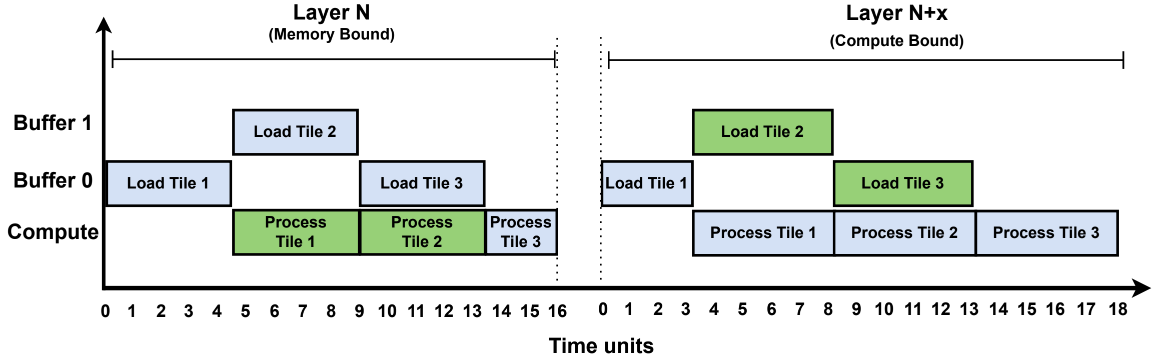


Figure 2: A depiction of computation and memory slowdown to exploit the energy saving opportunities

2 BACKGROUND AND MOTIVATION

This paper targets SA-based accelerators executing inference of DL models. However, the insights can be applied to other accelerator engines, such as GPUs and vector accelerators. As discussed earlier, the execution pattern in domain-specific accelerators with fixed sizes consists of data loading and computation phases, as shown earlier in Figure 1. Note that this simple depiction is only for illustration purposes; in reality, the loading of different components, such as weights, input feature MAP (IFMAP), and output feature MAP (OFMAP), differ significantly and are complex. Typically, the data loading and computations are interleaved, parallel, or combined. The exact behavior depends on the available on-chip buffer and data size.

2.1 Execution Model and Approach

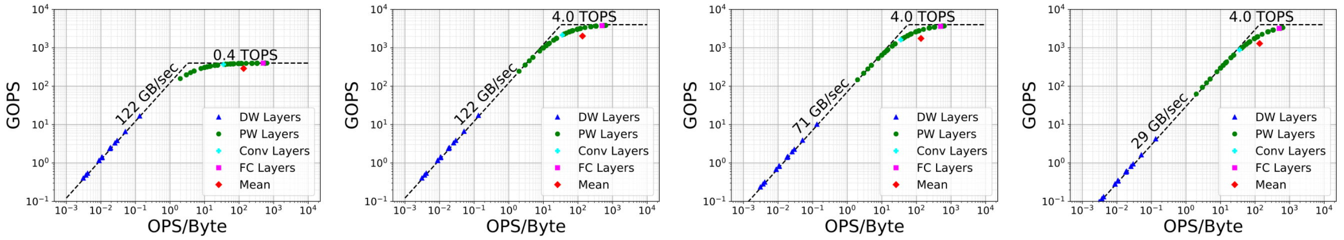
Given the diverse characteristics of the various layers in a model, there exists a mismatch between the data loading and computation, resulting in inefficiency. One key point here is that the interplay of relative speeds of computation and data loading determines the compute or memory boundedness of a specific DNN layer. Since different DNN layers perform different types of computations (e.g., point-wise convolution (PW), and depth-wise convolution (DW)) and have different sizes, the compute and memory bandwidth requirements change. Therefore, there will be stalls in cases of mismatch between computation and data loading speed. In this context, the stalling component operates faster than required, thus consuming additional energy without providing extra performance. Therefore, we propose to slow down the computation or memory, whichever is running faster, to improve energy efficiency as shown in Figure 2. Here computational and data-loading phases highlighted in "green" indicate slow execution or loading at lower energy. Please note that this slowing down of a few sections of the model’s execution will not affect the overall execution time which remains the same. In short, the goal of this work is to reduce energy while keeping the execution time the same as race-to-idle execution.

2.2 Variability in Computational and Bandwidth Requirement of DNN Layers

In order to apply the layer-wise resource allocation, we need to establish a need based on the variability in execution in different layers. This variability stems from inter- and intra-layer type heterogeneity, layer sizes, and parameters (e.g., strides, padding), and manifests itself in the form of variation in compute-to-memory ratio or arithmetic intensity.

To visualize the variability, we employed a roofline model [34]. We present roofline plots in Figure 3, where the x-axis shows the arithmetic intensity (AI) and the y-axis shows the Giga operations per second (GOPS). The target architecture here is a SA of 64 × 64 MAC units coupled with an on-chip-memory buffer (e.g., SRAM) and an off-chip memory (e.g., DRAM and HBM) similar to the Google TPU [12]. We considered two different accelerator design points: one high-performance cluster (HPC) targeting a high-performance cluster system and a high-efficiency targeting edge system. The HPC system is modeled as 4 Tera operations per second (TOPS) (i.e., same as Google Edge TPU) and 0.4 TOPS, coupled with memory bandwidths of 122GB/sec, 71GB/sec, and 29GB/sec. The reason behind the combination of these TOPS and bandwidth (BW) is to explore the design space close to TPU. In this context, we computed the AI and GOPS using a custom-built analytical model. Here, we assume that the on-chip-memory holds all the weights and a single tile of IFMAP and OFMAP. The roofline plots for SA with 0.4 TOPS and 4 TOPS are shown in Figure 3a and Figure 3b, respectively. Different layer types are presented in different colors and shapes, where DW, PW, convolution (Conv), and fully connected (FC) layers are shown in blue, green, cyan, and magenta colors, respectively. Looking at Figure 3a, one can see the variability between different types of layers as they lie in either compute- or memory-bound regions. Moreover, the degree of compute- and memory-bounded behavior is different among the layers of the same class owing to the size and AI of a layer.

Another important observation is that changes in accelerator architecture, e.g., memory BW and peak compute performance, affect the resource requirements for the layers because the layer can shift from the compute to the memory-bound region and vice versa. For example, when peak performance is changed from 0.4 to 4



(a) 0.4 TOPS - 122 GB/sec

(b) 4 TOPS - 122 GB/sec

(c) 4 TOPS - 71 GB/sec

(d) 4 TOPS - 29 GB/sec

Figure 3: Per layer roofline plot for EfficientNetB0

TOPS, the layers in the compute-bound region shift to the memory-bound region. Similarly, when the BW is reduced from 122 GB/sec to 71 GB/sec and 29 GB/sec, as shown in Figure 3b, Figure 3c and Figure 3d respectively, the layer in a compute-bound move to the memory-bound region. Moreover, the extent of memory or compute boundedness also changes with changing the above-mentioned accelerator design parameters. These changes in memory BW can be a design time consideration or run-time effect because of BW sharing when co-executing applications. In short, it is important to analyze the DL models in the context of the architecture and run-time scenarios, and resources must be allocated accordingly to improve efficiency.

3 ARADA: ADAPTIVE RESOURCE ALLOCATION

3.1 Resource Allocation Proposal

The main theme of the *ARADA* proposal is that different layers in a DL network executing on a particular hardware platform have different memory and computing requirements. In order to improve energy savings, the resource manager must make decisions to exploit this variation considering both the DL workload (e.g., the type of layers and sizes) and the hardware platform parameters (e.g., on-chip buffer sizes and speeds, off-chip memory bandwidth).

There are two key takeaways. First, the RA decisions must be taken at the granularity where compute and memory requirements are changing. Second, the RA decision must consider the DL workload and hardware. These RA decisions are implemented at run-time by the resource manager.

In the context of a SA architecture, two main resource allocation decisions are based on two scenarios. In the first scenario, we consider that a layer is *memory-bound*. Then the compute engine processes data faster than it is made available from the memory subsystem (i.e., on-chip memory + off-chip memory). In this case, the computation capability of the engine can be reduced to match the data production rate of the memory system without any performance loss. However, this reduced computational speed can save considerable energy and improve energy efficiency. In this case, energy reduction comes from power reduction. This reduction can be achieved by tuning various available knobs depending on the hardware, for example, V-F scaling, PE count, Streaming Multi-processors (SM) count (in the case of GPUs), etc.

In the second scenario, we consider layers that are *compute-bound*. In this case, the memory subsystem is producing data at a faster rate, and compute engine operates at a slower rate than the data production rate of the memory. Therefore, there are two possible avenues of energy reduction. First, slowing down the memory to match the computing speed and save energy without losing performance. Recent research proposals [8] have shown promising results in saving energy by applying voltage frequency scaling in memories, and modern memory architectures have started to support various operating voltages [31]. The second option is to increase the frequency above the nominal (e.g., turbo-boost) for a short period of time [29], provided the chip is operating within the thermal design power (TDP) limit.

In the context of this paper, as for the first scenario, we improve energy efficiency by matching the speed of compute engine to the speed of data loading. Thus operating at the optimal point for the given performance target (same as race to idle). For the second scenario, we propose to reduce memory bandwidth to match the data consumption rate of compute engine. Again this allows the accelerator to operate at optimal efficiency setting at the given performance. Note that this resource allocation to match the compute and data loading time will not affect performance compared to race to idle.

3.2 System Overview

We envision the resource allocation for energy efficiency to be done at the system level with a hybrid mechanism with offline and online components. Figure 4 depicts the overall system design: the hardware platform, Resource Allocation Analysis (RAA) phase, and run-time system. The RAA takes the DL application and hardware architecture as input and produces a resource allocation schedule.

The run-time system implements the resource allocation decisions, and feedback from the hardware platform is fed back to the run-time resource manager. This helps predict the application’s behavior and assists RM in the next decision-making point, i.e., after every layer.

3.3 Resource Allocation Analysis

The resource allocation analysis uses the DL workload and hardware specifications to generate a resource allocation schedule, which run-time resource managers can use to allocate resources. Such an analysis is only done once for each combination of DL workload and hardware platform. Therefore, at run-time, resource managers

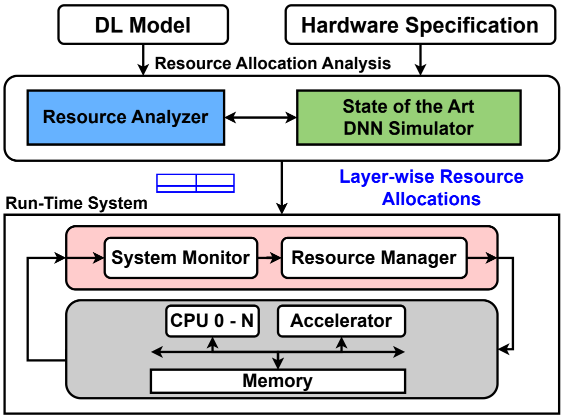


Figure 4: An overview of deep learning resource allocation (ARADA) framework

Algorithm 1 Resource allocation analysis for the ARADA

```
1:  Notations:
2:  CNN : Per layer specification of CNN
3:  ARCH : Compute and memory specification of hardware platform
4:  $F_{max}$ : maximum frequency for compute engine
5:  $V_{max}$ : maximum voltage for compute engine
6:  $BW_{max}$ : maximum bandwidth for off-chip memory
7:
8:  function ARADA_Analysis(CNN,ARCH)
9:      Resource_Allocation ← [ ]
10:     $T_{Total}, T_{Stall} \leftarrow$ Estimate_Compute_Memory_Requirement(CNN, ARCH)
11:     for all layer ∈ CNN do
12:         if ( $T_{Stall}[layer] > 0$ ) then
13:             $[V_{RA}, F_{RA}] \leftarrow$ Compute_Requirement(layer, ARCH, $T_{Total}, T_{Stall}$)
14:             $BW_{RA} \leftarrow BW_{max}$
15:         else
16:             $V_{RA} \leftarrow V_{max}$
17:             $F_{RA} \leftarrow F_{max}$
18:             $BW_{RA} \leftarrow$ Bandwidth_Requirement(layer, ARCH, $T_{Total}, T_{Stall}$)
19:         end if
20:         Resource_Allocation[layer] ← [$F_{RA}, V_{RA}, BW_{RA}, T_{Total}, T_{Stall}$]
21:     end for
22:     return Resource_Allocation
23: end function
```

Table 1: Specification of frequency and bandwidth allocation for MobileNet

Layer	1	2	3	4	5	6	7	8	9	10	11	12	13	14
F (MHz)	500	300	500	250	500	200	500	350	500	300	500	500	500	500
BW (GB/sec)	20	20	20	20	20	20	20	20	20	20	20	20	20	20
Layer	15	16	17	18	19	20	21	22	23	24	25	26	27	
F (MHz)	300	500	300	500	300	500	300	500	300	500	500	500	500	
BW (GB/sec)	20	20	20	20	20	20	20	20	20	20	20	8	20	

only have to implement the decisions resulting in low overheads at run time. In essence, by means of this analysis, we would like to identify the different types of layers and whether a layer is in compute or memory-bound region. To elaborate on the process, we present a pseudo-algorithm in Algorithm 1. First, we estimate the compute and memory BW requirements of all layers in a CNN using Estimate_Compute_Memory_Requirement() (line-10). This step ascertains if a layer is compute-bound or memory-bound. This can be accomplished by any state-of-the-art ML simulator (e.g., SCALE-Sim [30]). Then the algorithm loops over all the layers in the given model and estimates if the layer is compute or memory-bound (lines 11-21) and estimates appropriate resource allocations. In this context, if stall time is greater than zero, then the layer is memory-bound, and if it's less than zero, then the layer is compute-bound. If the layer is memory-bound, then we predict the compute requirements to save energy while keeping the performance unaffected as depicted by Compute_Requirement(). Details of this function are listed in Section 3.3.1. Similarly, in the case of the compute-bound layer, the memory bandwidth can be adjusted to the demand to save energy, i.e., Bandwidth_Requirement(). This analysis is presented in Section 3.3.2. It is important to note that we do not increase the execution latency or decrease the throughput of the DL model.

3.3.1 Compute Speed Prediction. The resource allocation can be estimated in terms of new frequency by using eq. (1), where the F_{RA}, F_{max}, T_{Total} and T_{Stall} are predicted frequency, maximum frequency, total time and stall time respectively. We assumed that voltage and frequency changes are proportional; a similar model can estimate the voltage reduction. Typically, voltage-frequency (V-F) space is quantized, and thus an additional step can be applied to adjust the frequency and voltage to the nearest higher legal value so that the overall execution time is not increased.

$$F_{RA} = F_{max} \times \frac{T_{Total} - T_{Stall}}{T_{Total}} \tag{1}$$

3.3.2 Memory Bandwidth Prediction. The possible reduction in memory bandwidth in the case of a compute-bound layer (i.e., $T_{Stall} = 0$) can be found using eq. (2). Again, the estimate must adjust to the available V-F levels in memory.

$$BW_{RA} = BW_{max} \times \frac{BW_{max} - BW_{required}}{BW_{max}} \tag{2}$$

The estimated voltage frequency and BW reduction estimates for each layer are compiled in a table for run-time use. An example table depicting the first ten layers of the *MobileNet* model [14] for the Google Edge TPU (as described in Section 4.1.1) is shown in Table 1. Here, we used a specified maximum frequency, i.e., 500 MHz, for the Edge TPU. Furthermore, we assumed a memory bandwidth of 20 GB/sec as peak bandwidth. As can be seen, for each layer, a frequency is specified for SA, and the required bandwidth is provided. The runtime can use these values to allocate resources.

4 EXPERIMENTAL SETUP

4.1 Target Architecture

The evaluation of ARADA in this paper focuses on a systolic array (SA) architecture with two design points targeting the two ends of the computing continuum: Edge TPU and HPC TPU. We further investigated each accelerator with a range of appropriate off-chip memory bandwidth specifications.

4.1.1 Edge TPU. In this system, we modeled an architecture similar to Google's Edge TPU [12], details of which are depicted in Table 2. We refer to this as *Edge TPU* in the rest of this paper. We have coupled the *Edge TPU* with five different types of off-chip memory, each with a diverse memory bandwidth range as listed in the table leading to five hardware configurations of *Edge TPU*.

Table 2: Edge TPU [12]

SA size	64 × 64
On-chip Buffer	4 MB
Dataflow	Output Stationary
Clock Frequency	500 MHz
Memory	LPDDR, LPDDR5, DDR4_4000, DDR4_4400, DDR5_4800

4.1.2 HPC TPU. In this system, we modeled an architecture similar to the Google TPUv3 [11], where four matrix multiplication units (MXU) (two per core and a total of two cores) share the on-chip buffer of 32 MB and the available bandwidth. In this context, we model HPC TPU as SA coupled with 8 MB on-chip memory. Complete architectural details that we assumed are depicted in Table 3. We refer to this as *HPC TPU* in the rest of this paper. We evaluated the accelerators with six different types of off-chip memories suitable for high-performance requirements. Overall the bandwidth of memories coupled with HPC TPU is on the higher side compared to the *Edge TPU*. Thus we have six hardware configurations of *HPC TPU*.

Table 3: HPC TPU

SA size	256 × 256
On-chip Buffer	8 MB
Dataflow	Output Stationary
Clock Frequency	940 MHz
Memory	DDR4_4400, DDR5_4800, DDR5_6800, DDR5_7800, HBM1, HBM2

4.2 Workloads

The workloads used in this study are depicted in Table 4 along with layer count. In this study, we only considered the convolution and fully connected layers of a model as they constitute the majority of the computational load. These workloads include CNNs of various types. For example, MobileNet, MobileNetV2, and EfficientNetB0 employ heterogeneous convolutions (having layers of different types), while ResNet-18 and GoogLeNet are homogeneous (i.e., standard convolution layers). Lastly, YOLOv4-tiny, FaceRecognitionID, FasterRCNN, and SpeakerID belong to the object detection, object recognition, and voice recognition categories.

Table 4: DL models used in the evaluation.

DL Model	MobileNet	MobileNetV2	EfficientNetB0	ResNet-18	GoogLeNet
Layer Count	28	54	82	21	54
DL Model	YOLOv4-tiny	FaceRecognitionID	FasterRCNN	SpeakerID	
Layer Count	28	54	82	21	

4.3 Techniques Evaluated

We evaluated ARADA in the following scenarios to provide potential energy savings and to accommodate technological limitations.

*4.3.1 **RTI**: Race to Idle (Baseline) .* In the Race to Idle scheme, the workloads are executed at fixed maximum resource allocation, i.e., maximum V-F setting and maximum memory bandwidth. This scheme serves as the baseline for our evaluation. All energy savings and bandwidth reduction results are presented in comparison with the baseline.

*4.3.2 **ARADA-IDEAL**: Ideal Resource Allocation .* In this case, we assumed a continuous V-F scaling and no overheads induced by V-F switching. This represents the highest achievable energy savings. We refer to this as ARADA-IDEAL.

*4.3.3 **ARADA-VF-OH**: Resource Allocation with DVFS Overhead .* In this scheme, the resource allocation is applied, considering the V-F switching overhead into account and refer it as *ARADA-VF-OH*. Thus the energy savings are less than ideal. We consider a moderate DVFS switching overhead of 10 microseconds [28]. In this context, it is important to note that higher DVFS reduces the exploitation of energy saving and vice versa.

*4.3.4 **ARADA-VF-OH-Q**: Resource Allocation with DVFS Overhead and Quantized V-F.* A continuous V-F scaling is often not possible in existing hardware therefore, we present the energy savings with a quantized space of 50 MHz steps referred to as *ARADA-VF-OH-Q*. The reason is two-fold, first, a typical embedded CPU typically provides frequency at a resolution of 100 MHz [20]. Second, in modern GPUs, the frequency can be changed at a resolution of 15 MHz [10]. Therefore, considering the evolution of technology, we considered moderate value. Voltage is also assumed to scale linearly proportional to the frequency step. Please note that a higher resolution of frequency switching enables more energy savings with ideal resource allocation corresponding to continuous frequency scaling.

4.4 Simulation Methodology

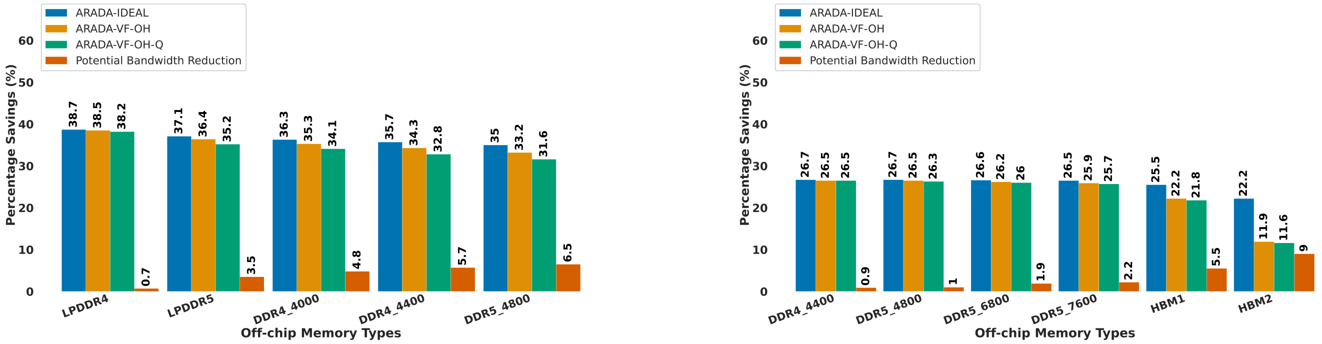
We used SCALE-Sim [30] to estimate compute and memory requirements of each layer for all the workloads while executing them on all hardware configurations. Considering the above-mentioned hardware descriptions, we have a total of 11 hardware configurations. The results are fed into the custom-built simulator, which performs the offline analysis and predicts the resource allocations.

4.4.1 Energy Reduction Estimation. In the scope of this study, we derived an analytical model to estimate the dynamic energy reduction as shown in eq. (6). Dynamic energy for a computational engine can be given by eq. (3), where $\mathrm{E_{dynamic}}$, α, C, V, F, $\mathrm{T_{execution}}$ represent the dynamic energy, activity factor, capacitance, voltage, clock frequency and execution time for a layer.

$$\mathrm{E_{dynamic}} = \alpha \times \mathrm{C} \times \mathrm{V}^2 \times \mathrm{F} \times \mathrm{T_{execution}} \tag{3}$$

$$\mathrm{E_{dynamic(RA)}} = \alpha \times \mathrm{C} \times \mathrm{V_{RA}^2} \times \mathrm{F_{RA}} \times (\mathrm{T_{execution}} + \mathrm{T_{Stall}}) \tag{4}$$

The activity factor (α) (depends on input data) and circuit capacitance (C) are constant for a specific architecture executing the same DNN on two different resource allocations. Moreover, the total execution time of the DNN layer is the same before and after resource allocation. However, the execution time of the computational part increases by the stall time as we slow the compute engine to match the memory load latency. In our proposal, V-F will change with resource allocation resulting in eq. (4) that represents the energy at the particular allocation of voltage and frequency. Dividing eq. (3) and eq. (4) will result in eq. (5) that shows the relationship between the energy before and after RA based on V-F reduction. We can further apply normalization by assuming $\mathrm{E_{dynamic}}$ as unity to get



(a) EdgeTPU : 64 × 64 SA with 4MB on-chip buffer

(b) HPC TPU : 256 × 256 SA with 8MB on-chip buffer

Figure 5: Potential Energy Savings and BW Reduction for Edge and HPC TPU architectures

eq. (6), where energy reduction can be estimated without the need for absolute energy values.

$$\frac{\mathrm{E}_{\mathrm{dynamic(RA)}}}{\mathrm{E}_{\mathrm{dynamic}}} = (\frac{\mathrm{V}_{\mathrm{RA}}}{\mathrm{V}})^2 \times \frac{\mathrm{F}_{\mathrm{RA}}}{\mathrm{F}} \times \frac{(\mathrm{T}_{\mathrm{execution}} + \mathrm{T}_{\mathrm{Stall}})}{\mathrm{T}_{\mathrm{execution}}} \tag{5}$$

$$\mathrm{E}_{\mathrm{dynamic(RA)}}(normalized) = (\frac{\mathrm{V}_{\mathrm{RA}}}{\mathrm{V}})^2 \times \frac{\mathrm{F}_{\mathrm{RA}}}{\mathrm{F}} \times (1 + \frac{\mathrm{T}_{\mathrm{Stall}}}{\mathrm{T}_{\mathrm{execution}}}) \tag{6}$$

5 EVALUATION

In the context of this paper, we evaluated energy savings and bandwidth reduction using an analytical model employing the results from a state-of-the-art simulator. Bandwidth reduction can be further used to apply V-F scaling on off-chip memory and save additional energy, but we leave the energy reduction study in off-chip memory for future work.

5.1 Average Energy and Bandwidth Reduction

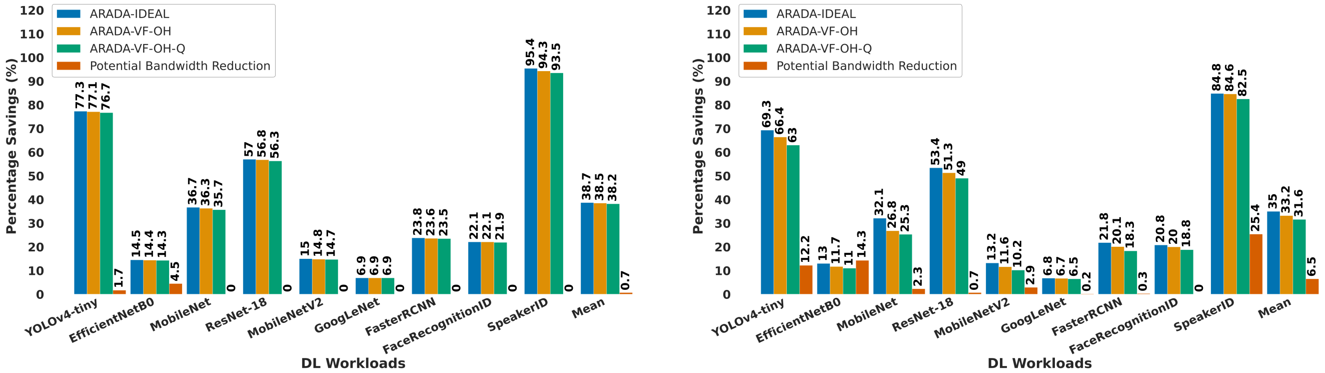
In this section, we evaluate the potential energy savings and memory bandwidth reduction of the two architectures as mentioned earlier in Section 4.1 with a range of off-chip memory types in Figure 5 compared to fixed allocation as in case of Race-to-Idle scheme. Both energy savings and memory bandwidth reduction are presented in normalized form along the y-axis, and the x-axis shows various memory types. Here we provide the energy savings for ARADA-IDEAL, ARADA-DVFS, and ARADA-DVFS-Q with respect to RTI. There are a couple of key takeaways here. First, as off-chip bandwidth increases, potential energy savings decrease, and possible BW reduction increases. For example, the energy savings for the *Edge TPU* decline from 38% to 31% for LPDDR4 to DDR5-4800, respectively for the ARADA-DVFS-Q scheme. And the bandwidth reduction possibility increases from 0.7% to 6.5% for the same range of memory types. The same pattern can be observed for the *HPC TPU* as well. This is because as the BW increases, more layers are shifted from the memory-bound to the compute-bound region and vice versa. The second important insight is that the ideal balance between the compute and memory capability (both size and bandwidth) is difficult to achieve in the context of efficiency. This fact is further enhanced by the changing workload scenarios, as we will observe in the next section. In short, it is fair to say that a specific resource allocation is required for each layer of individual DL workloads.

5.1.1 *Edge TPU .*

Next, we analyze each workload in detail, executing on *Edge TPU* for a range of off-chip memory types as shown in Figure 6. As a first observation, as we increase the BW starting from LPDDR4 in Figure 6a to DDR5 4800 in Figure 6b, the potential energy savings decrease and potential BW reduction increases as expected. Moreover, each workload has its own distinct behavior, and BW changes have a specific effect. However, the extent of change concerning memory technology is workload-dependent. Thus, it is important to estimate and apply resource allocation to each workload at per layer level. Another important point is the effect of DVFS overhead and V-F quantization. As bandwidth is increased the potential energy savings decrease because of smaller stall times. This also results in further reduction when considering ARADA-DVFS and ADARA-VF-OH-Q schemes as if stall time is less than DVFS overhead, reduction in V-F is not feasible. Stall times depend on the total execution time of layer or layer size in other words. Therefore the effect is different for various workloads. For example, the reduction in energy savings from ARADA-IDEAL to ARADA-VF-OH-Q is different for *YOLOv4-tiny* and *GoogLeNet* for LPPDDR4 and DDR5 4800 cases.

5.1.2 *HPC TPU.*

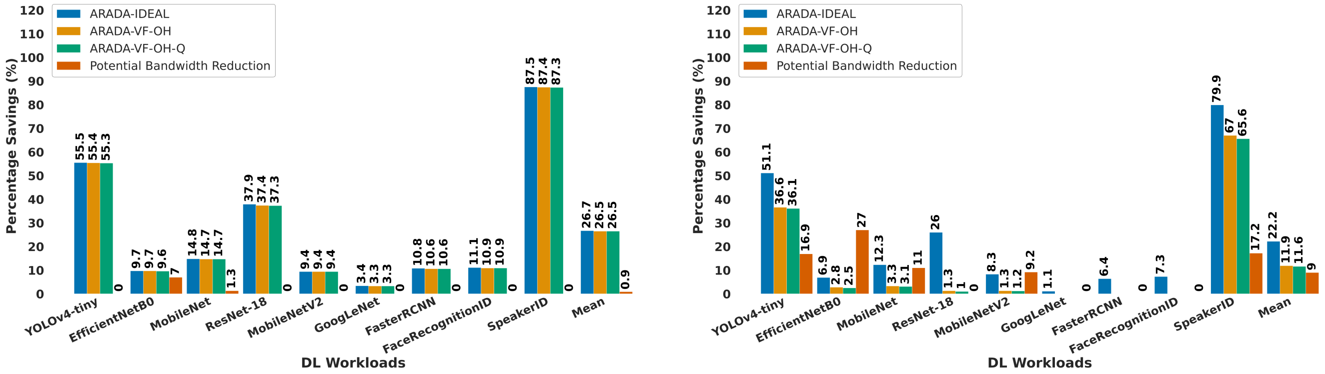
Results for the HPC TPU are presented in Figure 7, with a different range of memory types, i.e., DDR4 4400 Figure 7a and HBM2 Figure 7b. Again as we increase the BW, more layers shift from the memory-bound region to the compute-bound region. Different workloads behave differently owing to the fact that they have a unique mixture of compute and memory-bounded layers. Furthermore, the extent a particular layer is compute- or memory-bound is different, resulting in unique behavior. Accounting for DVFS overhead and quantization again reduces the energy savings. However, as can be seen in the case of Figure 7b, this reduction is significant. The reason is that HBM2 has very high bandwidth and this significantly reduces that stall times. If stall times are less than DVFS overheads then resource allocation is not applied. Therefore there is a significant difference between the DDR4-4400 and HBM2. Thus it is extremely important to analyze each workload in connection with the underlined architecture and allocate resources accordingly at runtime.



(a) LPDDR4

(b) DDR5 4800

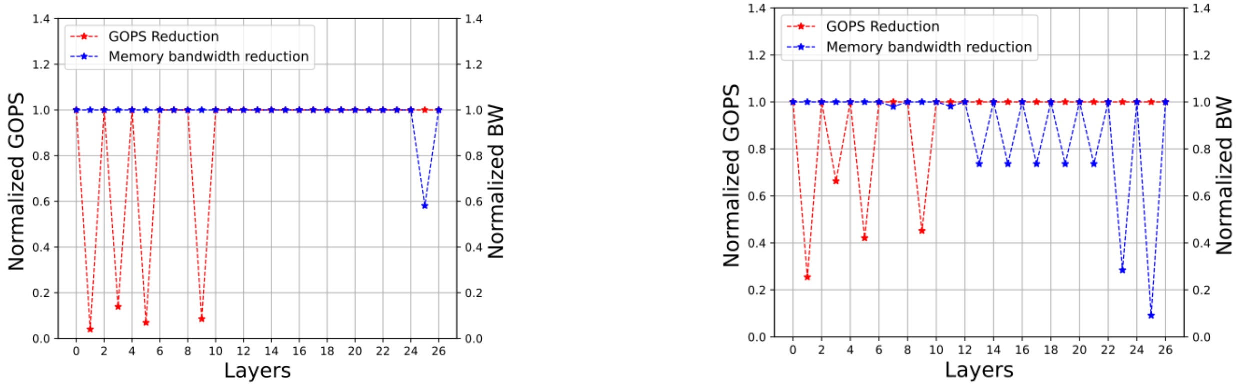
Figure 6: Energy Savings for *Edge TPU* with 4MB on-chip buffer and various off-chip memory types



(a) DDR4 4400

(b) HBM2

Figure 7: Energy Savings for various DL workloads for *HPC TPU* with 8MB on-chip buffer for different off-chip memory types



(a) DDR5-4800

(b) HBM2

Figure 8: Layer-wise compute and bandwidth reductions for MobileNet executing on 256 × 256 SA with 8 MB on-chip buffer

On the other hand, in the case of HBM2, there is a considerable amount of extra bandwidth. This fact is evident from the Figure 7b where for example the bandwidth reduction for EfficientNetB0 and MobilNetV2 increase to 27% and 9.2% for HBM2 from 7% and 0% for DDR4-4400. Another way to analyze is to say that the compute stalls (in the case of DDR4 4000) are replaced with memory stalls (in the case of HBM2). In short, we can ascertain that per-layer resource allocation is important for both ends of the compute continuum to get optimal efficiency out of underlined hardware.

5.1.3 Layer-by-Layer Analysis. To further investigate the detailed, intricate behavior, we present a per-layer analysis, showing the potential reduction in the TOPS and BW. In this context, we present results for *MobileNet* for executing on HPC TPU with DDR5 4800 and HBM2 in Figure 8a and Figure 8b respectively. There are several interesting observations here. First, as discussed earlier, most layers within a DL application are either compute-bound or memory-bound. However, some are not, indicating they are at an efficient execution point. This is the case, for example, for layer 10-24 for HPC TPU with DDR5-4800 as shown in Figure 8a. As the memory BW increases in HPC TPU with HBM2, some of these layers shift to compute-bound regions. For example, layers 13, 15, and 17 are shifted to the compute-bound region. The second important conjuncture is that the various layers have different degrees of compute-bounded or memory-bounded behavior resulting in different potential reductions in energy and bandwidth. In this context, it is important to note that this phenomenon is affected by the hardware platform, as can be seen by comparing the two figures. For example, layer 13 has approximately 25% BW reduction in the case of HBM2 and none in the case of DDR5-4800. Similarly, looking at layer 5, one can observe a TOPS reduction of 90% for DDR5-4800 and 60% for HBM2. Switching the off-chip memory from DDR5-4800 to HBM2 allows data to be moved faster reducing the memory-bounded behavior of the layer. In other words, the stall time waiting for data is decreased, as expected.

6 RELATED WORK

The problem of energy efficiency is one of the primary concerns in computer system design and gained significant attention recently [1–3, 9, 13, 19, 25, 32]. In the context of DL application execution, it is even more critical because of the high compute and memory requirements. Therefore, it has gained considerable attention. In the context of this paper, there are several categories of works.

The first category includes investigation and impact studies, where the impact of resource allocation on energy efficiency is studied. For example, Tang et al. [33] presented an experimental study where the impact of DVFS in GPUs on the performance and energy for inference and training is studied. The authors conclude that the optimal setting for performance and energy is not the highest or lowest. Similarly, Oh et al. [27] present an experimental study comparing energy and performance on CPU and GPU.

The second category includes works that propose adaptive techniques to employ resource allocation to improve energy efficiency. For example, Nabavinejad et al. [24] have proposed using batch size to enhance the DVFS capability and improve performance under a power cap. Yao et al. [35] offer a so-called "revenue" model that computes the optimal batch size and frequency setting. This scheme employs the curve-fitting on the exhaustive sampled data.

Jiang et al. [17] have proposed to use a power-down state using DVFS to save energy while in an idle period after processing an input image and waiting for the following image on FPGA. Liu et al. [21] proposed a mechanism based on sparsity detection to control individual PEs by employing DVFS and power gating to save energy. Nabavinejad et al. [23] offer a method for precision control and DVFS to reduce power under QoS of response time in server-based CNN inference. Yu et al. [36] proposes a heuristic to manage the frequency of inference servers based on requests in the queue to satisfy response time. Jiang et al. [16] propose an accelerator design and V-F scaling policy similar to power gating.

All the above proposals do not consider the varying compute and memory bandwidth requirements in the DL workloads during execution at layer-by-layer granularity. Therefore, they fail to fully exploit the available opportunity for improving energy efficiency. In contrast, ARADA exploits these variations at the layer-by-layer level and proposes heuristics to use them.

7 CONCLUSION AND FUTURE WORK

This paper investigates the problem of improving energy efficiency in DL inference. In this context, we identify that the identical resource allocation (RA) for complete DL model execution leads to inefficiency. Furthermore, RA decisions must be taken by considering both the DL workload and the hardware platform. DL workloads have compute-bounded and memory-bounded execution phases typically defined by the model's layers. To achieve energy efficiency, these phases must be handled differently. In short, the behavior of each layer must be analyzed, and consequently, the resources must be allocated according to the compute and memory bandwidth requirements. To this end, we proposed a method to analyze and allocate resources according to the needs of various phases of execution. Our approach *ARADA* aims to identify the compute- and memory-bounded layers within the DL application and allocate resources based on fixed compute and memory bandwidth budget specified in hardware specification.

Our investigation reveals significant potential energy savings and bandwidth reduction for a set of DL applications for two architectures, an Edge TPU, and an HPC TPU. We repeated the experiments using a range of memory types offering different bandwidths. The results showed that *ARADA* can, for example, on average, save 38% energy and 0.7% memory bandwidth for the Edge TPU coupled with LPDDR4 for the ADARA-VF-OH-Q scheme. As for future extensions, we would like to explore the possibility of extending *ARADA* for the GPUs and re-configurable DL accelerators such as [7, 18] where additional resources, e.g., PEs can be allocated as per compute requirement.

ACKNOWLEDGMENTS

This publication incorporates results from the VEDLIoT project, which received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 957197. This work was also partly supported by the Swedish Foundation for Strategic Research (contract number CHI19-0048) under the PRIDE project.

REFERENCES

[1] M. Waqar Azhar, Miquel Pericàs, and Per Stenström. 2019. SaC: Exploiting Execution-Time Slack to Save Energy in Heterogeneous Multicore Systems. In *Proceedings of the 48th International Conference on Parallel Processing* (Kyoto, Japan) *(ICPP 2019)*. ACM, Article 26, 12 pages. https://doi.org/10.1145/3337821.3337865

[2] M. Waqar Azhar, Miquel Pericàs, and Per Stenström. 2022. Task-RM: A Resource Manager for Energy Reduction in Task-Parallel Applications under Quality of Service Constraints. *ACM Trans. Archit. Code Optim.* 19, 1, Article 11 (jan 2022), 26 pages. https://doi.org/10.1145/3494537

[3] M Waqar Azhar, Per Stenström, and Vassilis Papaefstathiou. 2017. SLOOP: QoS-supervised loop execution to reduce energy on heterogeneous architectures. *ACM Transactions on Architecture and Code Optimization (TACO)* 14, 4 (2017), 1–25.

[4] Lin Bai, Yiming Zhao, and Xinming Huang. 2018. A CNN Accelerator on FPGA Using Depthwise Separable Convolution. *IEEE Transactions on Circuits and Systems II: Express Briefs* 65, 10 (2018), 1415–1419. https://doi.org/10.1109/TCSII.2018.2865896

[5] Amirali Boroumand, Saugata Ghose, Berkin Akin, Ravi Narayanaswami, Geraldo F. Oliveira, Xiaoyu Ma, Eric Shiu, and Onur Mutlu. 2021. Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks. In *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 159–172. https://doi.org/10.1109/PACT52795.2021.00019

[6] Kumar Chellapilla, Sidd Puri, and Patrice Simard. 2006. High performance convolutional neural networks for document processing. In *Tenth international workshop on frontiers in handwriting recognition*. Suvisoft.

[7] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 367–379. https://doi.org/10.1109/ISCA.2016.40

[8] Howard David, Chris Fallin, Eugene Gorbatov, Ulf R. Hanebutte, and Onur Mutlu. 2011. Memory Power Management via Dynamic Voltage/Frequency Scaling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing* (Karlsruhe, Germany) *(ICAC '11)*. Association for Computing Machinery, New York, NY, USA, 31–40. https://doi.org/10.1145/1998582.1998590

[9] Sai Santosh Dayapule, Fan Yao, and Guru Venkataramani. 2019. PowerStar: Improving Power Efficiency in Heterogenous Processors for Bursty Workloads with Approximate Computing. In *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 175–182. https://doi.org/10.1109/CloudCom.2019.00035

[10] Michael Ditty. 2022. NVIDIA ORIN System-On-Chip. In *2022 IEEE Hot Chips 34 Symposium (HCS)*. 1–17. https://doi.org/10.1109/HCS55958.2022.9895609

[11] Google. [n. d.]. Google TPU System Architecture. https://cloud.google.com/tpu/docs/system-architecture-tpu-vm accessed: 10.02.2023.

[12] Google. 2023. Google debuted Edge TPU. https://cloud.google.com/edge-tpu accessed: 10.02.2023.

[13] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. 2011. Dynamic Knobs for Responsive Power-Aware Computing. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (Newport Beach, California, USA) *(ASPLOS XVI)*. Association for Computing Machinery, New York, NY, USA, 199–212. https://doi.org/10.1145/1950365.1950390

[14] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[15] Aleksandar Ilic, Frederico Pratas, and Leonel Sousa. 2014. Cache-aware Roofline model: Upgrading the loft. *IEEE Computer Architecture Letters* 13, 1 (2014), 21–24. https://doi.org/10.1109/L-CA.2013.6

[16] Weixiong Jiang, Heng Yu, Xinzhe Liu, and Yajun Ha. 2019. Energy Efficiency Optimization of FPGA-based CNN Accelerators with Full Data Reuse and VFS. In *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. 446–449. https://doi.org/10.1109/ICECS46596.2019.8964717

[17] Weixiong Jiang, Heng Yu, Jiale Zhang, Jiaxuan Wu, Shaobo Luo, and Yajun Ha. 2020. Optimizing energy efficiency of CNN-based object detection with dynamic voltage and frequency scaling. *Journal of Semiconductors* 41, 2 (feb 2020), 022406. https://doi.org/10.1088/1674-4926/41/2/022406

[18] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. *SIGPLAN Not.* 53, 2 (mar 2018), 461–475. https://doi.org/10.1145/3296957.3173176

[19] J. Li and J. F. Martinez. 2006. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *The Twelfth International Symposium on High-Performance Computer Architecture, 2006*. 77–87. https://doi.org/10.1109/HPCA.2006.1598114

[20] I. Lin, B. Jeff, and I. Rickard. 2016. ARM platform for performance and power efficiency — Hardware and software perspectives. In *2016 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. 1–5. https://doi.org/10.1109/VLSI-DAT.2016.7482541

[21] Siqin Liu and Avinash Karanth. 2021. Dynamic Voltage and Frequency Scaling to Improve Energy-Efficiency of Hardware Accelerators. In *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. 232–241. https://doi.org/10.1109/HiPC53243.2021.00037

[22] Michael Mattioli. 2022. Meet the FaM1ly. *IEEE Micro* 42, 3 (2022), 78–84. https://doi.org/10.1109/MM.2022.3169245

[23] Seyed Morteza Nabavinejad, Hassan Hafez-Kolahi, and Sherief Reda. 2019. Coordinated DVFS and Precision Control for Deep Neural Networks. *IEEE Computer Architecture Letters* 18, 2 (2019), 136–140. https://doi.org/10.1109/LCA.2019.2942020

[24] Seyed Morteza Nabavinejad, Sherief Reda, and Masoumeh Ebrahimi. 2022. Coordinated Batching and DVFS for DNN Inference on GPU Accelerators. *IEEE Transactions on Parallel and Distributed Systems* 33, 10 (2022), 2496–2508.

[25] Mehrzad Nejat, Miquel Pericas, and Per Stenstrom. 2019. QoS-Driven Coordinated Management of Resources to Save Energy in Multi-core Systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 303–313. https://doi.org/10.1109/IPDPS.2019.00040

[26] Nvidia. [n. d.]. *NVDLA*. http://nvdla.org/

[27] Sangyoon Oh, Minsub Kim, Donghoon Kim, Minjoong Jeong, and Minsu Lee. 2017. Investigation on performance and energy efficiency of CNN-based object detection on embedded device. In *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*. 1–4. https://doi.org/10.1109/CAIPT.2017.8320657

[28] S. Park, J. Park, D. Shin, Y. Wang, Q. Xie, M. Pedram, and N. Chang. 2013. Accurate Modeling of the Delay and Energy Overhead of Dynamic Voltage and Frequency Scaling in Modern Microprocessors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 5 (May 2013), 695–708. https://doi.org/10.1109/TCAD.2012.2235126

[29] A. Raghavan, L. Emurian, L. Shao, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. K. Martin. 2013. Utilizing Dark Silicon to Save Energy with Computational Sprinting. *IEEE Micro* 33, 5 (Sep. 2013), 20–28. https://doi.org/10.1109/MM.2013.76

[30] Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2020. A Systematic Methodology for Characterizing Scalability of DNN Accelerators using SCALE-Sim. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 58–68. https://doi.org/10.1109/ISPASS48437.2020.00016

[31] Samsung. [n. d.]. LPDDR5X-K3KL2L20DM-JGCT Product Brief. https://semiconductor.samsung.com/us/dram/lpddr/lpddr5x/k3kl2l20dm-jgct accessed: 10.02.2023.

[32] Jinho Suh, Chieh-Ting Huang, and Michel Dubois. 2015. Dynamic MIPS Rate Stabilization for Complex Processors. *ACM Trans. Archit. Code Optim.* 12, 1, Article 4 (April 2015), 25 pages. https://doi.org/10.1145/2714575

[33] Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen Chu. 2019. The impact of GPU DVFS on the energy and performance of deep learning: An empirical study. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems*. 315–325.

[34] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2009), 65–76.

[35] Chunrong Yao, Wantao Liu, Weiqing Tang, Jinrong Guo, Songlin Hu, Yijun Lu, and Wei Jiang. 2021. Evaluating and analyzing the energy efficiency of CNN inference on high-performance GPU. *Concurrency and Computation: Practice and Experience* 33, 6 (2021), e6064.

[36] Junyeol Yu, Jongseok Kim, and Euiseong Seo. 2021. A DNN Inference Latency-aware GPU Power Management Scheme. In *2021 IEEE 3rd Eurasia Conference on IOT, Communication and Engineering (ECICE)*. 551–554. https://doi.org/10.1109/ECICE52819.2021.9645654