AxoNN: Energy-Aware Execution of Neural Network Inference on Multi-Accelerator Heterogeneous SoCs

Ismet Dagli
Colorado School of Mines
Computer Science Department
Golden, CO, USA
ismetdagli@mines.edu

Alexander Cieslewicz
Colorado School of Mines
Computer Science Department
Golden, CO, USA
acieslewicz@mines.edu

Jedidiah McClurg
Colorado School of Mines
Computer Science Department
Golden, CO, USA
mcclurg@mines.edu

Mehmet E. Belviranli
Colorado School of Mines
Computer Science Department
Golden, CO, USA
belviranli@mines.edu

Abstract

The energy and latency demands of critical workload execution, such as object detection, in embedded systems vary based on the physical system state and other external factors. Many recent mobile and autonomous System-on-Chips (SoC) embed a diverse range of accelerators with unique power and performance characteristics. The execution flow of the critical workloads can be adjusted to span into multiple accelerators so that the trade-off between performance and energy fits to the dynamically changing physical factors.

In this study, we propose running neural network (NN) inference on multiple accelerators of an SoC. Our goal is to enable an energy-performance trade-off by distributing layers in a NN between a performance- and a power-efficient accelerator. We first provide an empirical modeling methodology to characterize execution and inter-layer transition times. We then find an optimal layers-to-accelerator mapping by representing the trade-off as a linear programming optimization constraint. We evaluate our approach on the NVIDIA Xavier AGX SoC with commonly used NN models. We use the Z3 SMT solver to find schedules for different energy consumption targets, with up to 98% prediction accuracy.

1 Introduction

Computing devices are becoming highly heterogeneous with the increased utilization of domain specific accelerators (DSAs), each of which is optimized to perform a specific type of operation. This trend is fueled by the need of running applications that span a diverse set of computations for emerging fields such as artificial intelligence, machine learning, and autonomous systems. The latest generation of SoCs —such as NVIDIA's Xavier and Orin architectures, Apple's M1 and A15 Bionic chips, and Qualcomm's Snapdragon 8 SoCs— have dramatically increased the degree of architectural heterogeneity within the same die. In such systems, dozens of DSAs with diverse instruction set architectures (ISAs) work together to accelerate operations (*i.e.*, kernels or tasks in an application) that belong to emerging application domains.

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

An emerging architectural feature of such heterogeneous SoCs is a *shared system memory* that all DSAs and CPU can directly access and utilize. While this design choice is primarily motivated by the goal of reducing chip area and production costs, it also helps in eliminating additional data transfer overhead between the host and the device [4]. Having shared memory directly accessible by every DSA in the system enables assigning OPs in a workload to the DSAs more flexibly. This flexibility also enables *collaborative execution* in shared-memory heterogeneous system architectures (SM-HSA), where OPs in a workload can be executed on different DSAs [8] to exploit the varying benefits (*i.e.*, energy, throughput, latency, etc.) that different types of DSAs can optimally provide—*e.g.*, a convolution operation can be accelerated by a GPU for high performance, or by a DLA for better energy efficiency. In such SM-HSAs, near-optimal utilization of the system resources heavily relies on carefully assigning the OPs to the available DSAs based on the target performance and power goals of a given scenario [15].

Collaborative execution of popular workloads, such as neural network (NN) inference, on different types of DSAs is a relatively new and unexplored scheme which has the potential to provide unique benefits for budgeted execution scenarios. To demonstrate the feasibility of collaborative execution for achieving different performance and energy goals on a heterogeneous platform, we conduct an exploratory experiment, which is shown in Fig. 1. In this experiment, we map the layers of the VGG-19 [21] network to the

DAC '22, July 10–14, 2022, San Francisco, CA, USA

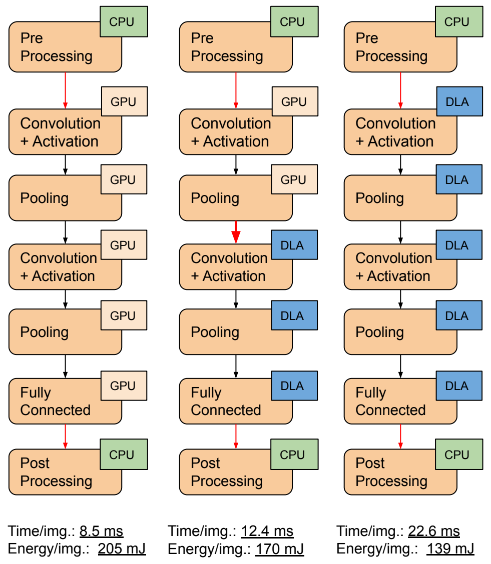

Figure 1: Simplified layer mappings for VGG-19 when executed with TensorRT on Xavier AGX: Leftmost and rightmost control flow graphs (CFG) show the traditional methods of executing the NN on a single type of DSA. The multi-accelerator execution shown in the middle employs a transition between DSAs in the execution flow to produce a schedule with a trade-off between latency and energy.

GPU and the DLA of NVIDIA's Xavier AGX SoC in three different ways. The left-most and right-most columns in the figure show where all layers are executed, either on the GPU or the DLA, respectively. The middle column illustrates a *collaborative* execution where the first n layers are run on the GPU, and the remaining m layers on the DLA. The total execution time and energy consumed are given under each column. Experimental results show that running all layers on the GPU results in the fastest execution time, whereas running all layers on the DLA is the most energy-efficient. On the other hand, the collaborative execution scheme shown in the middle results in a trade-off between execution time and energy, as more layers of the network are executed on the DLA.

A hybrid (i.e., GPU+DLA) execution scheme could be more feasible in real-life scenarios, when there is an energy constraint in the system. For example, when an autonomous aerial drone is running low on battery, scheduling of the NN layers to DSAs can be adjusted at the expense of a higher execution time (i.e., latency), hence resulting in a lower images/second detected by the NN. For the scenario given in Fig. 1, if the remaining energy budget per image detected is less than 205 milijoules, (total energy/image needed for GPU-only execution), but more than 139 milijoules, rather than running the entire NN on the DLA, choosing the GPU+DLA hybrid schedule in the middle will result in a more feasible operation. *The drone will still be able to complete its flight, but will be more responsive to the surrounding objects*, thanks to the lower latency (12.4 ms per image) achieved by a GPU+DLA hybrid execution against a DLA-only execution (22.6 ms per image).

While the most [6, 12, 14, 19, 25, 27] focus on improving the total throughput by using multiple DSAs concurrently, only a few [1, 23] have investigated the performance-energy trade-off in limited aspects. Besides, a limited number of studies [3, 5, 10] explore the benefits of using different types of DSAs collaboratively for the same application. To the best of our knowledge, none of the existing studies are able to address all of the following challenges altogether for multi-DSA collaborative execution:

- Holistic modeling of multi-accelerator execution that takes both the execution and data-transfer costs between different type of DSAs into consideration.
- Tunable objective for power consumption which can be targeted while finding schedules with optimal execution time.
- Generalized layer-wise characterization methodology for finding performance and energy costs of neural network inference on multi-DSA systems.

In this study, we propose an energy-aware multi-DSA execution scheme for NN inference on heterogeneous SoCs. Our proposed scheme, *AxoNN*, uniquely enables setting an energy consumption target (ECT) and finds a NN-layers-to-DSA mapping that minimizes the total execution time under a given ECT. *AxoNN* utilizes a novel inter-accelerator transitional cost model to integrate the penalty of switching between DSAs into the cost function. Our scheme characterizes each layer in the network for each target DSA, and explores multiple transitions between layers to find schedules that satisfy the given ECT. We represent the scheduling problem as a constrained-objective optimization problem.

Our paper makes the following contributions:

- We present *AxoNN*, a multi-accelerator execution scheme for diversely heterogeneous SoCs, which finds schedules with near-minimal execution time for a given ECT.
- We propose a novel, empirical model-creation technique to represent the cost of inter-DSA transitions on a shared-memory heterogeneous SoC.
- We build cost models for estimating energy and execution times which uniquely take transition times and hardware-pipelined accelerator architectures into account.
- We evaluate *AxoNN* on the NVIDIA Xavier AGX SoC by using its embedded GPU and DLA. Our results show that our methodology can find near-optimal schedules with one or two inter-DSA transitions within up to 98% and 97% time and energy prediction accuracy, respectively, while staying under the given ECT.

2 Multi-accelerator NN Inference on Diversely Heterogeneous SoCs

In NN inference, finding the desired trade-off requires a careful distribution of layers onto accelerators. However, using multiple DSAs to maximize the system's utilization while staying under resource constraints, such as ECT, introduces a number of challenges and considerations.

2.1 Challenges

Lack of flexibility in layer-to-DSA assignment: Each DSA has a different set of restrictions in terms of their capabilities for running different OPs. For example, even though layer activation functions are considered as separate layers on TensorRT layer set, TensorRT scheduler does not allow assigning activation layers and other layers separately between DSAs. NVIDIA's DLA has additional restrictions on layer parameters and batch sizes. Moreover, TensorRT does not allow to transitions from DLA to GPU after certain (*e.g.*, Eltwise) layers . Such limitations force some layers to fall back to

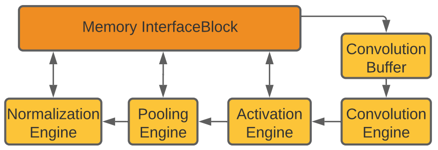

Figure 2: A simplified internal block diagram for NVIDIA's DLA.

the GPU, even though they are supposed to execute on DLA. Therefore, the flexibility on the potential inter-DSA transition points is restricted, and depends on the architecture of a NN, and on DSAs.

Grouping layers: Operator fusion has become a commonly applied optimization by popular frameworks, such as TensorRT [18] and TVM [2], that minimizes the cache misses between OPs and eliminates the duplicate OPs. Breaking potentially-fusible operations will increase execution time and, as a result, energy usage.

Profiling: Some highly-specialized accelerators —such as DLAs— run the consecutively-assigned layers as a single black box and do not allow internal profiling of execution times layer-by-layer. This limitation makes empirical modeling more challenging, since it presents an obstacle to fine-grained performance characterization.

2.2 Considerations

Inter-DSA transition overhead: On shared-memory SoCs, caches are often private to DSAs, due to complexity of cache coherency across diversely heterogeneous processing units. When the execution flow switches from one DSA to another on a shared memory system, which we call as *transition points*, the transient data present in private caches or buffers of DSAs needs to be written back to the shared memory. Such additional memory read/write operations need to be considered as transition overhead and added to the total execution time. The size of the memory pages being written or read is the primary factor determining the magnitude of this transition overhead. In most NNs, the size of the input and output data that each layer consumes and produces often changes after each layer. Moreover, the internal memory hierarchy of each DSA affects the transition overhead differently, even though the amount of data being read or written by two DSAs is the same [10]. Therefore, modeling the cost of inter-DSA transitions requires a careful consideration of the data size, layer type and the DSA that the transition originates from and/or arrives into.

Execution time characterization: While the execution time of a specific NN layer on a given DSA primarily depends on the layer type and the input data size, the location of the data before the layer is also an important factor. Therefore, we model layer execution time and inter-DSA transition time separately. The cold cache misses issued by DSAs as they begin executing a layer after a transition requires a warm-up period for layer-by-layer characterization. Another important factor affecting the execution time is the existence of internal hardware pipelines. As shown in Fig. 2, NVIDIA's DLA architecture embeds a pipeline of internal engines for common layers, such as convolution, activation, and pooling, in the order that these layers often appear in NNs. The data between the engines are often forwarded with direct data buses, and separate characterization of such layers may result in incorrectly estimated layer execution times. For example, since the pooling layer reduces the amount of data being passed to the next layer, measuring the execution time of the convolution and activation layers separately from the pooling layer will result in a longer execution time than the case where these three layers are profiled together. Therefore, layer-characterization needs to take such HW behavior into account for HW-pipelined DSAs.

3 Modeling Methodology

Taking into account the challenges and constraints, this section explains the methodology we utilize to build our empirical model.

3.1 Modeling inter-DSA transition cost

For the size of data needed to perform read/write operations, the transition overhead will increase as the size of the data increases. The X-axes in Fig. 3.a and 3.b represent the tensor sizes, *i.e.*, the size of the output produced after convolution/pooling layers. The Y-axis represents the transition cost in milliseconds if any transition is applied after the convolution/pooling layers. For example, performing the transition operation for the convolution layer with a data size of 3MB can result in 14x more time overhead compared to a data size of 100KB on the DLA. Thus, it is clear that the transition overhead decreases as the data movement decreases on both of the devices in our experiments. Besides, DLA has a second private buffer specifically for convolution operations, which directly affects DLA's behavior for different data sizes. Since the buffer has a relatively more limited size, larger data necessitates data movement from afar rather than a private buffer.

3.2 Energy and performance characterization

Based on architectural restrictions, we check whether a layer can run on all DSAs, or can be marked as a transition layer by using *canRunOnDLA* and *markOutput* TensorRT API calls. For example, a ReLU activation layer cannot run by itself on the DLA, but merging a ReLU activation layer with a convolution layer enables running both of them on the DLA. All of these values are obtained via offline

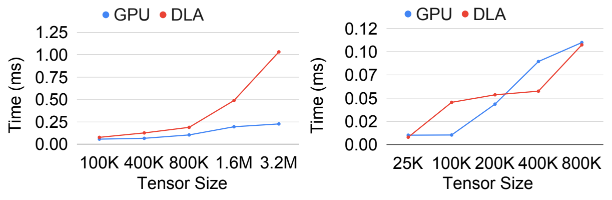

Figure 3: Empirical models for Convolution and Pooling layers' out-transition times after the layers are run on GPU and DLA. X axis indicates the tensor sizes.

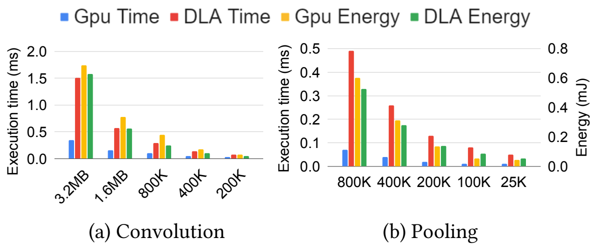

Figure 4: Empirical models for execution times of Convolution and Pooling layers on GPU and DLA for different tensor sizes.

Table 1: The notations used in this section.

Notation	Explanation
L_i	ith layer in a given layer set L, $0 \leq i \leq len(L)$
P_j	jth processor in a given processor set P, $0 \leq i \leq len(P)$
TR_i	Boolean variable set if a transition occurrs after L_i, $0 \leq i < len(L)$
$\tau(i, j, OUT)$	Output transition cost inflicted on P_i as the layer L_i transitions to P_j.
$\tau(i, j, IN)$	Input transition cost inflicted on P_j as the later L_i transitions from P_i.
$e(i, j)$	Energy consumption of layer L_i on processor P_j
$t(i, j)$	Execution time of layer L_i on processor P_j
ECT	Energy consumption target
S(L_i)	Scheduling set of layers, 0 < i < len(L)
$T(L, P, S, TR)$	Total time to execute a given layer set L, processor P, and schedule S
$E(L, P, S, TR)$	Energy consumption by a given set of layer L, processor P, and schedule S
$U(Li)$	The sub-unit executing the layer
$\gamma(L_i, s(L_i))$	Amount of time L_i saves by pipelining in its input
NumTransitions	Maximum amount of transitions allowed by user

profiler *IProfiler*, using an API call on TensorRT, to obtain execution time and energy consumption on GPU and DLA.

We analyze the execution behavior of the DSAs for different OPs by considering layer types and OP complexities, and measure the NN's resource utilization layer-by-layer. In Fig. 4(a-b), we measure execution time and energy consumption of different layers on VGG-19 by using GPU and DLA separately. The left vertical axis shows the execution time, whereas the right vertical axis represents the energy consumption. Depending on the data size, convolution OPs on GPU are 3x to 4.5x faster than on DLA, whereas pooling OPs on GPU are 3x to 7x faster than on DLA. For the energy comparison, convolution OPs on GPU consumes 1.1x to 1.8x more energy than on DLA, whereas the ratio for pooling on GPU over DLA varies from 0.6x to 1.15x.

4 Multi-accelerator Scheduling via Constraint-based Optimization

This section details how we build our cost functions and encode scheduling as a constraint-based optimization problem. First, we formulate the total execution time using empirical values found for transition and layer-wise execution times as shown in Section 3. Then, we assemble a constraint-based optimization problem that minimizes total execution time for a given ECT.

Table 1 lists the components needed to formulate our optimization problem. Layer set L is a NN-specific parameter which includes all layers L_i in the network with their sizes and activation functions. Processor set P represents all available processors and DSAs on the architecture. $\tau(i, j, OUT)$ is used to denote the OUT transition overhead inflicted on P_i as the transient data belonging to recently executed layer are flushed back to the system memory. In this case, P_j is the target DSA of the transition. Similarly, $\tau(i, j, IN)$ represents the IN transition overhead inflicted on P_j due to the cold cache misses caused by the initial memory instructions executed by P_j.

Since each layer can be mapped to a different processor, the final layer-to-processor schedule for a neural network is represented by:

$$S(L_i) = P_j \qquad where \quad 0 < i < m \quad \& \quad 0 < j < n \tag{1}$$

Since breaking fused operations and pipelined operations will cost extra time overhead, we consider another feature in our metholodogy, $pipeline(L_i, S(L_i))$ in Eq. 2. If the schedule does not prevent any OP from being pipelined, there will be no effect on the time and energy parameters. However, if the sub-unit used by the previous layer is not the same sub-unit on the current layer for the same DSA, it can severely affect execution time and energy results.

$$pipeline(L_i, S(L_i)) = \begin{cases} 0 & \text{if } U(L_i) = U(L_i - 1) \\ \gamma(L_i, s(L_i)) & \text{if } U(L_i) \neq U(L_i - 1) \end{cases} \tag{2}$$

After obtaining the related time parameters, the total execution time for a neural network $T(L, P, S(L \rightarrow P, TR)$ can be calculated via four different parameters, as shown in Eq. 3. Each layer L_i has a characteristic execution time on processor P_j, presented with a scheduling parameter $s(L_i)$. The transition cost $\tau(L_i, s(L_i), OUT|IN)$, is added to the total execution only if the result of layer L_i transitions to a new processor, represented by TR_i. Total energy consumption in Eq. (6) also has a closely similar pattern with the total execution time since the energy consumption is calculated by the same variables. We use TR_i boolean variable to indicate whether any transition occurs in Eq. (4) and to limit the number of transitions in Eq. (5).

$$\begin{aligned} T(L, P, S(L \rightarrow P), TR) = \sum_{i=0}^{len(L)} (t(L_i, s(L_i)) + \\ (TR_i \times \tau(L_i, s(L_i), OUT)) + (TR_i \times \tau(L_{i+1}, s(L_{i+1}), IN)) + \\ (TR_i \times pipeline(L_i, S(L_i)))) \end{aligned} \tag{3}$$

$$TR_i = \begin{cases} 1 & \text{if } S(i) \neq S(i+1) \\ 0 & \text{if } S(i) = S(i+1) \end{cases} \tag{4}$$

$$NumTransitions = \sum_{i=0}^{len(L)} TR_i \tag{5}$$

$$\begin{aligned} E(L, P, S(L \rightarrow P), TR) = \sum_{i=0}^{len(L)} e(L_i, s(L_i)) + \\ (TR_i \times e(L_i, s(L_i), OUT)) + (TR_i \times e(L_{i+1}, s(L_{i+1}), IN)) \end{aligned} \tag{6}$$

Our objective function is to minimize the execution time of a NN inference for a given set of layers and processors, with each possible mapping of layers to the processors, and an energy constraint ECT. Thus, we define our objective function and the primary constraint as follows:

$$\begin{aligned} \min \quad & T(L, P, S(L \rightarrow P)) \\ \text{s.t.} \quad & E(L, P, S(L \rightarrow P)) < ECT \end{aligned} \tag{7}$$

5 Evaluation

The constraints described in Section 4 can be handled with an off-the-shelf constraint solver. We solve the objective function and its dependencies shown in Section 4 with the Z3 SMT solver, which efficiently determines satisfiability of logical/numeric constraints.

5.1 Experimental setup

In this study, we use Nvidia's Jetson Xavier AGX SoC since it embeds a performance-efficient (*i.e.*, GPU) and an energy-efficient (*i.e.*, DLA) DSA, together with access to the same shared DRAM memory. The software versions utilized on our experimental platform are Ubuntu OS 18.04, Cuda 10.2, TensorRT 7.1.3, CuDNN 8.0.0, ONNX 1.6.0, and TensorFlow 2.3.1. We use the TensorRT engine to optimize the pre-trained models collected from several neural network models, VGG-16/19 [21], Resnet18/50 [7], Alexnet [13], and GoogleNet[22]. The reason we focus on these networks is that all layers in these networks can be scheduled on both the GPU and the DLA. This allows us to flexibly explore all possible layer-to-DSA assignments, without TensorRT engine falling back to GPUs.

5.2 Experimental Results

We design an experiment to observe the effect of transitions after different types of layers on a NN in Eq. 2. Since we mainly utilize two types of DSAs in our experiments (GPU and DLA), we first assign all layers to GPU, and measure the total execution time. Then, we repeat the experiment by applying a transition from GPU to DLA after the each layer on the neural network architecture and measure time on the DSAs. In order to analyze the effect of each transition, we subtract the time for each transition point from the previous transition experiment, and plot the results in Fig. 5. In other words, for each L_i assigned to a different device, we calculate the execution time difference according to Equation 3. Since the layer is running on a slower device (DLA), the difference in terms of the time is generally higher than zero in Fig. 5. The layer-wise execution time characterization given in Fig. 4 is inline with these results. The transitions after pooling layers have negative values. Since the effect of braking pipelines exceeds the slowdown of introduced by executing the next layer on DLA, the total execution time decreases. For this reason, applying a transition after such layers can provide an improvement in execution time.

We evaluate the feasibility of our model given in Section 4 with 6 different NN models. We define our objective functions and the main constraint on the Z3 solver as given in Equation 7. The main idea here is to restrict the energy with an upper limit by minimizing the execution time of NN inference. we provide the profiling results of layer execution time, energy, and transition characterization as inputs to the solver, and obtain the near-optimal schedule as output.

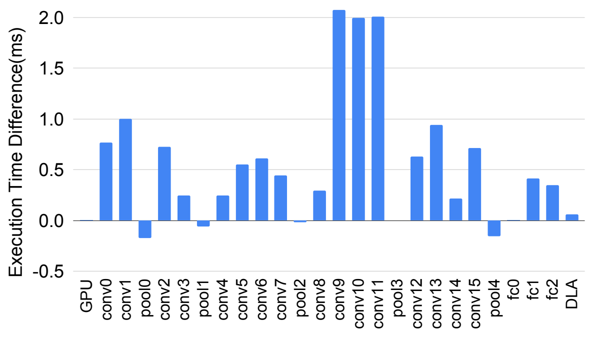

Figure 5: A transition is performed on the layer at the X-axis from GPU to DLA. As the number of layers increases on DLA, the execution time increases. Because the pipeline is broken, the transitions after pooling layers have negative values, as explained in Section 2.1

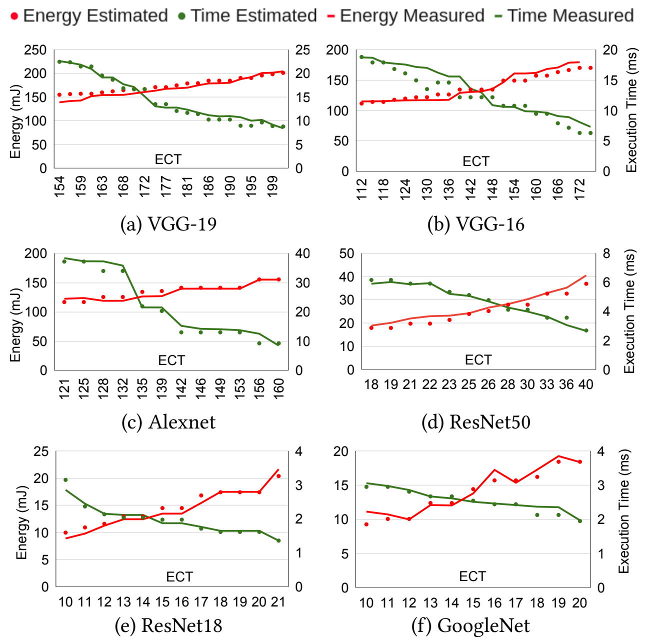

Figure 6: The comparison between the energy and execution times of the schedules estimated by *AxoNN* versus the actual energy and execution times measured for the corresponding actual runs. For each network we have targeted varying ranges of ECT that is between the energy consumption of all-DLA and all-GPU execution.

Our results are given in Fig. 6. X-axis shows the ECT constraint which is set within a range from minimum to maximum amount of energy spend by all-DLA and all-GPU executions, respectively. On the left and right vertical axes, we represent energy consumption and execution times, respectively, corresponding to each value of ECT. The values represented by green dots correspond to the execution time estimates whereas the green lines show the measured execution times when the schedule found by the solver for the specific ECT is executed. Similarly, the values represented by red dots and lines are for energy consumption. Overall, our results show that our model provides up to 97.1% accuracy on execution time prediction, and up to 98.2% accuracy on energy consumption prediction. In the worst cases, the execution time and energy prediction accuracy falls down to 78.1% and 71.9%, respectively.

5.3 Multi-Transition and Scheduling Overhead

While each transition between DSAs costs extra time in the schedule, the most feasible execution may still include multiple transitions, *i.e.*, going back and forth between DSAs. Therefore, we run our solver by allowing more than a single transition, by increasing the value of the *NumTransitions* variable (Eq. 5) to 3. Table 2 lists the number of inter-DSA transitions that the near-optimal schedules include. The overhead of scheduling (*i.e.*, solver execution time) is under 5 seconds when *NumTransitions* is set to 1, and under 1 minute when *NumTransitions* is set to 3.

6 Additional Related Work

GPipe [9] and PipeDream [16] split tasks between multi-DSAs by utilizing pipelines and considering transition time between accelerators for deep learning (DL) training. HetPipe [19] considers the first step of heterogeneity, and sets up multi-GPU clusters to apply

Table 2: The number of inter-DSA transitions that near-optimal schedules include when *NumTransitions* variable is set to 3.

Model	1 Transition	2 Transitions	3 Transitions
GoogleNet	19	3	0
VGG-19	16	4	2
VGG-16	18	4	0
Alexnet	10	2	0
ResNet50	8	4	0
ResNet18	9	3	0

the pipeline parallelism idea by maximizing utilization. However, none of the aforementioned works takes energy into account.

Narayanan et al. [17] propose round-robin scheduling for a target-latency deadline for DL workloads. Shamsa et al. [20] prioritize resource management over goals by considering dynamic changes. PCCS [26] presents an empirical model that formulates shared memory contention between multiple DSAs. However, these studies do not consider energy as a target or metric.

Pipelining in NN inference [24] is applied by distributing layers between CPU and GPU in order to maximize throughput of the system and synchronous data via cache-coherent interconnects. Kang et al. [11] optimize a single DL application's response time via the dynamic voltage and frequency scaling (DVFS) technique by finding the Pareto-optimal scheduling. Jeong et al. [10] offer a parallelization methodology for NN inference workloads to maximize throughput by leveraging TensorRT's GPU and DLA. However, none of these works considers using multiple accelerators for DL tasks to find a trade-off between energy and latency.

A similar idea to *AxoNN* is presented by MEPHESTO [15], which first characterizes the workloads on different DSAs, then models an energy-performance trade-off by collocating kernels and considering the memory contention on heterogeneous shared memory systems. However, this study does not take either the dependencies between OPs or the transition costs into account, and is therefore not suitable for NN inference. Barik et al. [1] propose a mapping algorithm between CPU and GPU by characterizing the power consumption of data-parallel specific workloads. Tzilis et al. [23] propose an online profiling model for estimating power consumption and performance under DVFS configuration for a given application. The aforementioned works do not consider challenges of executing DL applications on multiple DSAs.

To the best of our knowledge, our work is the first to apply layer-wise mapping on heterogeneous accelerators by considering both latency and energy.

7 Conclusion

This study presents *AxoNN*, a multi-accelerator execution scheme for heterogeneous SoCs. We explore the factors affecting the energy-aware scheduling of NN workloads onto DSAs. We analyze the transition costs between DSAs in a shared-memory system and characterize the execution time and energy consumption of different NN workloads. We build a scheduling model to find the minimum execution time for different energy targets. We test our methodology with 6 different networks, and test our results with the Z3 SMT Solver, obtaining up to 98% prediction accuracy.

Acknowledgements

We would like to thank the anonymous reviewers for their feedback. This material is based upon work supported by the National Science Foundation (NSF) under Grant No. CCF-2124010. Any opinions, findings, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

References

[1] Rajkishore Barik, Naila Farooqui, Brian T. Lewis, Chunling Hu, and Tatiana Shpeisman. 2016. A black-box approach to energy-aware scheduling on integrated CPU-GPU systems. In *CGO*.
[2] Tianqi Chen. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *OSDI*.
[3] Ismet Dagli and Mehmet E. Belviranli. 2021. Multi-accelerator Neural Network Inference in Diversely Heterogeneous Embedded Systems. In *RSDHA Workshop*.
[4] Marvin Damschen, Frank Mueller, and Jörg Henkel. 2018. Co-scheduling on fused CPU-GPU architectures with shared last level caches. In *IEEE TCAD*.
[5] Maria Angelica Davila Guzman. 2019. Cooperative CPU, GPU, and FPGA heterogeneous execution with EngineCL. In *The Journal of Supercomputing*.
[6] Li Han, Yiqin Gao, Jing Liu, Yves Robert, and Frederic Vivien. 2020. Energy-Aware Strategies for Reliability-Oriented Real-Time Task Allocation on Heterogeneous Platforms. In *ICPP*.
[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*.
[8] Sitao Huang. 2019. Analysis and Modeling of Collaborative Execution Strategies for Heterogeneous CPU-FPGA Architectures. In *ICPE*.
[9] Yanping Huang. 2019. GPipe: Efficient training of giant neural networks using pipeline parallelism. In *NIPS*.
[10] EunJin Jeong, Jangryul Kim, Samnieng Tan, Jaeseong Lee, and Soonhoi Ha. 2021. Deep Learning Inference Parallelization on Heterogeneous Processors with TensorRT. In *IEEE Embedded Systems Letters*.
[11] Duseok Kang, Jinwoo Oh, Jongwoo Choi, Youngmin Yi, and Soonhoi Ha. 2020. Scheduling of Deep Learning Applications Onto Heterogeneous Processors in an Embedded Device. In *IEEE Access*.
[12] Sheng-Chun Kao and Tushar Krishna. 2020. GAMMA: Automating the HW Mapping of DNN Models on Accelerators via Genetic Algorithm. In *ICCAD*.
[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*.
[14] Svetlana Minakova and Erqian Tang. 2020. Combining Task- and Data-Level Parallelism for High-Throughput CNN Inference on Embedded CPUs-GPUs MPSoCs. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*.
[15] Mohammad Alaul Haque Monil, Mehmet E. Belviranli, Seyong Lee, Jeffrey S. Vetter, and Allen D. Malony. 2020. MEPHESTO: Modeling Energy-Performance in Heterogeneous SoCs and Their Trade-Offs. In *PACT*.
[16] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: Generalized Pipeline Parallelism for DNN Training. In *SOSP*.
[17] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. 2020. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. In *OSDI*.
[18] NVIDIA. 2021. TensorRT. https://developer.nvidia.com/tensorrt
[19] Jay H. Park. 2020. HetPipe: Enabling Large DNN Training on (Whimpy) Heterogeneous GPU Clusters through Integration of Pipelined Model Parallelism and Data Parallelism. In *USENIX ATC*.
[20] Elham Shamsa, Anil Kanduri, Amir M. Rahmani, Pasi Liljeberg, Axel Jantsch, and Nikil Dutt. 2019. Goal-Driven Autonomy for Efficient On-chip Resource Management: Transforming Objectives to Goals. In *DATE*.
[21] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*.
[22] Christian Szegedy. 2015. Going deeper with convolutions. In *CVPR*.
[23] Stavros Tzilis, Pedro Trancoso, and Ioannis Sourdis. 2019. Energy-Efficient Runtime Management of Heterogeneous Multicores Using Online Projection. *TACO*.
[24] Hsin-I Wu, Da-Yi Guo, Hsu-Hsun Chin, and Ren-Song Tsay. 2020. A Pipeline-Based Scheduler for Optimizing Latency of Convolution Neural Network Inference over Heterogeneous Multicore Systems. In *AICAS*.
[25] Hongzhi Xu, Renfa Li, Chen Pan, and Keqin Li. 2019. Minimizing energy consumption with reliability goal on heterogeneous embedded systems. In *JPDC*.
[26] Yuanchao Xu, Mehmet Esat Belviranli, Xipeng Shen, and Jeffrey Vetter. 2021. PCCS: Processor-Centric Contention-Aware Slowdown Model for Heterogeneous System-on-Chips. In *MICRO-54*.
[27] Houssam-Eddine Zahaf, Abou El Hassen Benyamina, Richard Olejnik, and Giuseppe Lipari. 2017. Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms. *Journal of Systems Architecture*.