M.S. THESIS

Efficient Resource Management Technique for Edge Devices

엣지 디바이스에서의 효과적인 자원관리를 위한 기법

BY

최 용 준

FEBURARY 2021

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

M.S. THESIS

Efficient Resource Management Technique for Edge Devices

엣지 디바이스에서의 효과적인 자원관리를 위한 기법

BY

최 용 준

FEBURARY 2021

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Efficient Resource Management Technique for Edge Devices

엣지 디바이스에서의 효과적인 자원관리를 위한 기법

지도교수 엄 현 상

이 논문을 공학석사 학위논문으로 제출함

2020 년 12 월

서울대학교 대학원

컴퓨터 공학부

최 용 준

최 용 준의 공학석사 학위논문을 인준함

2021 년 01 월

Abstract

Edge computing is currently actively used for the benefits of low latency, reduced bandwidth usage due to efficient data processing, and improved security compared to server-centric computing. This has become a more important study as actively used to utilize deep learning applications that have recently brought about a big paradigm shift. However, in mobile and automotive environments where actual deep-running applications are applied, hardware resources and power-supply are limited. Also, for edge embedded devices such as Jetson, the resource contention caused by the integrated structure has an adverse effect on overall performance. We introduce RMED to adaptively adjust the resource usage of the workloads used to mitigate these problems. RMED is a variant of EdgeIso that applies isolation technology to mitigate existing resource contention and optimizes power consumption while maintaining performance for target workloads. RMED's approach achieves more than 20% energy savings in multitasking situations while successfully maintaining performance isolation of target workloads.

Keywords: Edge Computing, Power Consumption, Resource Management, Performance Isolation

Student Number: 2019-25497

Contents

Abstract **i**

Chapter 1 Introduction **1**

Chapter 2 Background and Motivation **5**

2.1 Edge Computing 5

2.2 Power Management Scheme 7

2.3 Resource Contention 9

Chapter 3 Design and Implementation **11**

3.1 Overview of EdgeIso 11

3.2 RMED: Resource Management for Edge Devices 12

3.2.1 Overall Architecture 12

3.2.2 Algorithm 14

3.3 Implementation 17

Chapter 4 Evaluation **18**

4.1 Experimental Setup 18

4.2 Performance Results 20

4.2.1 Deep-Learning Workloads 20

4.2.2 Latency-Critical Workloads 22
4.3 Overheads 24

Chapter 5 Related Work **26**

Chapter 6 Conclusion **28**

Bibliography **29**

초록 **32**

List of Figures

Figure 1.1 Brief Structure for Edge Computing 2

Figure 2.1 Power consumption of SSD Inference in Jetson TX2. . . 9

Figure 2.2 Power consumption of SSD Inference (NVPModel mode changing). 10

Figure 3.1 RMED Overall Architecture. 13

Figure 4.1 RMED with Single Shot MultiBox Detector 20

Figure 4.2 RMED with Single Shot MultiBox Detector + PointPillars (Power consumption aspect) 21

Figure 4.3 RMED with Single Shot MultiBox Detector + PointPillars (Resource contention aspect) 21

Figure 4.4 Power consumption of Tailbench benchmarks 23

Figure 4.5 The normalized latency of Tailbench benchmarks with Single Shot MultiBox Detector (SSD) and PointPillars (PP). 24

Figure 4.6 Slowdown Ratio Comparison of RMED with Single Shot MultiBox Detector + PointPillars 25

List of Tables

Table 2.1 Differences between Cloud Computing and Edge Computing 6
Table 2.2 NVPModel Mode Definition 8

Chapter 1

Introduction

Edge Computing have been widely used in various fields such as Mobile Phone, Wearable Device and Autonomus Driving[9, 10]. In particular, edge computing is used a lot in terms of low latency, network bandwidth reduction, and security by conducting inferences near users, especially in the application of deep learning workloads that perform real-time inferences using models learned by big data. Figure 1.1 shows the brief architecture of Edge Computing. Edge computing represents a lot of advantages by using edge devices that are close to users, compared to the way in which a centralized cloud server is used.

Deep learning applications are carried out through the inference process using models created through learning. At this time, the learning process, which processes the vast amount of input data used as learning data, is commonly performed on server computers with sufficient computing power. On the other hand, edge devices with actual deep learning applications, which deduce models created through learning, often have limited hardware resources and power to supply compared to servers, failing to show the desired performance. Most

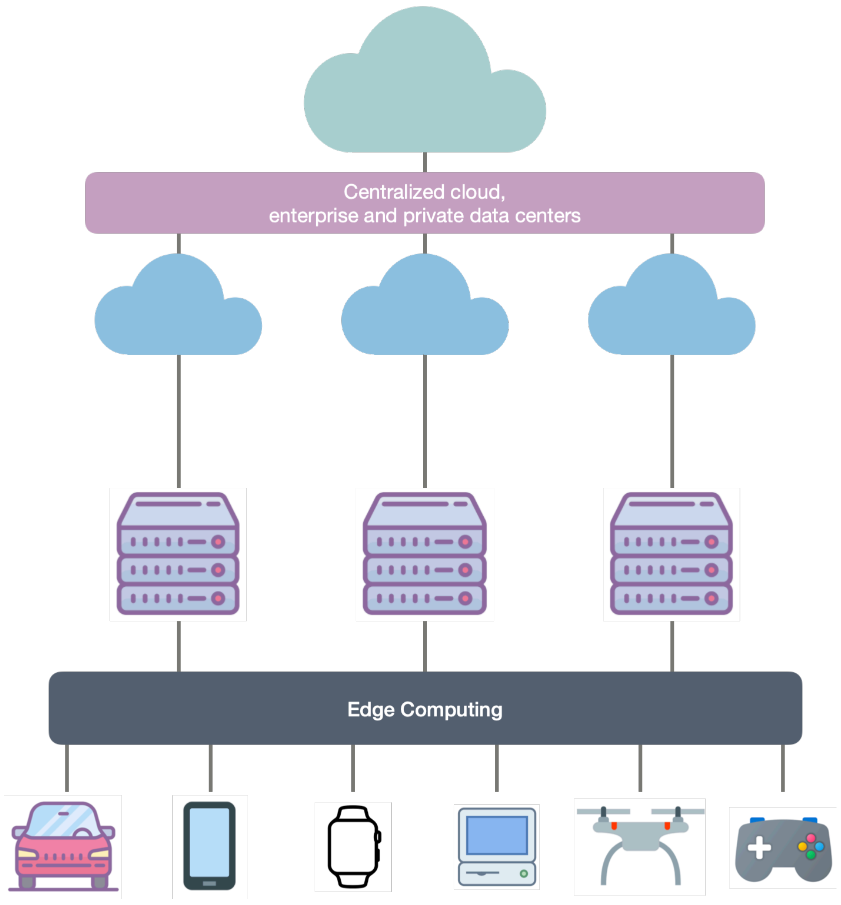

Figure 1.1: Brief Structure for Edge Computing

of the embedded devices used in edge computing recently have heterogeneous architecture, combine cores with different computing capabilities on a single die. This class of systems include CPU-GPU integrated systems, where the CPU and GPU are fused together on a die, sharing a common memory-bandwidth. Fused CPU-GPU platforms are presently found on mobile and embedded devices such as Jetson series from NVIDIA[4]. These systems are designed to share a common memory system to reduce transmission overhead between devices, but at the same time create resource contention that degrades overall performance. Also, high-speed communication technologies such as 5G, which have been in the spotlight recently, provide fast latency, but at the same time, they consume a lot of power. For these reasons, it is necessary to optimize resource management while maintaining the performance of target deep learning workloads on edge devices.

Dynamic Voltage and Frequency Scaling (DVFS) is a well-known method to improve the energy efficiency of computing systems. The voltage and frequency levels are typically adjusted through the use of system-level software known as frequency governors. The mechanism used by these governors to control the frequency knobs tend not to be application aware, which leads to energy inefficiencies. In addition, this problem is aggravated by the fact that, on modern SOCs such as the NVIDIA Jetson TX2 (which is used for this study), the number of possible frequency combinations can be very large, rendering a brute-force search for the energy-optimal point infeasible. In order to evaluate the energy efficiency of the Jetson TX2's default DVFS system, we find a diverse set of collaborative applications, and collect performance and power consumption data.

There has been a lot of significant studies to reduce power consumption in embedded or edge environments. Some studies have suggested reducing the power consumption in multitasking environments, but there are restricted in server-centric computing and need additional hardwares[8, 21, 20, 22].

In this paper, we propose an RMED model that adds power management function to existing EdgeIso[11] scheduler designed to overcome performance degradation caused by resource contention during multitasking in Edge environment. At the profiling stage, RMED measures the amount of power required to solo-run the target workload, and at the scheduling stage manages the minimum power required to maintain the performance of the target workload in a multitasking environment.

In experiments, we evaluate the three workloads, (SSD, PointPillars, Tailbench) on object detection and latency-critical workloads. Experiments are conducted on NVIDIA Jetson TX2 with Dual Denver 2 + Quad ARM A57 and 256-core Pascal GPU.

The contributions of our work are as follows:

- We present an optimized power consumption technique for multitasking in edge devices.
- We propose an online model that measures resource contention and power consumption by profiling the status of processes.
- We evaluate the energy savings and performance benefits of RMED and find that, on an average, we achieve about 20% reduction over the baseline power manager.

The composition of this paper is as follows. Chapter 2 describes the background and motivation. Chapter 3 describes the EdgeIso which is underlying this study and describes the architecture and algorithm of RMED. Chapter 4 describes the results of the experiment. Chapter 5 describes the relevant existing research. Chapter 6 concludes the paper.

Chapter 2

Background and Motivation

2.1 Edge Computing

Edge computing is a type of distributed computing that distributes computing tasks to devices close to users or uses the tasks by dividing them into servers and edges[12]. Table 2.1 shows differences between Edge Computing and Server-centric Cloud Computing. Server-centric computing, the opposite of edge computing, has the advantage of being able to use vast amounts of cloud server resources without restrictions. Large companies such as Amazon[1], Microsoft[2], and Google[3], which can provide these resources, are developing and providing services related to cloud computing and making users pay for the resources they use. However, this server-centric approach to cloud computing has limitations: First, the non-deterministic latency that can occur in the communication process between the server and the terminal makes it difficult to achieve the same results when the same initial conditions are given. Second, if the connection between the server and the terminal becomes unstable, stability

	Cloud Computing	Edge Computing
Adventages	-Large-scale data analytics and processing capabilities -High-level computing can be provided to various terminals	- Provides fast service response speed - Relieve the overload of communication infrastructure - Reduce damage from cyber attacks - Service stability against network failure
Disadvantages	- Service response speed decreased due to increase in number of terminals and network instability - Risk of communication infrastructure overload - Increased security risks from cyber attacks, such as data leakage	- Difficulty in analyzing and processing large-scale data

Table 2.1: Differences between Cloud Computing and Edge Computing

will be drastically reduced, causing problems in areas where user stability is paramount, such as self-driving cars. Third, there are threats to data security and the risk of contamination at the stage of transferring data collected from user terminals.

Unlike common cloud computing, edge computing enables data locality and data-driven networking and reduces application latency. Because of these factors, edge computing is leveraged for a variety of applications that require real-time interaction, massive data storage and movement, improved security and privacy, and multiple access networking. Also, by analyzing and utilizing data directly without going through the server at the edge where the user's data is collected as terminal data, the user can respond more quickly compared to waiting for the analyzed results in the cloud data center. Recently, the performance of terminal edge devices has been able to operate workloads without difficulty compared to the past, and in conjunction with these advantages, edge computing has been actively studied and used in mobile phone, smart-watches, and self-driving cars.

Edge hardware environments differ from the general development environments in terms of computational capacities, operating systems used, network connectivity, and power specifications. In software domain, system libraries, available functions as well as the encoding can be different. These ambiguities require extra effort and attention during edge application development. The embedded device we used in our research, the Jetson TX2, is an embedded AI computing device, containing two Denver core CPUs and one NVIDIA Pascal GPU. In such an embedded device, in order to perform an inference task in real time, several sensors are attached and used. At this point, the additional power consumption of sensors is not negligible, so optimizing the power consumed within the edge device is an important research task.

2.2 Power Management Scheme

Dynamic Voltage and Frequency Scaling (DVFS)[23] is a power management framework that allows a processors to change frequency and voltage status. DVFS can help achieve power savings when the workload does not require the highest performance (the highest frequency or voltage) of the processor. The frequency and voltage pairs are related because a corresponding increase in voltage is required to increase the frequency of the system. This is why the CPU core has the voltage required by the specific frequency, which is stored in the Frequency table generated by the Operating Performance Point (OPP) List. In general, CPUs have two major clock domains, Core Clock and Memory Clock. Core Clock controls the frequency of the core, and Memory Clock controls the frequency of the main memory. Efficient DVFS schemes scale both of these frequencies to ensure power efficient execution of applications. The core clock is independent for each core, allowing a multicore system to run different cores at different frequencies.

Jetson devices provide NVPModel[6] that allows user to easily change CPU core on/off and frequency control and GPU frequency control to preset values.

Mode	Mode Name	Denver Core Number & Frequency	ARM Core Number & Frequency	GPU Frequency
0	Max-N (Maximum Performance)	2, 2.0GHz	4, 2.0GHz	1.3GHz
1	Max-Q	0	4, 1.2GHz	0.85GHz
2	Max-P Core-All	2, 1.4GHz	4, 1.4GHz	1.12GHz
3	Max-P ARM (ARM Core Only)	0	4, 2.0GHz	1.12GHz
4	Max-P Denver (Denver Core Only)	2 2.0GHz	0	1.12GHz

Table 2.2: NVPModel Mode Definition

There are a total of 5 basic presets, each having a difference in whether the Denver core is activated, whether the ARM core is activated, CPU frequency, and GPU frequency (Table2.2). Users can create and use user-defined modes instead of the presets provided, at which point they should create valid frequencies and possible voltage pairs to avoid functional failures.

We first measured the amount of power consumed during the execution of the workload to analyze the performance differences due to allocation of resources. We used Tegrastats[19] to know the energy consumption status of the current workload. Tegrastats is NVIDIA-provided resource profiling technology that allows user to read the resources of the corresponding workload every specific cycle. Figure 2.1 shows the amount of power used to perform the Single Shot Multi-box Detector's inference workload for each resource. The types of voltages identified by tegrastats are Input, CPU, GPU, SystemOnChip, Wifi chip, and DDR Memory. We change the mode of NVPModel provided by NVIDIA, that is, gradually increase the frequency, and try to see the performance change of the target workload accordingly. As shown in figure 2.2, the modes that use Denver Core basically consume more power than using the general ARM Core. In addition, the use of Denver Core does not have a significant impact on overall workload performance, so it has been confirmed that not using Denver Core (if it does not require the computing power of the Denver Core) is beneficial to power management in typical situations. Disabling Denver Core on the Jetson

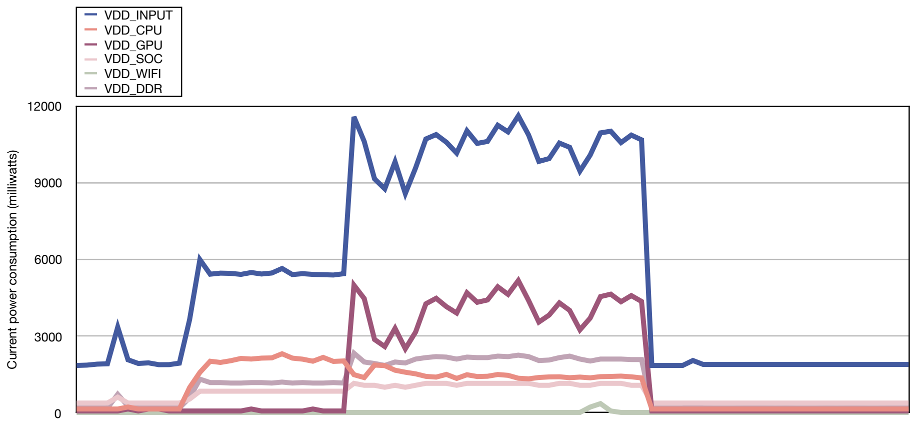

Figure 2.1: Power consumption of SSD Inference in Jetson TX2.

board as the default setting is the same reason. Also, figure 2.1 shows that there is a phase where each power consumption rapidly increases according to the execution time of the workload. This is to minimize power consumption in the idle state due to DVFS, which is basically on, and maximize power according to the required resource of the used workload. For the above reasons, the advantage in power consumption was confirmed when the frequency of the core allocated to the target workload was variably managed in a multitasking environment. We tried to make a more efficient resource management framework by reflecting the characteristics of the workload compared to the power consumption optimization due to DVFS application in Jetson.

2.3 Resource Contention

Resource contention occurs when the same resource is shared by multiple tasks. It occurs due to conflict over task access and oversubscription to a resources within multi-tenant machines which can be exuberated within different sce-

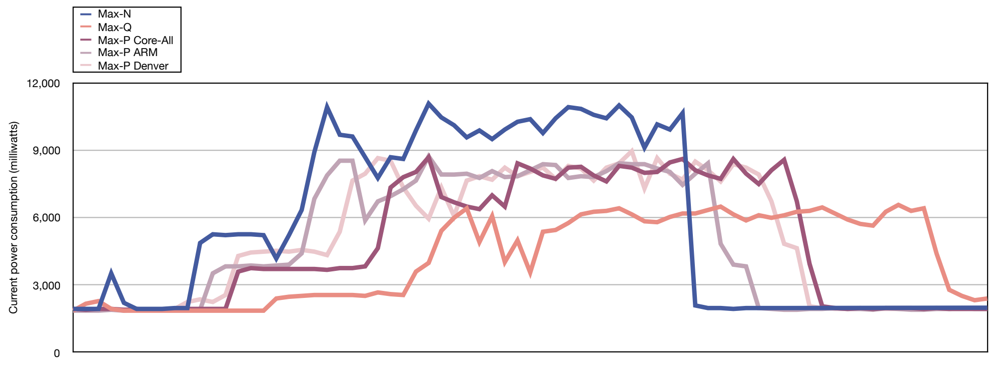

Figure 2.2: Power consumption of SSD Inference (NVPModel mode changing).

narios including: hardware heterogeneity, ineffective algorithm logic, additional task clones requiring more resources and resource usage being higher than accepted threshold value. Among them, hardware heterogeneity is the main reason of resource contention, which occurs due to a mismatch between hardware specification and specified application constraints (e.g. budget, deadline, etc.) leading to task performance degradation. Especially for edge embedded devices such as Jetson, resource contention due to the integration structure significantly reduces overall performance in multitasking situations. Also, the resource contention has a negative effect on power consumption in multitasking situations[13]. Therefore, reducing resource contention through scheduling can have a positive effect on power management as well.

Chapter 3

Design and Implementation

In this chapter, we propose a resource management framework available in edge devices. We first briefly introduce EdgeIso, an edge scheduling framework underlying this study. Second, we explain how the algorithm of resource management applied to the our framework. This approach is written in user-level python on the Linux operating system.

3.1 Overview of EdgeIso

EdgeIso [11] profiles the resource contentions between tasks and isolates resources for maintain performance for target workload. The purpose of EdgeIso is to address the performance degradation caused by insufficient allocation of resources for latency-critical tasks in multitasking situations. EdgeIso consists of two main parts: profiler and scheduler. The profiler measures the resource usage of the target task over a short period of time, comparing the resource usage for each non-contention situation and multi-tasking situation. The comparison values are used to identify contention for CPU, LLC, and memory bandwidth, and to select the most important contention factors. It also observes changes in the phase of the target workload for gradual resource allocation, which tracks changes in the number of threads due to resource usage. The scheduler applies the appropriate isolation technique according to the type of Dominant Resource Contention (DRC) tracked by the Profiler. At this time, isolation techniques are applied using pre-defined policies according to the type of DRC, which are each core allocation, cycle throttling, and GPU frequency throttling. After the scheduling step is over, the profiling step repeats the process of measuring the degree of resource contention.

Dominant resource contention has a negative impact on the performance of the target workload while also generating additional power consumption due to resource redistribution time. Therefore, identifying and preventing DRCs can be considered desirable for power saving. The resource management technique proposed in this paper is a method of adding power optimization based on the basic construction of EdgeIso. Considering the existing research aims to ensure the performance of the target workload by reducing resource contention, this research aims to optimize the power consumption that is important factor in edge devices while minimizing performance loss while bringing the advantages of EdgeIso.

3.2 RMED: Resource Management for Edge Devices

3.2.1 Overall Architecture

Figure 3.1 shows overall process of RMED briefly. Based on EdgeIso's resource profiling and scheduling, additional techniques for power optimization have been added. The power consumption optimization step begins at the profiling stage,

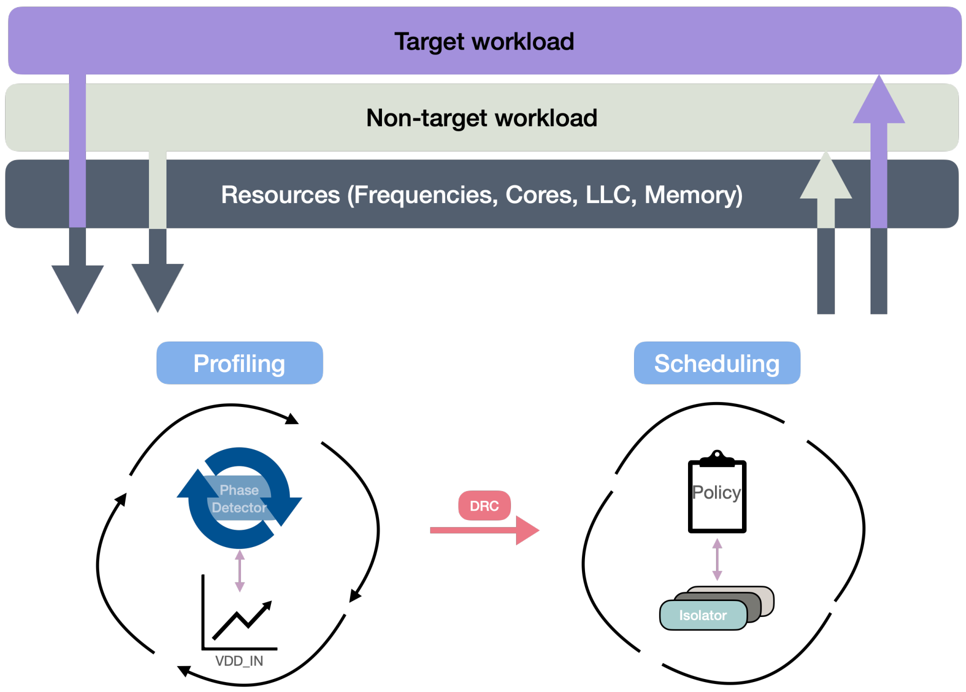

Figure 3.1: RMED Overall Architecture.

which identifies the resource usage of the workload. Phase-detector has two functions: first, it compares the resources used by solo-run of the target workload with the resources used in multitasking to hand over the DRC to the scheduler, and monitors the VDD values read as tegra-stat to track changes from CPU-centric to GPU-centric tasks of the target workload. When looking at the trend of power consumed in the process of inference deep learning-based workloads (See figure 2.1), it can be seen that the utilization of CPU increases in the layer loading stage, and then passes to GPU jobs. Considering this phenomenon, the CPU frequency was overclocked in the layer model loading stage to speed up the CPU job, and downclocked when the GPU job was transferred to reduce the power consumed by the CPU.

Since most embedded devices used in research fields such as self-driving or mobile devices are co-run situations, they need to utilize the characteristics of workloads identified during solo runs while also comparing them with resources used for co-runs. For example, in the case of SSD (Single Shot Multi-box Detector), which is a kind of object detection framework based on deep learning, the number of layers of the model used, the size of the model, and the size of the input data affect the resources consumed during the solo run[24]. These factors apply not only to machine learning-based workloads, but also to the majority of general-purpose workloads used by edge devices[25].

The purpose of the resource management framework is to provide reasonable energy savings. Determining an appropriate frequency for the target workload can depend on runtime information that is not available during static analysis, so frequencies based on pre-written policies can generate erroneous predictions for some applications and adversely affect performance. Therefore, the solution must be solved by gradually adjusting the frequency while analyzing the resource in real time. Therefore, RMED alone is not sufficient for energy efficient execution of CPU-GPU collaborative applications. This limitation of RMED is solved by introducing a technology that gradually applies the isolator according to the DRC applied by the existing EdgeIso. This technology uses run-time feedback from performance counters and phase detection and other real-time utilization metrics to reduce power consumed by resource contention, allowing an effective RMED to function as a more effective framework.

3.2.2 Algorithm

Algorithm 1 shows the pseudo-code of RMED, which is combine of EdgeIso framework and Power management technique. In a multitasking situation, profiling is performed in the background, and selects the main resources that affect resource contention (line 2). The DRC selection uses specific threshold compared to the idle resource usage data for each resource type, which can be observed to determine whether a contention has occurred (line 4). The resource contention is mitigated by gradually applying the isolator according to the selected DRC (line 5). After application of the isolator, resource contention is continuously observed to determine whether to apply additional isolation, and switch to idle state if there is no competition (line 6-12).

While applying isolation technology to alleviate resource contention, power optimization techniques are simultaneously performed. In order to detect switching phases from CPU Jobs to GPU Jobs, the target workload's VDD_CPU value is tracked to recognize phase where energy consumption is soaring (line 13). After the transition of the target workload to the GPU Job is confirmed, a predefined policy is used to underclock the frequency of the CPU core assigned to the target workload to reduce overall power consumption (line 14-21). If a DRC is detected due to a bottleneck light on the CPU that can occur when the frequency is lowered, it can be returned to the previously set frequency (line 22-23).

Going back to the profiling step, the result after applying the scheduling is observed, feedback is received, and the entire process is repeated.

Algorithm 1: Pseudo-code of the RMED Algorithm

```
1  while True do
2      resource_contention = resource_contention_detect()
3      if resource_contention then
4          target_resource = compare_threshold()
5          isolator = set_appropriate_isolator(target_resource)
6          switch target_resource do
7              case strengthen
8                  isolator.set_strengthen()
9              case weaken
10                 isolator.set_weaken()
11             case idle
12                 isolator.set_idle()
13     phase_transition = phase_detection()
14     if phase_transition then
15         switch current_phase do
16             case CPU_centric
17                 frequency.overclock()
18                 stable_status = status_check()
19             case GPU_centric
20                 frequency.underclock()
21                 stable_status = status_check()
22         if !stable_status then
23             frequency.idle()
```

3.3 Implementation

With the method suggested above, in order to ensure the performance of the target workload and optimize power consumption during multitasking, the code was added to the Profiler and Scheduler parts of the existing EdgeIso platform. The whole code was implemented based on Python, and the system call function was additionally implemented in online profiler to parse and get the power consumption of the currently running process based on tegra-stat, and to adaptively control CPU core frequency in Scheduler. Since the code is written at the user level, there may be overhead in terms of performance, but there is an advantage that it can be used universally in linux-based embedded devices using ARM cores such as Jetson series.

Chapter 4

Evaluation

4.1 Experimental Setup

We evaluate our strategy on NVIDIA Jetson TX2 with the considering the Edge Computing environment. NVIDIA Jetson TX2 has a Dual Denver 2 + Quad ARM A57 and a 256-core Pascal GPU. The Jetson TX2 can use a total of 8GB of memory and the bandwidth of the memory is 50GB/s. We set up the experimental environment on linux ubuntu 16.04, and installed basic Jetson related packages such as JetPack. The workloads selected for the experiment were source-built on an additional external SSD, Samsung 860 Pro 512GB was used.

We evaluate different combination of workloads considering the general edge environment where multitasking occurs. As for the workload combination, an experiment was performed on the assumption of a situation in the recognition part that simultaneously detects and classifies objects after receiving images through various sensors in an autonomous vehicle, which is being actively studied recently. Assuming the process of recognizing an image through a camera, we used Single Shot MultiBox Detector (SSD) [14] for 2D object detection task. We used the pre-trained VGG models for SSD inference job, and in order to put the load on the CPU and GPU as much as possible, preparations were made to select the image that goes into the input data.

Also, assuming the process of recognizing an object through LIDAR, we used PointPillars [15], the point clouds bounding box detection for 3D objects. PointPillars uses a pillars representation based on SECOND[16] voxelisation with infinite height. Compared to 2D object detection, 3D object detection is more difficult because higher position accuracy of the 3D boundary box is required in the space. It also causes more resource consumption in terms of the size of input data used in training and reasoning. The KITTI benchmark dataset [5] was employed to evaluate our proposed method. It contains 7481 training and 7518 testing point clouds, including three categories: car, cyclist, and pedestrian. The training dataset was divided into a training set (3712) and a validation set (3769), since the ground truth of the testing dataset is not publicly available. The proposed framework was trained for 200k iterations using the Adam optimizer. The initial learning rate was 0.002, the exponential decay rate was 0.8, and there was a decay every 18,750 iterations.

We compared our method with using the basic DVFS manager as a baseline, and obtained results in terms of power consumed in performing the workload and the mitigation of resource contention in multitasking situations.

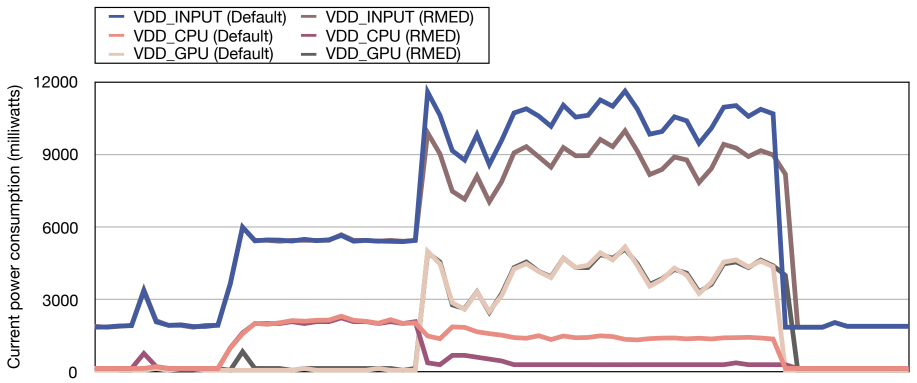

Figure 4.1: RMED with Single Shot MultiBox Detector

4.2 Performance Results

4.2.1 Deep-Learning Workloads

Figure 4.1 shows the power consumed per second by the Single Shot Multi-box Detection inference task. Compared to the basic DVFS is applied, the amount of power consumed by the CPU in the GPU-centric phase is reduced by RMED. As the voltage applied to the CPU decreases, the overall input power is reduced, resulting in about 22% power reduction compared to the baseline. In addition, since the CPU frequency is reduced while maintaining the GPU frequency in the GPU-centric phase, it shows that the overall execution time is not significantly affected.

Figure 4.2 shows the performance in a multitasking situation using both SSD and PointPillars inference. In a multitasking situation, as in the case of performing a single workload, the total power consumed is reduced by adjusting the frequency of the CPU core in the GPU-centric part. Specifically, It shows that the total input power consumption in GPU-centric phase is reduced by about 20% compared to the baseline.

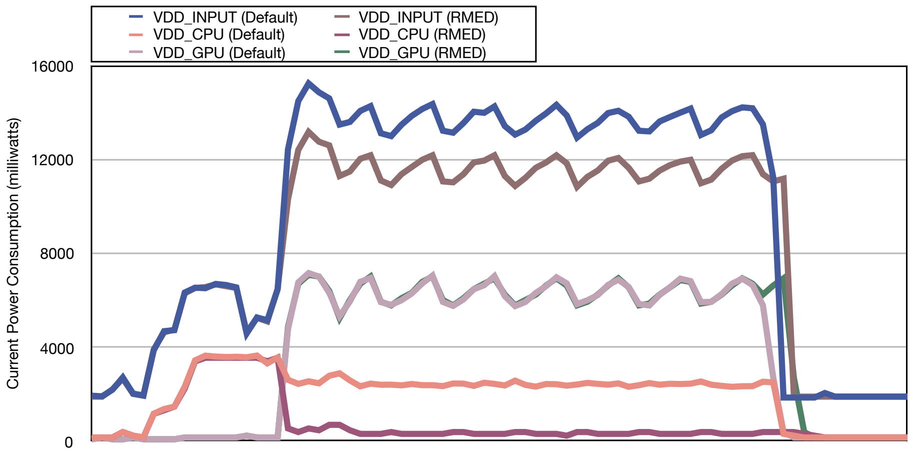

Figure 4.2: RMED with Single Shot MultiBox Detector + PointPillars (Power consumption aspect)

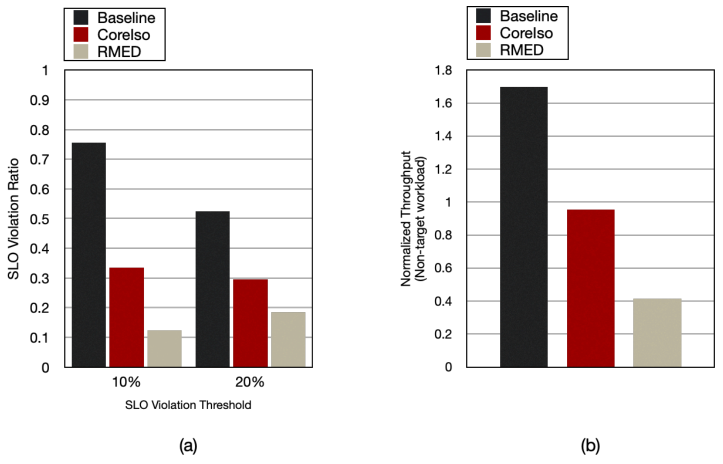

Figure 4.3: RMED with Single Shot MultiBox Detector + PointPillars (Resource contention aspect)

In addition, to see how much RMED mitigates resource contention in a multitasking situation, we determine and compare the Service Level Objetives (SLOs) relative to a single performance of each workload. Figure 4.3 (a) shows the SLO violation ratio when threshold is set to 10%, and 20%, respectively. Baseline is a basic state where isolation technology is not applied, and CoreIso means when a core is directly assigned to a workload using cgroup, a linux kernel feature that limits and isolates the use of resources by processes. RMED violates SLOs less than 20% regardless of the threshold. When the SLO threshold is 10%, the ratio of SLO violation can become around 10%. Figure 4.3 (b) shows the normalized throughput of background task. We conducted the experiment using instruction rate (IPS; instruction per second) to measure the throughput of non-target workload. When RMED is applied, it shows the lowest throughput for non-target workload because RMED limits the execution of non-target workload to reduce memory contention. However, this non-target workload performance degradation can be acceptable in a prioritized multitasking edge environment without affecting the performance of the target workload.

These results show that RMED is effectively distributing resources in resource contention and minimizing performance degradation for concurrent workloads.

4.2.2 Latency-Critical Workloads

Considering the edge environment where latency-critical work is focused, we experimented with a multitasking environment that simultaneously executes the deep learning inference task and the Tailbench[17] benchmark.

Tailbench aggregates a set of interactive benchmarks from web servers and databases, to speech recognition and machine translation systems, and proposes a new methodology to analyze their performance. We used img-dnn, sphinx, and

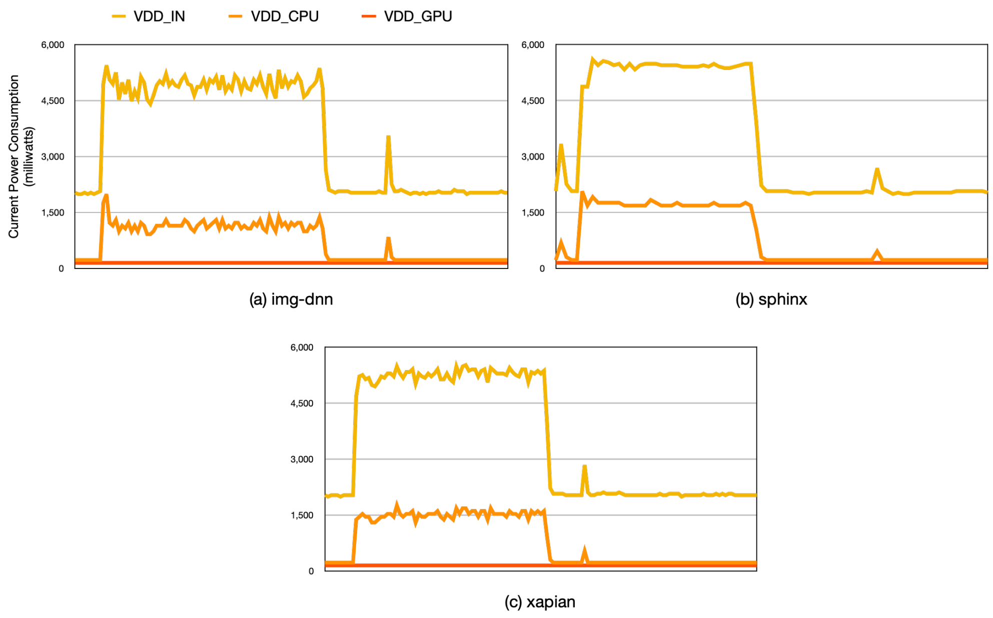

Figure 4.4: Power consumption of Tailbench benchmarks

xapian among several benchmarks included in the tailbench, and performed each workload and deep learning inference job at the same time. Since each workload of the tailbench only needs CPU computation, we experimented on how effectively reducing resource contention without lowering the frequency of the CPU core (figure 4.4). Figure 4.5 shows the normalized latency of each Tailbench benchmark. Because img-dnn is a memory-intensive task that consumes a lot of memory bandwidth, it causes performance degradation of target workloads in multi-tasking situation. As shown in figure 4.5(a), RMED achieves much lower latency than other schemes by mitigating memory contention. In the case of sphinx(b), RMED shows a latency that is not significantly different from the alternative methods, but generally shows a reduction in resource contention. In the case of xapian(c), latency in multi-tasking situations is significantly different compared to the other two workloads, because xapian requires

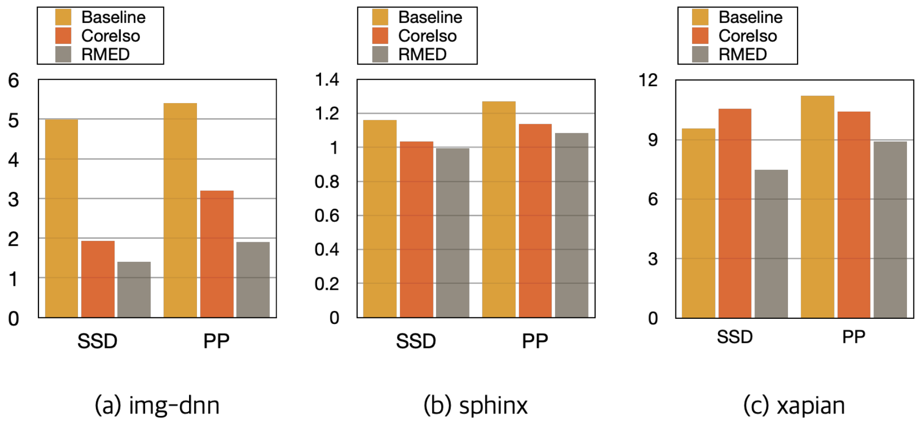

Figure 4.5: The normalized latency of Tailbench benchmarks with Single Shot MultiBox Detector (SSD) and PointPillars (PP).

a tighter resource occupancy, which is a benchmark that is heavily affected by hardware performance.

4.3 Overheads

Since RMED brings advantages in power consumption by lowering CPU core frequency in the GPU-centric phase of the deep learning workload, some overheads exist in terms of performance such as execution time. In addition, in multitasking situations, isolation schemes that reduce resource contention have an overhead in processing data across the profile and scheduler, but this does not have a significant impact on overall resource utilization. Figure 4.6 shows the slowdown ratio from various combinations. The ratio of overhead varies depending on the nature of the workload being used (e.g. memory-intensive, compute-intensive, latency-intensive, etc). Compared to the basic conditions of solorun and corun, we can see that there is a slow down of an average of less than

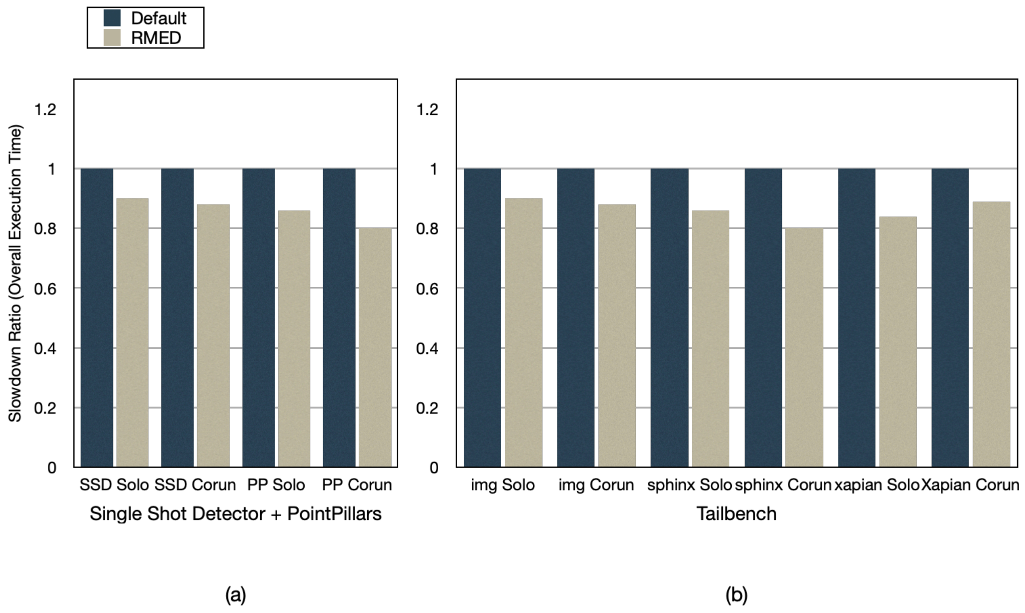

Figure 4.6: Slowdown Ratio Comparison of RMED with Single Shot MultiBox Detector + PointPillars and Tailbench Benchmarks

6%. Given the overall benefits of RMED, this overhead does not significantly affect overall performance.

Chapter 5

Related Work

While several recent works have been done on optimizing latency and energy simultaneously in multicore systems, not much research targets Edge Computing environments and explores their unique characteristics in performance optimization.

Hardware-based power management consists largely of DVFS or other techniques for power capping. Zhang and Hoffmann[7] present a online hybrid coordination technique that sets hardware power capping using RAPL and then iteratively uses a binary search mechanism to find proper processor performance state, core count, and socket count in each dimension.

You et al. [8] researched the resource allocation problem for multiuser MEC system, and minimized the mobile devices' energy consumption with the computation latency constraint. They jointly considers the offloading and the communication resource allocation in a TDMA model. However, the work above neglects the competition for bandwidth (impacts the overall application performance) among the users at the wireless access points or base station.

Qasaimeh et al. [21] conducted a study of energy consumption when using deep learning frameworks for object detection in joint design platforms such as CPU, GPU, and FPGA. According to their experiment results, energy consumption compared to performance is the most effective when using FPGA, but this method is difficult to apply to other general platforms.

Mishra and Nikita[20] combines machine learning and control theories to schedule CPU resources on heterogeneous multicores. For a given application, their scheme uses control-theoretic methods to dynamically adjust the resource allocation, and machine learning to estimate the application's latency and power for a given resource allocation plan.

Zhang et al. [22] designed an energy-efficient computation offloading scheme with the purpose of minimizing the energy consumption. However, due to the nature characteristic of offloading, there is a disadvantage in latency rather than running the workload on the edge device itself.

Chapter 6

Conclusion

Resource management in integrated-architecture devices has become a major concern in edge computing research. To solve this problem, the scientific community and the industry have proposed several approaches to reduce the resource consumption and contention of computing systems, but are limited to HPC or Cloud server based. In this paper, we present RMED, a resource management scheduler for edge devices. RMED mitigates the resource contention on the edges and savings power consumption in multitasking situations. The experimental results show that when a variety of deep learning-based workloads and latency-critical tasks are combined, RMED save about 20% of the total input power and effectively perform resource isolation.

Bibliography

[1] Amazon Web Services, https://aws.amazon.com/ko/

[2] Microsoft Azure, https://azure.microsoft.com/ko-kr/

[3] Google Cloud Platform, https://cloud.google.com/

[4] NVIDIA Jetson TX2 Developer Kit Module, https://www.nvidia.com/ko-kr/autonomous-machines/embedded-systems/jetson-tx2/

[5] Kitti 3D Object Detection Evaluation 2017 data, http://www.cvlibs.net/datasets/kitti/eval_3dobject.php

[6] NVPModel: NVIDIA Jetson TX2, https://www.jetsonhacks.com/2017/03/25/nvpmodel-nvidia-jetson-tx2-development-kit

[7] Zhang, Huazhe, and Henry Hoffmann. "Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques." ACM SIGPLAN Notices 51.4 (2016): 545-559.

[8] You, Changsheng, et al. "Energy-efficient resource allocation for mobile-edge computation offloading." IEEE Transactions on Wireless Communications 16.3 (2016): 1397-1411.

[9] Mao, Yuyi, et al. ”A survey on mobile edge computing: The communication perspective.” IEEE Communications Surveys & Tutorials 19.4 (2017): 2322-2358.

[10] Ai, Yuan, Mugen Peng, and Kecheng Zhang. ”Edge computing technologies for Internet of Things: a primer.” Digital Communications and Networks 4.2 (2018): 77-86.

[11] Y. Nam, Y. Choi, B. Yoo, H. Eom and Y. Son, ”EdgeIso: Effective Performance Isolation for Edge Devices,” 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), New Orleans, LA, USA, 2020, pp. 295-305, doi: 10.1109/IPDPS47924.2020.00039.

[12] C. You and K. Huang, ”Multiuser Resource Allocation for Mobile-Edge Computation Offloading,” 2016 IEEE Global Communications Conference (GLOBECOM), Washington, DC, 2016, pp. 1-6, doi: 10.1109/GLOCOM.2016.7842016.

[13] Yu S, Yang H, Wang R, Luan Z, Qian D (2017) Evaluating architecture impact on system energy efficiency. PLOS ONE 12(11): e0188428. https://doi.org/10.1371/journal.pone.0188428

[14] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. E. Reed. SSD: single shot multibox detector. CoRR, abs/1512.02325, 2015. 5, 6

[15] Lang, Alex H., et al. ”Pointpillars: Fast encoders for object detection from point clouds.” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.

[16] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. Sensors, 18(10):3337, 2018.

[17] H. Kasture and D. Sanchez, "Tailbench: a benchmark suite and evaluation methodology for latency-critical applications," 2016 IEEE International Symposium on Workload Characterization (IISWC), Providence, RI, 2016, pp. 1-10, doi: 10.1109/IISWC.2016.7581261.

[18] SAMSUNG 960 PRO SSD Specification, https://www.samsung.com/semiconductor/minisite/ssd/product/

[19] Tegrastats Utility, https://docs.nvidia.com/jetson/page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/AppendixTegraStats.html

[20] Mishra, Nikita, et al. "CALOREE: Learning control for predictable latency and low energy." ACM SIGPLAN Notices 53.2 (2018): 184-198.

[21] Qasaimeh, Murad, et al. "Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels." 2019 IEEE International Conference on Embedded Software and Systems (ICESS). IEEE, 2019.

[22] Zhang, Ke, et al. "Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks." IEEE access 4 (2016): 5896-5907.

[23] Kim, Wonyoung, et al. "System level analysis of fast, per-core DVFS using on-chip switching regulators." 2008 IEEE 14th International Symposium on High Performance Computer Architecture. IEEE, 2008.

[24] Zhao, Zhong-Qiu, et al. "Object detection with deep learning: A review." IEEE transactions on neural networks and learning systems 30.11 (2019): 3212-3232.

[25] Fan, Qiang, and Nirwan Ansari. "Application aware workload allocation for edge computing-based IoT." IEEE Internet of Things Journal 5.3 (2018): 2146-2153.

초록

엣지 컴퓨팅은 현재 서버 중심 컴퓨팅에 비해 낮은 대기 시간, 효율적인 데이터 처리로 인한 대역폭 사용 감소, 보안 향상 등의 이점으로 적극 활용되고 있다. 이는 최근 큰 패러다임 전환을 불러온 딥러닝 애플리케이션을 적극 활용하면서 더욱 중요한 연구가 됐다. 그러나 실제 딥러닝 애플리케이션이 적용되는 모바일 및 자동차 환경에서는 하드웨어 자원과 전력 공급에 한계가 있다. 또한 Jetson과 같은 엣지 임베디드 기기의 경우, 통합 구조에 의해 야기되는 자원 경합은 전체 성능에 부정적인 영향을 미친다. 이러한 문제를 해결하기 위해 사용되는 워크로드의 리소스 사용량을 적응적으로 조정하는 RMED를 도입한다. RMED는 격리 기술을 적용하여 리소스 경합을 완화하고 타깃 워크로드에 대한 성능을 유지하면서 전력 소비를 최적화하는 EdgeIso의 변형이다. RMED의 접근 방식은 멀티 태스킹 상황에서 20% 이상의 에너지 절감을 달성하는 동시에 대상 워크로드의 성능 격리를 성공적으로 달성한다.

주요어: Edge Computing, Power Consumption, Resource Management, Performance Isolation, Autonomous Driving

학번: 2019-25497