Master's Thesis

Efficient Machine Learning on Heterogeneous Computing Systems through a Coordinated Runtime System

Jihoon Hyun

Department of Computer Science and Engineering

Graduate School of UNIST

2019

Efficient Machine Learning on Heterogeneous Computing Systems through a Coordinated Runtime System

Jihoon Hyun

Department of Computer Science and Engineering

Graduate School of UNIST

Efficient Machine Learning on Heterogeneous Computing Systems through a Coordinated Runtime System

A thesis
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

Jihoon Hyun

June 11, 2019
Approved by

Advisor
Woongki Baek

Efficient Machine Learning on Heterogeneous Computing Systems through a Coordinated Runtime System

Jihoon Hyun

This certifies that the thesis of Jihoon Hyun is approved.

6. 11. 2019

signature

Advisor : Woongki Baek

signature

Myeongjae Jeon : Committee Member #1

signature

Young-ri Choi : Committee Member #2

Contents

List of Figures

List of Tables

Abstract

As machine learning grows, a heterogeneous computing system is actively used for a solution to increase the efficiency of machine learning. Although there are the prior studies for improving the efficiency of machine learning, the runtime support for heterogeneous computing system remains unexplored field. Our paper presents CEML, which is a runtime system to enhance the efficiency of machine learning on heterogeneous computing systems. CEML characterizes the machine-learning application in terms of the performance and power consumption at runtime, builds accurate the estimation models that estimate the performance and power consumption of the machine-learning application. CEML dynamically adapts the heterogeneous computing system to the efficient system state estimated to enhance the efficiency while satisfying constraints. We demonstrate the effectiveness of CEML by the evaluation in terms of the accuracy of estimators, the energy efficiency, the re-adaptation functionality, and runtime overheads on two full heterogeneous computing systems.

Chapter 1. Introduction

As machine learning grows, machine learning is actively used to solve complex problem like image classification, object detection and language translation. Although many of these machine-learning applications can be used in mobile and embedded devices, the large resource requirement including computation power, memory footprint and power consumption makes difficult to apply machine-learning to mobile and embedded devices. Heterogeneous computing systems are helpful to enhance the machine learning efficiency on mobile and embedded devices [12]. The heterogeneous computing system can execute the operations of machine-learning application concurrently using multiple heterogeneous computing devices that have different characteristics.

Prior works for machine learning efficiency have studied the system software [4, 12, 14], architectural support [8, 11, 15, 28]. The prior works are limited in that these works do not consider runtime support that controls and coordinates all the devices including memory in the target heterogeneous computing system [4, 12, 14], and/or OS kernel or GPU driver modification [5, 8, 9, 11, 15, 28]. These limitations make these works hard to be applicable to existing commodity heterogeneous computing systems [5, 8, 9, 11, 15, 28].

To tackle this problem, this paper presents CEML [13], a coordinated runtime system for efficient machine learning on heterogeneous computing systems. CEML profiles the target machine-learning application to analyze performance and power characteristics at runtime, and generates performance and power consumption estimation models. Based on the estimation models, CEML finds the efficient device frequencies to satisfy user-defined constraints and to maximize the metric using the exhaustive search and local search algorithms.

In this paper, we make the following contributions:

- This paper proposes CEML [13], a coordinated runtime system for improving the efficiency of machine-learning applications on heterogeneous computing systems. The performance and power estimators of CEML predict the performance and power consumption when the target application executes on the target heterogeneous computing system at given device frequencies. The runtime manager of CEML finds the efficient device frequencies for target machine-learning application on the target heterogeneous computing system based on the optimization metrics (e.g., performance under the power budget and energy optimization).

- We design CEML as a runtime system that does not need modification to the underlying hardware and system software including GPU driver or OS kernel. The exhaustive and local search algorithms are employed in CEML to find efficient device frequencies.

- The paper evaluates the accuracy of estimators of CEML, improvement of the energy efficiency with CEML versions, the re-adaptation functionality of CEML, and the utilization and performance overheads using various machine learning applications on the commodity heterogeneous computing systems. The results show that CEML effectively reduces energy consumption (i.e., 30.8% less on Jetson TX2) than the baseline version that configures the heterogeneous computing

devices to the maximum frequencies, which is a common environment in heterogeneous computing systems. Although CEML does not require per-application offline profiling, the energy efficiency improvements by CEML versions are similar to the improvement by the static best version that requires the offline profiling.

The remainder of this paper is organized as follows. Section 2 gives the background knowledge needed to understand this paper and the motivation for this work. Section 3 describes the experimental environment. Section 4 presents the design and implementation of CEML. Section 5 explains the experimental results of the proposed technique and its causes. Section 6 introduces related work and explains the differences form this paper. Section 7 describes the conclusions of this paper.

Chapter 2. Background and Motivation

2.1 Heterogeneous Computing

A heterogeneous computing system consists of more than two heterogeneous computing devices that have different characteristics. There are two types of heterogeneities for a heterogeneous computing system. Functional heterogeneity means that the heterogeneous computing devices differ in instruction-set architectures (e.g., GPU and CPU). Performance and power heterogeneity mean that the heterogeneous computing devices show different characteristics in terms of performance and power consumption.

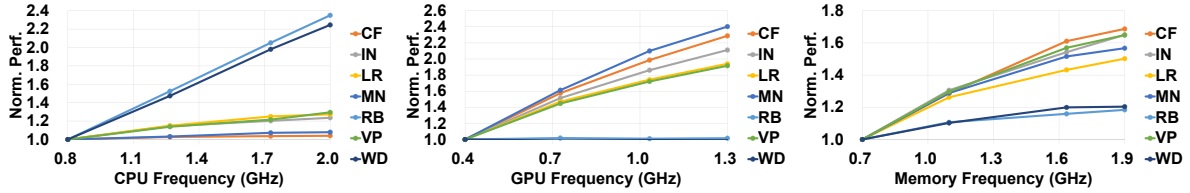
When a machine-learning application is executed on a heterogeneous computing system, the computing devices execute the machine-learning operations concurrently. The heterogeneous computing devices have dependency with execution order, and communicate data with other devices. In this work, it is assumed that the target heterogeneous computing system comprises CPU, GPU and memory as a single-chip processor. The communication between the heterogeneous computing devices is done with the main memory. This configuration assumed in this paper is commonly employed in the commodity heterogeneous computing systems involving mobile and embedded platforms [1, 2].

A heterogeneous computing device provides different voltage and frequency (V/F) levels. In this work, it is assumed that the heterogeneous computing devices including CPU, GPU and memory of the target system provides DVFS functionality and different V/F levels. CPU, GPU and memory provide N_{f_C}, N_{f_G}, and N_{f_M} V/F levels, respectively. The target heterogeneous computing system provides the interface to dynamically control V/F level of each device, and the V/F levels can be changed in small transition time. In this work, a *system state* is defined as a tuple that consists of the frequency of each device (i.e., (f_C, f_G, f_M)). We also define a *system state space* as the set including all the feasible system states of the target heterogeneous computing system.

2.2 TensorFlow

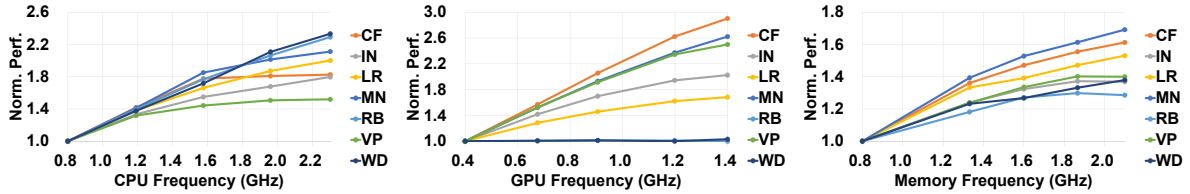
TensorFlow is a representative machine-learning development and execution framework [4]. A machine-learning algorithm is expressed as a dataflow graph using TensorFlow. A dataflow graph in TensorFlow comprises of multiple tensors as edges of a graph and operations as vertex of a graph. A tensor is a multidimensional array, which has elements that show one of the data types (e.g., float16, float32 or int32). The operation in a dataflow graph takes more then zero tensors as input and creates more than zero tensors as output [4].

The operations have dependencies according to dataflow graph. When all the input tensors are created, the operation can be executed. The scheduler of TensorFlow allocates the operation on the device based on the scheduling decision. The decision made by TensorFlow scheduler is based on various factor such as the functional constraint of each computing device, the current device utilization, the computational resource requirement of the operation, and programmer-specific scheduling decision [4].



(a) CPU frequency (b) GPU frequency (c) Memory frequency

Figure 2.1: Performance of the machine-learning applications with the device frequencies change on Jetson TX2



(a) CPU frequency (b) GPU frequency (c) Memory frequency

Figure 2.2: Performance of the machine-learning applications with the device frequencies change on Jetson Xavier

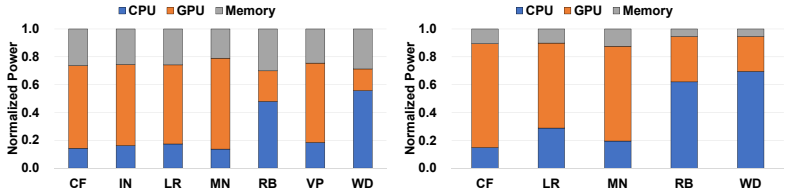
TensorFlow supports concurrent execution of independent operations across the available computing devices to enhance the utilization of the devices in the target system.

CEML focuses the efficiency of the training stage of the applications. In each step, a batch of data is consumed to train target machine-learning algorithm. The performance is defined as steps performed per second. It is assumed that the target machine-learning applications in this works are implemented as TensorFlow applications. While CEML is evaluated using TensorFlow applications, CEML can be applied in other machine-learning frameworks such as Caffe or PyTorch [14].

2.3 Motivation Example of CEML

Each machine-learning application shows widely different characteristics in terms of the performance and power consumption on heterogeneous computing systems. To motivate our work, we provide motivation examples using seven machine-learning benchmarks on two heterogeneous computing systems (i.e., Jetson TX2 and Jetson Xavier). Figures 2.1 and 2.2 show the performance characteristics of the evaluated machine-learning applications on Jetson TX2 and Jetson Xavier, respectively (see Section 3 for details). Figure 2.3 shows the power characteristics of evaluated benchmarks on Jetson TX2 and Jetson Xavier. As shown in Figures 2.1 and 2.2, the performance change of the machine-learning application with the device frequency while other device frequencies are fixed at maximum device frequencies on Jetson TX2 and Jetson Xavier, respectively. Figure 2.3 shows that the power consumption breakdown for each benchmark when all device frequencies are maximum frequencies. The data trends are observed as follow:

First, each benchmark shows widely different performance characteristics. For example, the performance sensitivity of CF on Jetson TX2 is relatively high to GPU frequency, and the performance



(a) Jetson TX2 (b) Jetson Xavier

Figure 2.3: Power consumption of each device normalized to total power consumption on Jetson TX2

sensitivity of `WD` on Jetson TX2 is relatively high to CPU frequency. However, the performance of `CF` and `WD` are insensitive to CPU frequency and GPU frequency, respectively. We also observe that the performance sensitivity of a benchmark differs depending on the heterogeneous computing systems. As shown in Figures 2.1 and 2.2, The performance of `MN` is insensitive to CPU frequency on Jetson TX2. On the contrary, the performance sensitivity of `MN` is relatively high to CPU frequency on Jetson Xavier.

Second, each benchmark shows widely different power consumption characteristics. For example, the GPU consumes the most significant power among CPU, GPU and memory with `CF` on Jetson TX2, whereas CPU consumes most significant power among the three devices with `WD` on Jetson TX2. We also observe that the power characteristics of a benchmark on each heterogeneous computing system is different. For example, in case of `CF`, GPU consumes more significant power on Jetson Xavier than Jetson TX2. In contrast, in case of `LR`, CPU has more power proportion on Jetson Xavier than Jetson TX2.

The observed data trends demonstrate that the software support for efficient machine learning on heterogeneous computing systems needs to control the devices coordinately. The machine-learning application shows different characteristics in terms of performance and power consumption, and same machine-learning application shows different characteristics depending on the heterogeneous computing system. Therefore, the static approaches that require extensive offline profiling are impractical for real-world heterogeneous computing systems. Because the heterogeneous computing systems have large system state space, and each benchmark has different characteristics in terms of performance and power consumption.

Chapter 3. Methodology

In this work, we employ two commodity heterogeneous computing systems that are NVIDIA Jetson TX2 and NVIDIA Jetson Xavier [1] as full heterogeneous computing system. NVIDIA Jetson TX2 includes the ARMv8 architecture dual-core Denver processor, the Pascal architecture 256-core GPU, and 8GB LPDDR4 memory. NVIDIA Jetson Xavier is equipped with the octa-core CPU based on the ARMv8.2 architecture, the Volta architecture 512-core GPU, and 16GB LPDDR4 memory. The evaluated V/F ranges of CPU, GPU and memory on Jetson TX2 are 0.81–2GHz, 0.42–1.3GHz, and 0.67–1.87GHz, respectively. The evaluated V/F ranges of CPU, GPU and memory on Jetson Xavier are 0.81–2.27GHz, 0.42–1.38GHz, 0.8–2.13GHz, respectively. Ubuntu 16.04 (kernel 4.4.38) and Ubuntu 18.04 (kernel 4.9.140) are installed in Jetson TX2 and Jetson Xavier, respectively.

The heterogeneous computing systems (i.e., Jetson TX2 and Jetson Xavier) evaluated in this paper include the CPU, GPU, and memory power consumption sensors that provide the power consumption of each device. CEML profiles the power consumption data and builds the power estimation models based on these sensors. Jetson TX2 and Jetson Xavier also provide the utilization information of each device through the Linux file system.

Seven machine-learning benchmarks provided in TensorFlow official repository are used in characterization and evaluation in this paper. (i.e., CIFAR-10 (`CF`) with 4000 steps, ImageNet (`IN`) with 2000 steps, Learning to Remember Rare Events (`LR`) with 1000 steps, MNIST (`MN`) with 30000 steps, REBAR (`RB`) with 10136 steps, Video Prediction (`VP`) with 1000 steps, and Wide & Deep (`WD`) with 32550 steps) [3].

Chapter 4. Design and Implementation

CEML consists of the three main units; the *performance estimator*, the *power estimator*, and the *runtime manager*. The performance and power estimators of CEML predict the performance and power consumption of when the target machine-learning application executes on the target heterogeneous computing device at given system state. The runtime manager profiles the data samples provided by the application and the sensors. The runtime manager derives the coefficients of estimation models including the performance and the device utilization estimation models, and configures the system to the efficient system state for target machine learning application.

We design CEML while applying the following design principles. First, the V/F levels of the devices in the target heterogeneous computing system are managed in a coordinated manner to enhance the efficiency of target machine-learning application on the heterogeneous computing system. We design CEML as a versatile system to support various optimization metrics and constraints (e.g., EDP, performance optimization under the power limit and energy optimization). CEML does not requires per-application offline profiling. To minimize potential overhead, we design and implement CEML as a lightweight system.

4.1 Performance Estimator

The *performance estimator* predicts the performance of the machine-learning application based on the performance estimation model for a target machine learning application on a heterogeneous computing system at a systems state. We define the performance as the *steps performed per unit time* of target machine-learning application.

To design the performance estimation model, we profile the performance characteristics of each benchmark. First, we characterize the performance sensitivity to the frequency of each device. Figures 2.1 and 2.2 show the performance sensitivities to the frequency of each device on Jetson TX2 and Jetson Xavier, respectively. In Figures 2.1 and 2.2, the other devices are configured to the maximum frequency. We observe that the performance of a machine-learning benchmark shows linear proportionality to the frequency of each device.

We also observe that the performance sensitivity to the device frequency depends on the frequencies of other devices. For example, Figure 4.2 shows the performance sensitivities of GPU-sensitive benchmarks to GPU frequency at maximum and minimum memory frequencies on Jetson TX2. We omit the result on Jetson Xavier because they show similar behavior. As shown in Figure 4.2, the performance sensitivities to GPU frequency decreases as memory frequency decreases. It is mainly because as memory frequency decreases, the memory becomes the bottleneck of the overall performance. This example demonstrates that the performance sensitivity to the frequency of each device depends on other device frequencies.

The performance estimator provides Equation 4.1 to predict the performance of target machine-learning application for given system state based on aforementioned observations. f_C, f_G and f_M mean

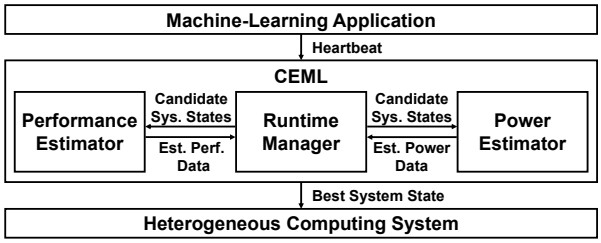


Figure 4.1: Overall architecture of CEML

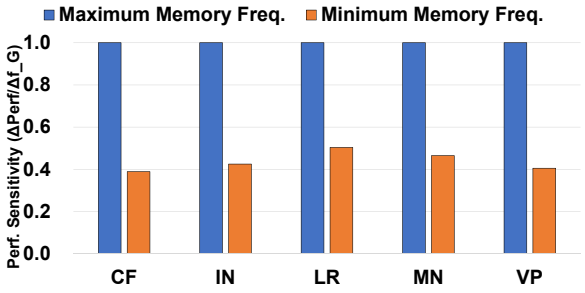


Figure 4.2: Performance sensitivity to GPU frequency at maximum and minimum memory frequencies

CPU, GPU and memory frequency, respectively. The system state is a tuple including CPU, GPU, and memory frequency (i.e., (f_C, f_G, f_M)). Equation 4.1 consists of a linear term for each device frequency and a quadratic term for each two devices. A linear term represents linear proportionality to frequency of each device. A quadratic term represents each performance sensitivity dependency for each two devices.

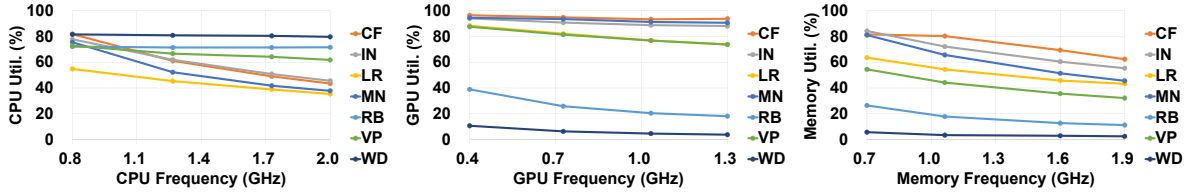
$$\begin{aligned} Perf =& \alpha_C \cdot f_C + \alpha_G \cdot f_G + \alpha_M \cdot f_M + \alpha_{C,G} \cdot f_C \cdot f_G + \\ & \alpha_{G,M} \cdot f_G \cdot f_M + \alpha_{M,C} \cdot f_M \cdot f_C + \beta \end{aligned} \tag{4.1}$$

Equation 4.1 has seven coefficients. Therefore, CEML needs to profile seven performance data samples that are collected with different system stats to derive the coefficients in Equation 4.1. CEML profiles seven performance data samples at runtime. We will discuss how CEML profiles the performance data at runtime in Section 4.3.

4.2 Power Estimator

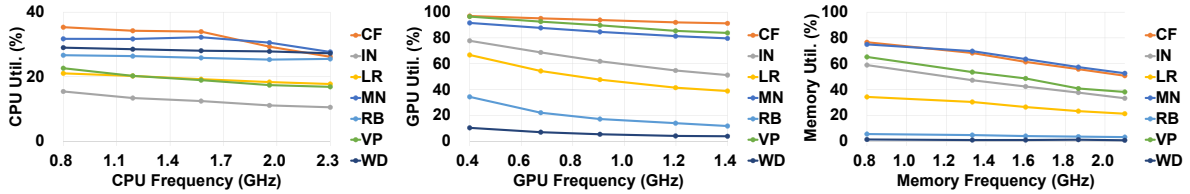
The *power estimator* predicts the power consumption of each device when the target machine-learning application is executing at the device frequencies. We assume that the power consumption of each device shows linear proportionality to the utilization of each device. This assumption is used in other works, and it is accurate and simple [25, 29].

CEML employs Equation 4.2 to predict the device power consumption (i.e., the device D power consumption for given system state). D is the target device, and f_D is the frequency of the device. ϵ_{D,f_D} and ζ_{D,f_D} are experimentally determined based on offline profiling. We note that this offline profling is performed only once in a heterogeneous computing system. To determine the coefficient of Eqaution 4.2, we develop the stress benchmarks for each device to stress each device for target utilization, respectively.



(a) CPU frequency (b) GPU frequency (c) Memory frequency

Figure 4.3: Device utilization with each device frequency on Jetson TX2



(a) CPU frequency (b) GPU frequency (c) Memory frequency

Figure 4.4: Device utilization with each device frequency on Jetson Xavier

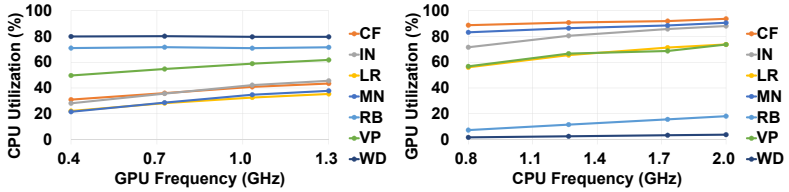
$$P_{D,f_D} = \epsilon_{D,f_D} \cdot U_D + \zeta_{D,f_D} \tag{4.2}$$

CEML needs to estimate the utilization of each device to estimate power consumption using Equation 4.2 for target system state. To make device utilization estimation model, we investigate the device utilization sensitivity of the benchmarks to the device or the other device frequency.

Figures 4.3 and 4.4 show that the device utilization sensitivity of the machine-learning application to the device frequency on Jetson TX2 and Jetson Xavier, respectively. We observe that each device utilization shows linear proportionality to each device frequency. It is because the device works shorter time when the device frequency is higher.

We also observe that the device utilization is affected by the frequency of other devices. Figure 4.5 shows that the device utilization sensitivities to other device frequencies on Jetson TX2. We omit the result on Jeston Xavier because of similar behavior. Figure 4.5 demonstrates that the device utilization of each device has linear proportionality to other device frequencies. It is mainly because the device takes more data per unit time when the other device is faster. It makes the device utilization higher. While omitted, there is utilization sensitivity interaction between each device pair similar to the aforementioned performance estimation model.

Based on observations, CEML employs Equation 4.3 to predict CPU utilization for target system state. CEML also uses similar equation to estimate GPU and memory utilization. The utilization estimation model employed in CEML consists of a first-order term for each device and a quadratic term for each device pair. The first-order term represents the device utilization sensitivity to each device frequency, and the quadratic term represents the device utilization sensitivity between each device pair.



(a) CPU util. vs. GPU freq. (b) GPU util. vs. CPU freq.

Figure 4.5: CPU (GPU) utilization sensitivity to GPU (CPU) frequency on Jetson TX2

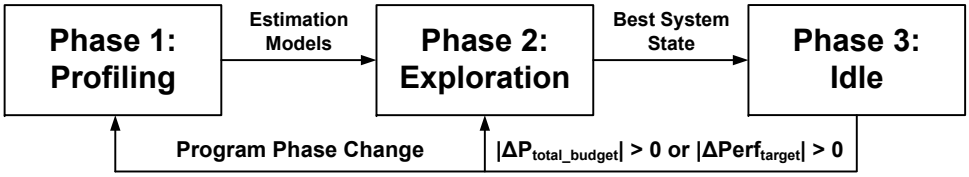


Figure 4.6: Overall execution flow of the runtime manager

$$U_C = \gamma_{C,C} \cdot f_C + \gamma_{C,G} \cdot f_G + \gamma_{C,M} \cdot f_M + \gamma_{C,CG} \cdot f_C \cdot f_G + \gamma_{C,GM} \cdot f_G \cdot f_M + \gamma_{C,MC} \cdot f_M \cdot f_C + \delta_C \quad (4.3)$$

To estimate CPU utilization for target system state using Equation 4.3, CEML needs to determine the coefficients of Equation 4.3. Because there are seven coefficients, CEML needs to collect seven utilization data samples. CEML collects the device utilization samples without offline profiling. We will discuss how CEML collects the device utilization samples at runtime in Section 4.3. The total power consumption of the heterogeneous computing system is determined by Equation 4.4. P_C, P_G and P_M are the power consumption of CPU, GPU, and memory, respectively.

$$P = P_C + P_G + P_M \quad (4.4)$$

4.3 Runtime Manager

The *runtime manager* of CEML collects the data samples including performance data generated by the application and the utilization data of each device. It then derives the coefficients of the performance and device utilization estimation models. It also determines the efficient system state based on the local or exhaustive search algorithms.

The runtime manger consists the three phases; profiling, exploration, and idle phases. Figure 4.6 exhibits the execution flow of the runtime manager of CEML.

During the *profiling phase*, the runtime manager collects the seven performance and device utilization samples with different system state. The runtime manager configures the heterogeneous computing system to sample system state. It then measures the execution time for small part of the training steps of the total training steps. The runtime manager also collects the device utilization sample of each device

Table 4.1: Seven system states profiled during profiling phase

System	Profiled system states (GHz)
Jetson TX2	(2, 1.3, 1.87), (1.42, 0.83, 0.67), (1.42, 0.42, 1.33), (0.81, 0.83, 1.33), (0.81, 0.42, 1.06), (0.81, 0.62, 0.67), (1.11, 0.42, 0.67)
Jetson Xavier	(2.27, 1.38, 2.13), (1.57, 0.91, 0.8), (1.57, 0.42, 1.6), (0.81, 0.91, 1.6), (0.81, 0.42, 1.33), (0.81, 0.68, 0.8), (1.19, 0.42, 0.8)

Algorithm 1 The exhaustive search function

```
1:  procedure EXPLORESYSTEMSTATESPACEWITHEXHAUSTIVESEARCH
2:      bestSystemState ← getInitialSystemState()
3:      bestEfficiency ← estimateEfficiencyOfSystemState(bestSystemState)
4:      for f_C ∈ F_C do
5:          for f_G ∈ F_G do
6:              for f_M ∈ F_M do
7:                  cSystemState ← (f_C, f_G, f_M)
8:                  cEfficiency ← estimateEfficiencyOfSystemState(cSystemState)
9:                  if cEfficiency > bestEfficiency ∧ checkConstraint(cSystemState) then
10:                     bestSystemState ← cSystemState
11:                     bestEfficiency ← cEfficiency
12:     configureSystemToSystemState(bestSystemState)
```

at the same time. This process repeats seven times to collect seven data samples with different system states. Table 4.1 shows the sample system states for each evaluated heterogeneous computing system. The sample system states are experimentally chosen to cover wide range of the device frequencies. Each data sample is collected during 1% of total training steps.

During the *exploration phase*, the runtime manager builds the performance and power estimation models based on collected data samples. Specifically, the runtime manager derives the coefficients of Equations 4.1 and 4.3.

After constructing the estimation models, the runtime manger explores the system state space to determine efficient system state using *exhaustive search algorithm* and *local search algorithm*. CEML is able to optimize various optimization scenarios based on user-defined metric. The exhaustive and local search algorithm that show different convergence system state and overheads employed in the runtime manger are used to explore the system state space.

Algorithm 1 shows the pseudocode of the exhaustive search algorithm. It explores the system state space in exhaustive manner. It keeps the best system state that is estimated to have best efficiency for target machine learning application while satisfying user-specific constraints among the explored system states. It starts with an initial system state (Line 2). The `getInitialSystemState` function returns certain system state that satisfy user-specific constraints. It estimates the efficiency of target system state, and if the target system state is more efficient than current state based on estimated efficiencies, it keeps the current state as the best system state (Line 9). When all the system states in system state space are explored, it ends the search algorithm. The time complexity of the exhaustive search algorithm is

Algorithm 2 The local search function

```
1:  procedure EXPLORESYSTEMSTATESPACEWITHLOCALSEARCH
2:      bestSystemState ← getInitialSystemState()
3:      bestEfficiency ← estimateEfficiency(bestSystemState)
4:      while true do
5:          cSystemState ← getBestNeighborSystemState(bestSystemState)
6:          cEfficiency ← estimateEfficiency(cSystemState)
7:          if cEfficiency > bestEfficiency then
8:              bestSystemState ← cSystemState
9:              bestEfficiency ← cEfficiency
10:         else
11:             break
12:     configureSystemToSystemState(bestSystemState)
```

relatively high (i.e., $O(N_{f_C} \cdot N_{f_G} \cdot N_{f_M})$). However, it is still useful for the commodity heterogeneous computing system that has small system state space.

The runtime manager provides the local search algorithm that has relatively low time complexity. The local search algorithm explores the system state space similar to the hill-climbing algorithm. The local search algorithm is shown in Algorithm 2. It starts with the initial system state (Line 2). It calculates the efficiencies of the neighbor system states, and chooses the best neighbor system state that is expected to maximize the user-defined metric while satisfying the constraints among the neighbor system state (Line 5). It compares the current best system state and the best neighbor system state, and then chooses the system state to be more efficient (Line 7-9). If the best neighbor system state has higher efficiency than current state, it repeats the process. Otherwise, it ends the search algorithm (Line 11).

When the search process is terminated, the runtime manager configures the heterogeneous computing system to the best system state that is estimated to maximize the efficiency for the target machine-learning application (Line 12 in Algorithm 1 and Line 12 in Algorithm 2).

During the *idle phase*, CEML monitors the application and system to detect the program phase changes such as training stage to evaluation phase. When the runtime manager detects the program phase change, it triggers the adaptation process that restart with profiling phase to determine a new efficient system state. The runtime manager also monitors the user-defined metric or constraints change. If the runtime manager detects the metric or constraints change, it transitions into the exploration phase to determine a new efficient system state based on new metric or new constraints.

Chapter 5. Evaluation

We evaluate the effectiveness of CEML in terms of the accuracy of estimators, the energy efficiency, the re-adaptation functionality of CEML, the runtime overheads.

First, we evaluate the estimation accuracy of CEML using seven machine-learning benchmarks. We collect 25 test datasets including performance and overall power consumption data for each benchmark with 25 different systems. The estimation error of each benchmark is the average estimation error of test datasets. Figures 5.1 and 5.3 show the average estimation error of each benchmark on Jetson TX2 and Jetson Xavier, respectively. CEML achieve high estimation accuracy across all evaluated benchmarks. The average performance and power estimation error across the benchmarks are 5.4% and 8.8% on Jetson TX2, and 6.3% and 9.9% on Jetson Xavier. This result demonstrates the performance and power estimation models employed in estimators of CEML effectively predict the performance and power consumption of the target machine-learning benchmarks. It is because the linear terms and quadratic terms are effective to model sensitivity to each device frequency and sensitivity dependency between devices in both the performance and power estimation models.

We evaluate the effectiveness of CEML in terms of energy consumption. We define the energy consumption as the Joules per training step. Each benchmark is evaluated in four versions. The baseline version works with the maximum CPU, GPU, and memory frequencies to execute each benchmark. The static best version chooses the most efficient system state based on extensive offline profiling. Because of experiment time issue, the offline profiling for the static best version collects data samples with 32 different system states. The `CEML-L` and `CEML-E` versions select the best system state with local and exhaustive algorithms provided in CEML, respectively.

Figures 5.2 and 5.4 show the energy consumption of the evaluated benchmarks with each version on Jetson TX2 and Jetson Xavier. We normalize the energy consumption of the versions to the baseline version. The geometric mean of each version is also shown at the rightmost bar.

The CEML versions effectively improve the energy efficiencies of target heterogeneous computing systems. For example, CEML-E reduces the energy consumption by 30.8% and 35.1% compared to the baseline version on Jetson TX2 and Jetson Xavier, respectively. The baseline version configures the system to the maximum device frequencies to execute the target machine-learning application without regarding the performance and power characteristics of the target applications. CEML versions select the best system state that is determined based on the search algorithms to enhance the energy efficiency of the target machine learning application, and controls all devices in a coordinated manner.

CEML versions consume relatively low energy comparable with the static best version. For example, CEML-E consumes 2.9% and 12.2% higher energy than the static best version. The static best version finds more efficient system state than CEML versions. However, the static best version takes too long time to get the efficient system state because it is based on extensive per-application offline profiling. CEML versions execute small portion of training step at suboptimal system state during profiling phase, it makes CEML version less efficient than the static best version even with same system state (e.g., `RB` on Jetson TX2 and `MN` on Jetson Xavier). CEML versions may find less efficient systems state because

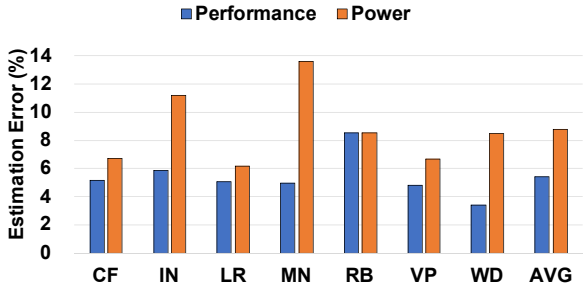


Figure 5.1: Estimation errors of the estimators on Jetson TX2

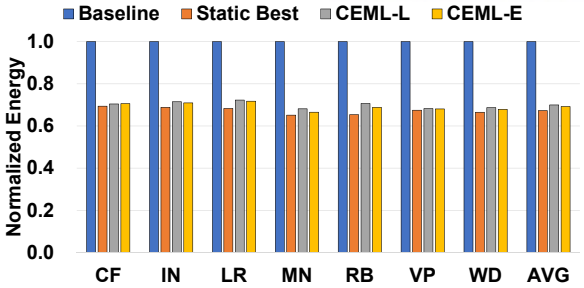


Figure 5.2: Energy consumption on Jetson TX2

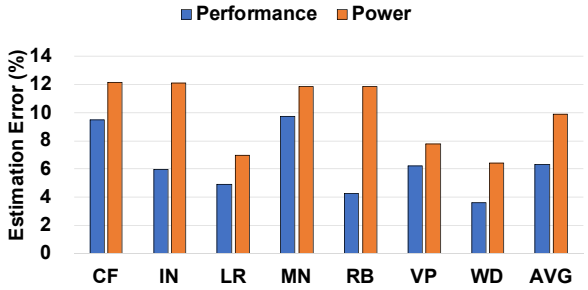


Figure 5.3: Estimation errors on Jetson Xavier

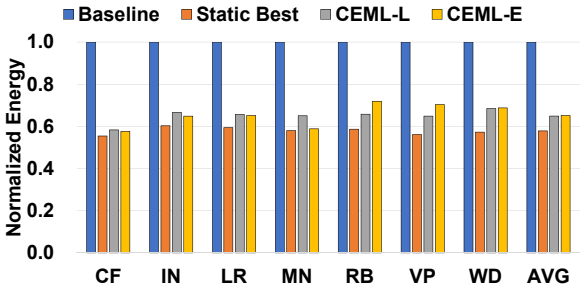


Figure 5.4: Energy consumption on Jetson Xavier

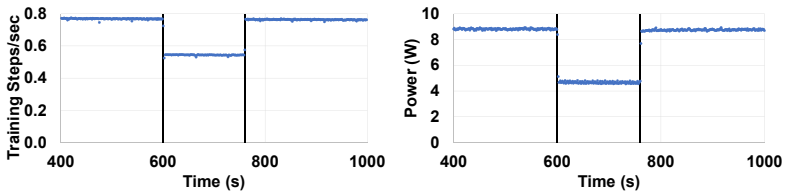
of estimation errors (`IN` and `LR` on Jetson TX2 and `RB` and `VP` on Jetson Xavier). Nevertheless, the improvement of the energy efficiency with CEML versions shown in Figures 5.2 and 5.4 exhibits the effectiveness of CEML without extensive offline profiling.

CEML-L achieves similar energy efficiency to CEML-E. For example, CEML-L consumes 2.6% higher energy than CEML-E on Jetson TX2. Because CEML-L may converge local optimal system state, sometimes CEML-L selects less efficient system state than CEML-E (e.g., `MN` on Jetson TX2 and `IN` and `MN` on Jetson Xavier). Because of estimation error, CEML-L may find more efficient system state than CEML-E (e.g., `RB` and `VP` on Jetson Xavier).

We also show an example scenario to demonstrate the effectiveness of the re-adaptation functionality of CEML. In this example scenario, the power constraint of the target heterogeneous computing system (i.e., Jetson TX2) changes while the execution of the benchmark (i.e., `LR`) with CEML. The metric in this evaluation is performance (i.e., steps performed per second).

Figures 5.5a and 5.5b show the performance and power consumption changes in this example scenario. The `LR` benchmark starts under the power constraint (i.e., $9W$) which is high. Because this power constraint is enough to execute the benchmark at the fastest system state (i.e., the maximum device frequencies for CPU, GPU, and memory), CEML configures the target heterogeneous computing system to this system state. As shown in Figures 5.5a and 5.5b the power constraint transitions to the power constraint (i.e., $5W$) which is low at $t = 600.1$. CEML effectively detects the constraint change, it triggers that the runtime manger restarts exploration phase to fine a new efficient system state under the power constraint, and adapts the target heterogeneous computing system to the new efficient system state satisfying the constraint. The total budget recovers to the previous power constraint (i.e., $9W$). CEML detects this change, and adapts the target heterogeneous computing system at $t = 760.1$.

Finally, we evaluate the runtime overheads of CEML in terms of the CPU utilization and the



(a) Performance (b) Power consumption

Figure 5.5: Effectiveness of re-adaptation

performance. CEML versions incur insignificant performance overheads. CEML makes small utilization with all the evaluated benchmarks (i.e., 1.0% on average). The performance overheads of the `CEML-L` and `CEML-E` versions are insignificant with all the evaluated benchmarks. Specially, the average exploration time of each CEML version is 7.2 and 176.4 microseconds and 35.7 and 827.2 microseconds on Jetson TX2 and Jetson Xavier, respectively.

Chapter 6. Related Work

Previous works have widely studied the system software and architectural supports to enhance the efficiency of heterogeneous computing systems [10, 20, 22, 21, 25, 29]. The prior works are helpful, however they only consider partial devices of heterogeneous computing devices (i.e., CPU [10, 20, 21, 29], CPU and GPU [22, 25], GPU and memory [27]), and do not consider machine-learning application characteristics.

Prior works have studied the techniques to improve the energy efficiency of symmetric multiprocessor [18, 19, 23, 24]. Although these techniques can be apply to heterogeneous computing systems, they lack the various heterogeneities of the heterogenous computing devices, and do not consider about the characteristics of the machine-learning applications.

Previous works have studied the techniques to optimize GPU in terms of the utilization, the performance, and the energy efficiency [6, 7, 16, 17, 26]. These techniques need the architectural modification for the GPU [16, 17, 26] or does not consider all the heterogeneous computing devices [6, 7].

Our work characterizes machine-learning applications in terms of the performance and power consumption, and coordinately controls all the devices in the target heterogeneous computing system. CEML needs no modification of the existing hardwares. We demonstrate the effectiveness of our work using two commodity heterogeneous computing systems (i.e., Jetson TX2 and Jetson Xavier).

Previous works have investigated the software supports to improve efficiency of machine learning [4, 12, 14]. They focus on the system software support with task scheduling for distributed and parallel systems to enhance the efficiency of machine learning [4, 12, 14]. We design and implement CEML as a runtime system that does not require OS or GPU driver modification, and focus on the heterogeneous computing systems.

There are previous works that designed and implemented hardware accelerators for efficient machine-learning [5, 8, 9, 11, 15, 28]. Although they are helpful to improve efficiency of machine-learning application, they cannot be applicable to existing systems without hardware modification. Because CEML is a runtime system that requires no hardware modification, CEML is directly applicable to commodity heterogeneous computing systems.

Chapter 7. Conclusions

In this work, we present CEML, a runtime system for efficient machine learning on heterogeneous computing systems in a coordinated way. CEML analyzes the target machine-learning application to characterize its performance and power consumption at runtime with small overheads. The runtime manager of CEML builds accurate the performance and power estimation models, effectively finds the efficient system state, and adapts the target heterogeneous computing system. We show experimental result to demonstrate the effectiveness of CEML in terms of the accuracy of the estimators, energy efficiency, the adaptation functionality, and runtime overheads. In the evaluation of CEML, CEML versions show the energy efficiency comparable with the static best version that is based on per-application offline profiling.

Bibliography

[1] http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html.

[2] http://www.samsung.com/semiconductor/products/exynos-solution/application-processor/EXYNOS-5-OCTA-5422.

[3] https://github.com/tensorflow/models.

[4] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[5] S. Angizi, Z. He, and D. Fan, "Dima: A depthwise cnn in-memory accelerator," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2018, pp. 1–8.

[6] Q. Chen, H. Yang, M. Guo, R. S. Kannan, J. Mars, and L. Tang, "Prophet: Precise qos prediction on non-preemptive accelerators to improve utilization in warehouse-scale computers," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17. New York, NY, USA: ACM, 2017, pp. 17–32. [Online]. Available: http://doi.acm.org/10.1145/3037697.3037700

[7] Q. Chen, H. Yang, J. Mars, and L. Tang, "Baymax: Qos awareness and increased utilization for non-preemptive accelerators in warehouse scale computers," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '16. New York, NY, USA: ACM, 2016, pp. 681–696. [Online]. Available: http://doi.acm.org/10.1145/2872362.2872368

[8] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 367–379.

[9] Q. Deng, L. Jiang, Y. Zhang, M. Zhang, and J. Yang, "Dracc: A dram based accelerator for accurate cnn inference," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18. New York, NY, USA: ACM, 2018, pp. 168:1–168:6. [Online]. Available: http://doi.acm.org/10.1145/3195970.3196029

[10] M. Han, J. Park, and W. Baek, "Chrt: A criticality- and heterogeneity-aware runtime system for task-parallel applications," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 942–945.

[11] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 243–254. [Online]. Available: https://doi.org/10.1109/ISCA.2016.30

[12] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang, "Djinn and tonic: Dnn as a service and its implications for future warehouse scale computers," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: ACM, 2015, pp. 27–40. [Online]. Available: http://doi.acm.org/10.1145/2749469.2749472

[13] J. Hyun, J. Park, K. Y. Kim, S. Yu, and W. Baek, "Ceml: a coordinated runtime system for efficient machine learning on heterogeneous computing systems," in *Euro-Par 2018: Parallel Processing*, ser. Euro-Par '18, M. Aldinucci, L. Padovani, and M. Torquati, Eds. Cham: Springer International Publishing, 2018, pp. 781–795.

[14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: ACM, 2014, pp. 675–678. [Online]. Available: http://doi.acm.org/10.1145/2647868.2654889

[15] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: ACM, 2017, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/3079856.3080246

[16] K. Y. Kim and W. Baek, "Blpp: Improving the performance of gpgpus with heterogeneous memory through bandwidth- and latency-aware page placement," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, Oct 2018, pp. 358–365.

[17] K. Y. Kim, J. Park, and W. Baek, "Iacm: Integrated adaptive cache management for high-performance and energy-efficient gpgpu computing," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, Oct 2016, pp. 380–383.

[18] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *Proceeding of the 41st Annual International*

Symposium on Computer Architecuture, ser. ISCA '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 301–312. [Online]. Available: http://dl.acm.org/citation.cfm?id=2665671.2665718

[19] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: ACM, 2015, pp. 450–462. [Online]. Available: http://doi.acm.org/10.1145/2749469.2749475

[20] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, "Hierarchical power management for asymmetric multi-core in dark silicon era," in *Proceedings of the 50th Annual Design Automation Conference*, ser. DAC '13. New York, NY, USA: ACM, 2013, pp. 174:1–174:9. [Online]. Available: http://doi.acm.org/10.1145/2463209.2488949

[21] J. Park and W. Baek, "Hap: A heterogeneity-conscious runtime system for adaptive pipeline parallelism," in *Proceedings of the 22Nd International Conference on Euro-Par 2016: Parallel Processing - Volume 9833*, ser. Euro-Par '16. New York, NY, USA: Springer-Verlag New York, Inc., 2016, pp. 518–530. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-43659-3_38

[22] ——, "Rchc: A holistic runtime system for concurrent heterogeneous computing," in *2016 45th International Conference on Parallel Processing (ICPP)*, Aug 2016, pp. 211–216.

[23] ——, "Quantifying the performance and energy-efficiency impact of hardware transactional memory on scientific applications on large-scale numa systems," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2018, pp. 804–813.

[24] J. Park, E. Cho, and W. Baek, "Rmc: An integrated runtime system for adaptive many-core computing," in *2016 International Conference on Embedded Software (EMSOFT)*, Oct 2016, pp. 1–10.

[25] A. Pathania, A. E. Irimiea, A. Prakash, and T. Mitra, "Power-performance modelling of mobile gaming workloads on heterogeneous mpsocs," in *Proceedings of the 52Nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: ACM, 2015, pp. 201:1–201:6. [Online]. Available: http://doi.acm.org/10.1145/2744769.2744894

[26] M. H. Santriaji and H. Hoffmann, "Merlot: Architectural support for energy-efficient real-time processing in gpus," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2018, pp. 214–226.

[27] A. Sethia and S. Mahlke, "Equalizer: Dynamic tuning of gpu resources for efficient execution," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 647–658. [Online]. Available: http://dx.doi.org/10.1109/MICRO.2014.16

[28] L. Song, Y. Wang, Y. Han, X. Zhao, B. Liu, and X. Li, "C-brain: A deep learning accelerator that tames the diversity of cnns through adaptive data-level parallelization," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 123:1–123:6. [Online]. Available: http://doi.acm.org/10.1145/2897937.2897995

[29] J. Yun, J. Park, and W. Baek, "Hars: A heterogeneity-aware runtime system for self-adaptive multithreaded applications," in *Proceedings of the 52Nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: ACM, 2015, pp. 107:1–107:6. [Online]. Available: http://doi.acm.org/10.1145/2744769.2744848