2021 IEEE/ACM Redefining Scalability for Diversely Heterogeneous Architectures Workshop (RSDHA) | 978-1-6654-5877-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/RSDHA54838.2021.00006

Multi-accelerator Neural Network Inference in Diversely Heterogeneous Embedded Systems

Ismet Dagli
Computer Science Department
Colorado School of Mines
ismetdagli@mines.edu

Mehmet E. Belviranli
Computer Science Department
Colorado School of Mines
belviranli@mines.edu

***Abstract*—Neural network inference (NNI) is commonly used in mobile and autonomous systems for latency-sensitive critical operations such as obstacle detection and avoidance. In addition to latency, energy consumption is also an important factor in such workloads, since the battery is a limited resource in such systems. Energy and latency demands of critical workload execution in such systems can vary based on the physical system state. For example, the remaining energy on a low-running battery should be prioritized for motor consumption in a quadcopter. On the other hand, if the quadcopter is flying through obstacles, latency-aware execution becomes a priority. Many recent mobile and autonomous system-on-chips embed a diverse range of accelerators with varying power and performance characteristics which can be utilized to achieve this fine trade-off between energy and latency.**

In this paper, we investigate Multi-accelerator Execution (MAE) on diversely heterogeneous embedded systems, where sub-components of a given workload, such as NNI, can be assigned to different type of accelerators to achieve a desired latency or energy goal. We first analyze the energy and performance characteristics of execution of neural network layers on different type of accelerators. We then explore energy/performance trade-offs via layer-wise scheduling for NNI by considering different layer-to-PE mappings. We finally propose a customizable metric, called multi-accelerator execution gain (MAEG), in order to measure the energy or performance benefits of MAE of a given workload. Our empirical results on Jetson Xavier SoCs show that our methodology can provide up to 28% energy/performance trade-off benefit when compared to the case where all layers are assigned to a single PE.

***Index Terms*—Heterogeneous systems, neural networks inference, autonomous systems, energy/performance trade-off**

I. Introduction

Autonomous systems are becoming wide-spread over the last decade as high performance computing (HPC), machine learning (ML) and robotics advance further to provide the required technical capabilities that full autonomy requires. Safety-critical workloads on autonomous systems require high-performance compute capability in order to meet hard-deadlines for safe execution. In the meantime, the increase in the computational demand results in higher energy consumption, hence making the power consumption required for computation comparable to the power needed by the mechanical operation of a cyber-physical system (CPS), such as a quadcopter drone. For this reason, computational resources should be utilized based upon the desired energy/performance trade-off (EPT) depending on the capabilities of the underlying system.

Modern mobile and embedded SoCs, such as Qualcomm's Snapdragon Xilinx Zynq MPSoC and NVIDIA's Xavier platforms, employ a diverse range of domain specific accelerators to perform critical tasks with low latency and power. For example, NVIDIA's Xavier SoC includes two specialized programmable accelerators, NVIDIA Deep Learning Accelerator (DLA) and Progammable Vision Accelerator (PVA) in addition to a graphical processing unit (GPU). In some cases, a common operation such as the convolution operation can be executed on more than one type of accelerator, e.g., DLA, GPU and PVA on Xavier, with varying latency and power characteristics. In such cases, a collaborative execution of the underlying workload across different processing elements (PE) could be needed to maximize or adjust the desired resource efficiency and utilization. For example, while the GPU on Xavier provides the best performance, the DLA on the same platform is almost twice energy efficient for most NNI layers. However, under a target energy budget, the most feasible action might be to assign most latency sensitive layers to GPU while leaving the layers who demonstrate best energy-efficiency to the DLA. While such a distribution will not result in either best performance or least energy usage, systems can take advantage of a trade-off in between, to obtain a latency lower than all-DLA execution even though the power budget is not sufficient for all-GPU execution.

In mobile and autonomous systems, performance objectives are dynamic and depend on the physical conditions. Moreover, in most cases, such systems have limited system resources, such as battery capacity, to meet the performance objectives. Continuously changing dynamics of CPS make the optimization problem more challenging. While adjusting the frequency of the hardware through well-known techniques like DVFS is a common practice, recent availability of alternative accelerators in the same system for a given operation makes another execution paradigm possible: Collaborative multi-accelerator execution (MAE). Collaborative MAE targets to explore varying EPT options to adapt the computing resource usage into real-life physical requirements of the system and it has been considered by a very limited number of studies only. The work in [1] targets to distribute tasks across different types of accelerators for better resource utilization under a given set

978-1-6654-5877-1/21/$31.00 ©2021 IEEE
DOI 10.1109/RSDHA54838.2021.00006

of constraints. Houssam-Eddine et al. [2] proposed a real-time application model to analyze and run alternatives of tasks on heterogeneous hardware to explore preemptive scheduling of tasks. However, these studies consider only performance and disregard energy consumption of the resulting schedules.

Our contribution: In this paper, we consider spanning the execution of NNI across different type of accelerators in a diversely heterogeneous system and our goal is to explore the trade-off between execution time or energy consumption. We explore the execution of various NNI workloads on a heterogeneous system by partitioning the layers among several accelerators. Each layer will be assigned to PEs based on their capabilities of performing better for a target EPT. This paper makes the following contributions:

- We show that there exists a trade-off between performance and energy consumption. This trade-off can be optimized by assigning and executing several layers on different processing units.
- We build a metric, called multi-accelerator execution gain "MAEG", to represent the trade-off between energy and power. This metric represents the trade-off between two accelerators based on time and energy benefits of them.
- We present and analyze the results of our design space exploration (DSE) of the execution of GoogleNet network on DLA and GPU of Xavier AGX architecture with varying layer distributions.

II. Background

Nvidia's Xavier AGX and NX SoCs pack three different types of programmable accelerators, a CUDA programmable GPU, a programmable vision accelerator (PVA) and a deep learning accelerator (DLA), along with powerful tools to measure performance and energy. In this study, we focus on exploring multi-accelerator execution of NNI on the DLA and the GPU of Xavier platform.

A. TensorRT

TensorRT [3] is a framework consisting of a NNI optimizer and runtime to achieve high-performance on various platforms. TensorRT engine builder applies pre-runtime optimizations such as proper layer fusing, precision adjustment and memory requirement reduction. The engine builder tries multiple variations and combinations of layers in the given network and finds an optimal configuration that results in improved latency, throughput and memory footprint for the underlying architecture.

B. NVIDIA Deep Learning Accelerator

DLA [4] is an open-source domain specific accelerator hardware for neural network inference. The architecture internally embeds an internal pipeline of layer-specific engines for convolution, activation, pooling and reshape. NVIDIA Xavier SoC embeds a *small* configuration of DLA, named as NVDLA, capable of 11.4 TOPS int8 or 5.7 TFLOPS FP16 performance. While performance of DLA in Xavier is less than half of the GPU in this architecture, power consumption is less than $1/4^{th}$ of the GPU for most operations. DLA in Xavier SoC can be only used via the TensorRT library and there exist some generic and layer-wise restrictions. Specifically, at the time this paper is written, TensorRT does not provide execution of some uncommon layers,*i.e.* LeakyReLU layer and the supported batch sizes are limited to the range between 1 to 32. TensorRT allows developers to specify the *GPUFallBack* option to map the layers that cannot be run on the DLA to the GPU. Since the fall-backs require all the transient data belonging the previous layer to be flushed back to the main system memory, layer transitions between DLA and GPU should be carefully programmed to prevent performance degradation.

III. Considerations for Multi-Accelerator Execution

A. Motivation

In diversely heterogeneous SoCs, an operation (*i.e.*, a task or kernel in an application) can often be accelerated via different domain-specific accelerators (DSA) with varying performance, energy, and latency characteristics. For example, a convolution operation can be set to run on the CPU, GPU, programmable vision accelerator (PVA), or deep learning accelerator (DLA). The DSA that would provide the optimal execution time and/or energy efficiency of the convolution operation depends both on the accelerator capabilities, and on the properties of the convolution operation such as matrix size and filter dimensions. Depending on the dynamic requirements of the system (*e.g.*, high throughput, low energy) and runtime parameters of the operation, such as the number of objects and image size, the programmer (or the system scheduler) can be capable of choosing to map different operations to different DSAs throughout the execution of an application.

As a result, the flexibility to be able to run an operation on different DSAs for different performance and power targets enables *collaborative execution* where different DSAs are used for different operations in a workload. Alternatively, collaborative execution of popular workloads, such as NNI, on multiple DSAs is relatively a new and unexplored scheme, which has the potential to provide unique benefits for budgeted execution scenarios. To demonstrate the feasibility of such executions, we have designed preliminary results from an experiment that is depicted in Table I. Our results show that distributing the workloads (*i.e.* layers) of a neural network inference across a GPU and a DLA *collaboratively* could provide a customizable trade-off between energy and performance

Layer Distribution	Latency(ms)	Energy(j)
All-GPU	2.99	20.08
111 (GPU) - 29 (DLA)	3.46	19.15
99 (GPU) - 41 (DLA)	3.75	15.80
52 (GPU) - 88 (DLA)	4.2	13.97
11 (GPU) - 129 (DLA)	4.81	11.93
ALL-DLA	5.52	10.05

TABLE I: Latency and energy results obtained by GPU, DLA and collobarative execution of DLA and GPU with potential transition points among layers

on NVIDIA's Xavier NX system. More specifically, as more layers of the network are executed on the DLA and the rest are run on the GPU, the energy consumption (Joules per image) can be significantly improved with some adverse impact on the latency (Seconds per image).

B. Considerations

Building a generalized methodology for EPT-aware multi-accelerator execution requires the following to be considered:

1) Execution time and energy characterization

Common PEs such as CPU, GPU, FPGA, and ASIC have different architectural designs and constraints which limit the workloads that can be run depending on their requirements, such as memory space demands. For heterogeneity-aware assignment of different layers, characteristic features of layers must be either measured or predicted. Recent performance analysis of neural networks on SoC [5] shows communication-computation ratio depending on several parameters, which are challenging to detect and optimize during compile time. Kernel size and type are two essential factors that affect the runtime and energy consumption of an operation. Layers on neural networks are mostly based on matrix operations, therefore height and width of the input matrix have an affect on the execution time. Other parameters such as activation size, activation function, and the number of parameters also affect the complexity of different layer types.

2) Inter-PE data sharing and transfer

The communication between PEs in a collaborative MAE scenario must be considered carefully to minimize the extra R/W overhead while achieving the EPT goals. Transient values are often used to represent disposable memory spaces that are only needed temporarily and they are often marked as not to be serialized to disk or flushed to the memory. By default, TensorRT engine optimizer treats the tensors passed between layers as transient unless the layer is marked as an output layer. When multiple accelerators are used interchangeably in a neural network, depending upon the point where the execution transitions between two accelerators, there will be additional writes and reads by these two OPs due to usage of non-transient data.

3) Sharing the load across PEs

Load balance must be considered as well in order to maximize the overall efficiency of the system since PEs have different computing power capabilities and the tasks include several subtasks with different complexities. Different types of PEs have various capabilities for running the kernels depending on their operation type and data size. In order to obtain maximum performance out of available PEs, loads must be distributed among PEs by considering estimated/expected execution time and energy consumption. This is a well-studied mapping problem, which is NP-complete [6], and existing solutions [7], [8] employ heuristics or dynamic scheduling techniques to manage load distribution across PEs. Otherwise, unequal distribution of workloads across PEs will possibly result in an overall slowdown of the system.

IV. Case Study: GPU+DLA Collaborative Execution for Convolutional Neural Network Inference

In this section, we present our exploration of collaborative MAE over a case study. We first investigate the *MAE transition point* (MTP), where the execution flow in the NNI switches between DLA and GPU. We then provide a customizable EPT parameter that will help schedulers to find an optimal place for the transition point.

A. Setup

In this study, we use Nvidia's Jetson Xavier AGX and NX SoCs since they embed one performance efficient (i.e., GPU) and two energy efficient (i.e., DLA) accelerators together with access to the same shared DRAM memory. We limit our experiments to one DLA because TensorRT does not allow to use multiple DLAs for the execution of the same neural network. At the time this paper is written, the latest version of JetPack provides the necessary setup for our experiments, which are Ubuntu OS 18.04, Cuda 10.2, TensorRT 7.1.3, OpenCV 4.1.1. We use TensorRT engine to optimize the pre-trained models collected from Dusty-nv repository [9]. We run our experiments on a CNN model, GoogleNet [10], whose all layers can be executed on DLA. This allows us to flexibly explore all possible layer to PE assignments, without TensorRT engine falling back to GPUs on DLA assigned layers; hence we avoid unwanted layer transitions between the GPU and DLA.

Transitions from the GPU to DLA can only be programmed manually by the *setDeviceType* TensorRT API call. Transitions back to GPU from DLA happens when a planned DLA execution falls back to GPU or *setDeviceType* is not set for a particular layer. All layers in a network can be also set to execute DLA globally by setting the *useDLACore* parameter for the TensorRT runtime executable, *trtexec*.

B. Factors Affecting the Selection of MAE Transition Point

We explore the MAE trade-off by investigating the selection of the MAE Transition Point (MTP) where the execution flow is handled from one accelerator to another. For the simplicity of our analysis, we assume that (1) the execution begins in either GPU or DLA, (2) there is only one MTP in the execution of an NNI, and (3) DLA and GPU are not used at the same time.

1) Overhead of MAE Transition

Layer transition between PEs requires moving layer input/output data across the memory subsystems of both PEs. In Jetson Xavier SoCs, for example, the DLA has its own private buffer, called *convolution buffer* [4]. While performing the execution of a group of layers in DLA, if the execution is to be transitioned into the GPU, this process will require saving the state of the *convolution buffer* in DLA into the shared memory of the SoC (i.e., DRAM). Once the output of the last DLA layer is visible by other PUs in the SoC, the GPU execution for the remainder of the layers begins by the CuDNN kernel call inserted by the TensorRT engine. Additionally, after the transition, the cold cache misses caused by the initial memory

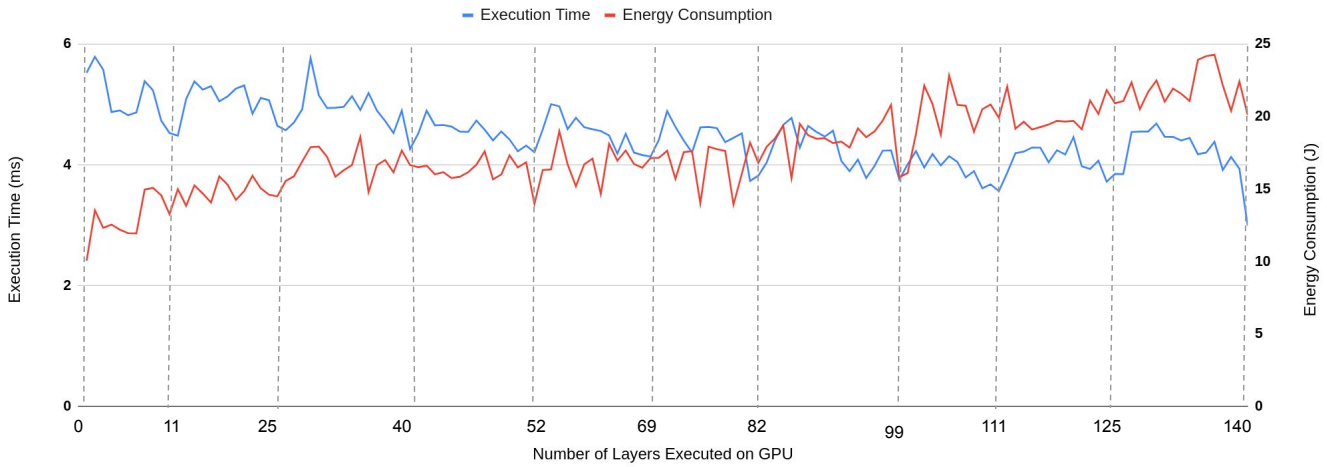

Fig. 1: Execution time and energy consumption comparison on GoogleNet with different workload assignment across PEs, making the transition from GPU to DLA after the layers with the numbers shown in the X-axis. The data points at X=99, corresponds to the inter-accelerator transition at layer number 99, which means that layers 1-99 are executed on GPU and remaining layers (100-140) are executed on DLA.

instructions executed by the GPU will result in an implicit warm-up period. This period will slow down the execution of layers for an uncertain amount of time, depending on the size of the cache, the number of ports, and the available memory bandwidth.

2) The location of the MAE Transition Point

In convolutional neural networks (CNN), core layers, such as Convolution layers, are frequently followed by specific operations such as ReLU and Pooling. Such group of layers are often combined as a bigger group of layers to reduce memory copy and access overheads [11]. In our case, this optimization is performed automatically during TensorRT engine building step. However, layers cannot be merged if the transition point between accelerators is picked without considering the optimized groups of layers. The resulting slowdown is even higher when the transition overhead is also added to the cost of missing optimization opportunity. Additionally, many neural network architectures embed some layers producing a significantly reduced amount of data as output, which is often a result of repetitive pooling operation calls. If the transition is picked after the output data is reduced, the data that needs to be transferred into the next layer as input will also be minimal. Thus, the overhead of PU-transition will be minimized.

C. MAE Transition Point Exploration

To observe the effects for MTP on execution time and total energy consumption we performed a design-space exploration by applying a single transition between from GPU to DLA after each of the 140 layers in GoogleNet [10]. We have obtained the execution times via *trtexec* and energy consumption by integrating the PU-specific power consumption values reported by *tegrastats* call.

Fig. 1 shows the resulting execution time and energy consumption corresponding to different transition points we tried, which are shown on the X-axis. For example, the two data points at $X = 52$ correspond to the execution where layers 1-to-52 are assigned to GPU and the layers 53-140 are assigned to DLA. Overall, the model tends to run faster and consume more energy as more layers are running on GPU. When all layers are executed on DLA (i.e., $X = 0$) the execution time is longest but the energy consumption is lowest. Execution time drops almost 50% when all layers are executed on GPU (i.e., $X = 140$), but energy consumption increases almost 100%.

There are some data points in the experiment where running a few more layers on DLA results in less energy consumption and less execution time. For example, when two transition points at $X = 98$ and $X = 99$ are compared, we observe 31% less energy consumption and 12% less execution time even though the model at $X = 99$ executes 1 additional layer on DLA compared to the former model. In this particular example, when $X = 98$, transition occurs in the middle of an inception layer [10], and it prevents efficient fusion of this group of layers. Moreover, during such transitions, our offline profiling shows that extra input and output reformatting layers are added by TensorRT engine builder in order to perform data serializing operations. Therefore, transitions at these points show a considerable increase in both execution time and energy consumption. On the other hand, there are some other set of transitions, such as $X = 99$ and $X = 100$, which corresponds to relatively close energy and time values. Since no fusion optimization operation and no input/output data conversion between layers are performed between such layers, the results for time and energy do not significantly change.

D. Standalone Characterization of Group of Layers

In GoogleNet [10] architecture, inception layer is proposed as a combination of layers with different sliding window

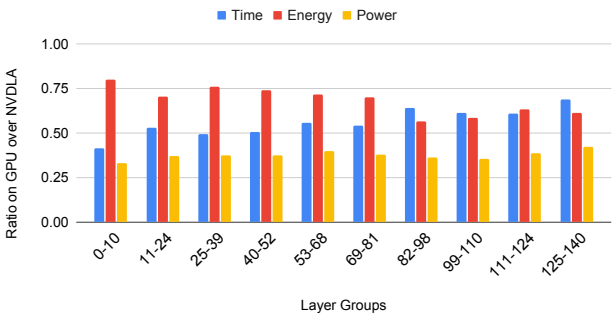

Fig. 2: The time, energy and power ratios of DLA executions over GPU when different layer groups are separately executed and measured for time, energy, and power.

sizes. Inception layers are often repeated multiple times with varying parameters on matrix and convolutional filter sizes. To better understand how layers perform when they are executed on DLA or GPU *separately* (i.e., not collaboratively), we used *IProfiler*, an API call on TensorRT, to characterize their execution time and energy consumption. However, to simplify our experiments, we grouped 140 layers into 10, where each group contains an inception layer and associated layers which are added for loss mitigation. The absolute execution time and energy values for the layer groups are given in Table II. Using the absolute values in Table II, we also derive the relative time, energy and power consumption ratios of DLA, when GPU values are taken as the baseline, and illustrate the comparison in Fig. 2. It is important to note that we do not consider transitions in this subset of experiments, and rather focus on individual characterization of group of layers on DLA and GPU separately.

The results for the execution time and energy when the layers are grouped give additional insights. The DLA performed faster while executing layers towards the end of the network, since the kernel sizes got smaller. GPU is capable of exploiting more data parallelism with larger kernels, and the small buffers in DLA are more effective when the matrix sizes are smaller. More interestingly, there is a clear trade-off relationship between time and energy results on the comparison of results on DLA vs. GPU: GPU consumes less energy when compared to DLA towards the end of the layer groups whereas executions on GPU tend to proportionately last longer.

Layer Groups	DLA Time	GPU Time	DLA Energy	GPU Energy
0-10	1.89310	0.783052	2.8813120	3.6161341
11-24	0.51206	0.26987	0.62317702	0.88437388
25-39	0.75149	0.36954	1.0175226	1.34292507
40-52	0.48472	0.24493	0.58505933	0.79261807
53-68	0.49370	0.27547	0.59392819	0.83247336
69-81	0.44470	0.24087	0.53320357	0.76381145
82-98	0.57197	0.36504	0.69266553	1.22690952
99-110	0.48300	0.29495	0.55062285	0.94533365
111-124	0.39439	0.23978	0.47485639	0.75196764
125-140	0.42973	0.29452	0.52127176	0.849399717

TABLE II: Latency(ms) and energy(J) results obtained by standalone execution of different layers on GPU and DLA separately.

E. Leveraging EPT with Newly Proposed MAEG Metric

We show the relationship between time and energy over GPU and DLA in Sections IV-C and IV-D. Since there is a non-linear relationship between execution time and energy consumption, finding optimized schedules based on two different variables can be challenging. Therefore, for a given transition point, we propose a metric called multi-accelerator execution gain, *MAEG*, which represents the trade-off relation between energy and time as a single, optimizable value. In this work, we construct *MAEG* based on collaborative execution on two accelerators only, but we plan to generalize it for arbitrary number of accelerators.

$$MAEG = \alpha \times TimeGain + \beta \times EnergyGain$$

$$TimeGain = \frac{|Time_{acc\_1} - Time_{MAE}|}{|Time_{acc\_1} - Time_{acc\_2}|}$$

$$EnergyGain = \frac{|Energy_{acc\_2} - Energy_{MAE}|}{|Energy_{acc\_1} - Energy_{acc\_2}|}$$

$$1 = \alpha + \beta$$

The *MAEG* depends on *time* and *energy gain*s which are calculated by finding the ratio of time or energy on a given transition point over a single execution result on both accelerators. More specifically, time gain is calculated by differences in our multi-accelerator executions and a single execution on an accelerator over differences in both single executions on both accelerators. Since the energy consumption tends to be less on the other device, we define the similar formula as a energy gain but we take the low-power accelerator as a base on the dividend of the formula. Both results are divided by both ends of the energy/time spectrum, which are the differences in energy/time when the entire network is run separately on both accelerators. α and β values are customizable parameters in order to dynamically modify the significance of time and energy factors so that the user can weigh the importance of either factors into their schedules as needed.

To show how MAEG can be used to explore the benefits of multi-accelerator execution, we perform an experiment and the results are shown in Fig. 3. In this experiment, we increase the batch size (as previously shown by [12] to be increasing the total throughput) and try to find an optimal transition point for each batch size. The MAEG values shown in the Y-axis are calculated by taking $\alpha = 0.5$ and $\beta = 0.5$. The results provide optimal transition based on the highest MAEG values for each set of batch sizes. The trendline across various batches tends to behave similarly as several MAEG results, depending on the batch numbers. The results show that the near-optimal multi-accelerator execution with MAEG parameters $\alpha = 0.5$ and $\beta = 0.5$ for batch numbers can be obtained at $batch = 4$. This is a general case because DLA shows best performance/energy

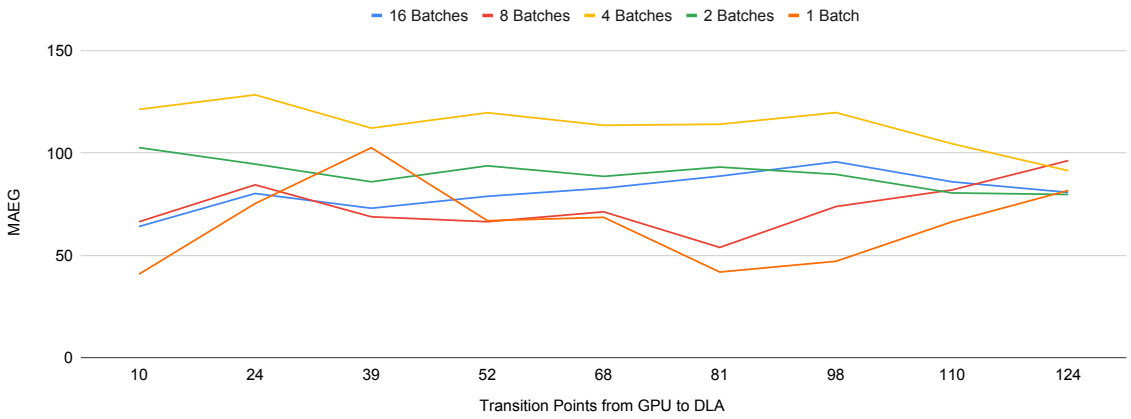

Fig. 3: MAEG results by batch numbers. The total gain is calculated by using $\alpha = 0.5$ and $\beta = 0.5$ values. X-axis indicates where the transition from GPU to DLA is performed.

efficiency with lower batch values. Alternatively, one can use a similar analysis to find other reasonable trade-offs for different batch values. For example, for batch size 8, setting the GPU to DLA transition point to layer 39 would provide a decent trade-off between energy and performance, if they are equally important.

V. Additional Related Work

A popular method to improve latency per energy efficiency of neural networks is to perform automated hardware mapping for DNNs [13]. Kao et al. take tiling, ordering, and parallelism into account by using genetic algorithms. Energy-efficient applications mostly favor heterogeneous architectures. Some energy-optimized scheduling on heterogeneous environments are performed mainly for real-time tasks [14]–[16]. The main goal of such works is to minimize energy consumption as much as possible by considering the real-time latency constraints. Yang et al. [17] provides a framework in deep learning applications by considering errors and data for both energy efficiency and latency on CPUs and GPUs of Jetson Tx2 devices. A study [18] on the behavior of popular CNNs presents a workload characterization performed for various energy-efficient implementations on CPUs and GPUs. While many focus on how to improve energy efficiency, Martin et al [19] generates an estimation method on the degree of energy consumption for machine learning applications. They also classify the existing energy consumption methods for general and machine learning applications. However, none of these work consider time and energy as a trade-off metric in multi-accelerator systems and they merely focus on improving energy aspects of the DNNs on single accelerator systems.

To increase the performance of neural networks, some researchers have primarily focused on data parallelism, splitting the data between available devices. The performance of collaboration of CPU and GPU can beat the performance of a single PE [20], [21]. If the workload between them is well-distributed depending on the data size, a higher speed-up is achievable. Heterogeneous systems provide better performance and lower energy consumption when kernels are well-optimized for the target processing units [22]. Using FPGAs along with CPU and GPU for the same operation improves the results on energy consumption and energy-delay. Model and data parallelism techniques are integrated into a hybrid parallelism method so that the communication bottleneck is minimized [23]. This work applies a dynamic algorithm search as a partition method for layers between accelerators. However, these works do not consider optimization opportunities, such as layer fusion, and energy as a metric while applying workload distribution among PEs.

Using multiple accelerators for high performance NNI has recently been investigated by a few studies only. PREMA [24] provides a priority-based NNI execution technique to enable preemptive execution. Scheduling the workloads with a layer-aware method for multiple accelerators is explored in [25]. Yeh et. al [26] proposes increasing throughput by considering hard-deadlines of the overall system with a latency-aware method However, these works only consider improving execution time and do not consider energy in their optimizations.

VI. Conclusion

This study provides an exploration of energy/performance trade-off on DNN models in diversely heterogeneous mobile SoCs. We show that different accelerators in the system show varying energy/performance benefits for different set of layers in a neural network. We investigate multi-accelerator execution by exploring the resulting effects of different inter-accelerator transition points. We also introduce a metric, called MAEG, to represent the energy/performance trade-off between two accelerators, as a single optimizable value.

Acknowledgements

This material is based upon work supported by the National Science Foundation (NSF) under Grant No. CCF-2124010. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

REFERENCES

[1] R. Pujol, H. Tabani, L. Kosmidis, E. Mezzetti, J. Abella, and F. Cazorla, "Generating and exploiting deep learning variants to increase heterogeneous resource utilization in the nvidia xavier," in *ECRTS*, 2019.

[2] Z. Houssam-Eddine, N. Capodieci, R. Cavicchioli, G. Lipari, and M. Bertogna, "The hpc-dag task model for heterogeneous real-time systems," *IEEE Transactions on Computers*, pp. 1–1, 2020.

[3] NVIDIA, "Tensorrt," 2021. [Online]. Available: https://developer.nvidia.com/tensorrt

[4] "Nvidia deep learning accelerator," http://nvdla.org/, (Accessed on 09/17/2021).

[5] A. Karbachevsky, C. Baskin, E. Zheltonozhskii, Y. Yermolin, F. Gabbay, A. M. Bronstein, and A. Mendelson, "Early-stage neural network hardware performance analysis," *Sustainability*, vol. 13, no. 2, 2021. [Online]. Available: https://www.mdpi.com/2071-1050/13/2/717

[6] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, vol. 15, no. 11, pp. 1427–1436, 1989.

[7] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, "Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 6, pp. 911–924, 2010.

[8] Y. Xu, K. Li, J. Hu, and K. Li, "A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues," *Information Sciences*, vol. 270, pp. 255–287, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002002551400228X

[9] D. Franklin, "Deploying deep learning," https://github.com/dusty-nv/jetson-inference, 2016-2019.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[11] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Q. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: an automated end-to-end optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, A. C. Arpaci-Dusseau and G. Voelker, Eds. USENIX Association, 2018, pp. 578–594. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/chen

[12] N. Keskar, J. Nocedal, P. Tang, D. Mudigere, and M. Smelyanskiy, "On large-batch training for deep learning: Generalization gap and sharp minima," 2017, 5th International Conference on Learning Representations, ICLR 2017.

[13] S. C. Kao and T. Krishna, "Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[14] H.-E. Zahaf, A. E. H. Benyamina, R. Olejnik, and G. Lipari, "Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms," *Journal of Systems Architecture*, vol. 74, pp. 46–60, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138376211730019X

[15] Y. Gao, L. Han, J. Liu, Y. Robert, and F. Vivien, "Minimizing energy consumption for real-time tasks on heterogeneous platforms under deadline and reliability constraints," Inria - Research Centre Grenoble – Rhône-Alpes, Research Report RR-9403, Apr. 2021. [Online]. Available: https://hal.inria.fr/hal-03202996

[16] L. Zhou, L. N. Bhuyan, and K. K. Ramakrishnan, "Gemini: Learning to manage cpu power for latency-critical search engines," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 637–349.

[17] B. Yang, X. Cao, C. Yuen, and L. Qian, "Offloading optimization in edge computing for deep-learning-enabled target tracking by internet of uavs," *IEEE Internet of Things Journal*, vol. 8, no. 12, pp. 9878–9893, 2021.

[18] D. Li, X. Chen, M. Becchi, and Z. Zong, "Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus," in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, 2016, pp. 477–484.

[19] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75–88, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731518308773

[20] M. A. Guzman, R. Nozal, R. Tejero, M. Villarroya-Gaudó, D. S. Gracia, and J. L. Bosque, "Cooperative cpu, gpu, and fpga heterogeneous execution with enginecl," *The Journal of Supercomputing*, vol. 75, pp. 1732–1746, 2019.

[21] R. Nozal, J. L. Bosque, and R. Beivide, "Enginecl: Usability and performance in heterogeneous computing," *Future Gener. Comput. Syst.*, vol. 107, pp. 522–537, 2020.

[22] P. Pandit and R. Govindarajan, "Fluidic kernels: Cooperative execution of opencl programs on multiple heterogeneous devices," in *CGO '14*, 2014.

[23] L. Song, J. Mao, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "Hypar: Towards hybrid parallelism for deep learning accelerator array," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019, pp. 56–68.

[24] Y. Choi and M. Rhu, "Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units," *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 220–233, 2020.

[25] Y. H. Oh, S. Kim, Y. Jin, S. Son, J. Bae, J. Lee, Y. Park, D. U. Kim, T. J. Ham, and J. W. Lee, "Layerweaver: Maximizing resource utilization of neural processing units via layer-wise scheduling," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 584–597.

[26] T. T. Yeh, M. D. Sinclair, B. M. Beckmann, and T. G. Rogers, "Deadline-aware offloading for high-throughput accelerators," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 479–492.