creative commons

COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학석사학위논문

Performance Guarantee of ADAS on Integrated ECU Concurrently Hosting Multi-Domain Applications

다중 도메인 애플리케이션을 동시에 호스팅하는 통합 ECU에서 ADAS의 성능 보장

2023년 08월

서울대학교 대학원
컴퓨터공학부
유 서 환

Performance Guarantee of ADAS on Integrated ECU Concurrently Hosting Multi-Domain Applications

다중 도메인 애플리케이션을 동시에 호스팅하는 통합 ECU에서 ADAS의 성능 보장

지도교수 이 창 건

이 논문을 공학석사 학위논문으로 제출함

2023년 05월

서울대학교 대학원

컴퓨터공학부

유 서 환

유서환의 공학석사 학위논문을 인준함

2023년 06월

위 원 장	김 지 홍	(인)
부위원장	이 창 건	(인)
위 원	김 태 현	(인)

Abstract

Performance Guarantee of ADAS on Integrated ECU Concurrently Hosting Multi-Domain Applications

Seo Hwan Yoo
Department of Computer Science and Engineering
The Graduate School
Seoul National University

Since ECUs are difficult to maintain and require high hardware costs such as wiring, previously distributed ECUs are being integrated. As a result of ECU integration, multi-domain applications share hardware resources with ADAS, leading to mutual interference. In this study, we conducted research to guarantee the performance of ADAS on integrated ECU hosting multi-domain applications. At first, we quantitatively observed that performance of ADAS is affected by contention for shared resources. This led us to acknowledge the necessity of isolating ADAS from the resources used by other multi-domain applications to mitigate the impact of shared resources contention. So, we built a resource isolation environment using LXC so that the minimum required resources by ADAS can be isolated from those used by other multi-domain applications. Before deriving the minimum resource requirement for ADAS, we preliminary optimized ADAS to reduce the minimum resource requirement. Afterwards, we experimentally derived the minimum resource requirement for guaranteeing

ADAS performance with the proposed algorithm. Finally, we confirmed that the performance of ADAS is guaranteed with the derived minimum resources in the environment where multi-domain applications are concurrently executed.

keywords : Autonomous Driving, Performance Guarantee, Minimum Resources
Student Number : 2021-25483

Contents

1 Introduction 1

2 Motivation 4

2.1 Performance Degradation of ADAS Due to Shared Resources . . 4

2.2 Optimizing ADAS for Reduced Resource Requirement 6

3 Optimizing and Minimum Resource Allocation for ADAS 8

3.1 Building LXC-based Environment for Concurrent Execution of Multi-Domain Applications 8

3.2 ADAS Optimization via Event-Driven 9

3.3 Minimum Resource Requirement to Guarantee ADAS Performance 10

4 Experiment Results 12

4.1 ADAS Optimization 12

4.2 Performance Guarantee of ADAS 12

5 Conclusion 17

References 18

List of Figures

1 Zigzag scenario 5

2 Core mapping for shared resources contention experiment 5

3 The proportion of alignment delay in default ADAS 7

4 (a) Periodic-polling ADAS (b) Event-driven ADAS 10

5 Deriving minimum CPU requirement of ADAS 13

6 Core mapping for CPU isolation 14

7 Deriving minimum memory BW requirement of ADAS 15

8 Core mapping for CPU and memory BW isolation 16

List of Tables

1 ADAS performance affected by shared resource contention . . . 6

2 ADAS performance improvement by event-driven method 13

3 Performance isolation with minimum CPU 14

4 Performance isolation with minimum CPU and memory BW . . 16

1 Introduction

Recently, as advanced features such as autonomous driving and in-vehicle infotainment(IVI) are added to vehicle, there is an increasing need for additional Electronic Control Units (ECUs) to support added functionalities. However, since maintaining multiple ECUs is challenging, and the hardware costs, such as wiring, are expensive, there is a trend towards controlling vehicles using a single integrated chipset with excellent performance, rather than increasing the number of ECUs for accommodating new feature[1]. Examples of this trend include Nvidia Drive's Thor and Samsung's Exynos Auto V9.

As a result of the integration of ECUs, multi-domain applications such as powertrain and infotainment now share the hardware resources with Advanced Driver Assistance System (ADAS). Through experimentation, we observed that the performance of ADAS is influenced by the contention for shared resources, specifically CPU and memory bandwidth(BW). By using synthetic workloads that utilize CPU and memory BW as best effort, we quantitatively examined the impact of shared resources on ADAS performance. It was observed that the average E2E(End to End) response time of ADAS increases by 66% due to CPU contention and by 19% due to memory BW contention. In both cases, the vehicle collided with a stopped vehicle due to contention. Considering the potential threat to passengers' lives caused by system failures in ADAS, guaranteeing the performance of ADAS on integrated ECU is not a choice but a necessity.

To guarantee the performance of ADAS on integrated ECU, it is necessary to mitigate the impact of contention for shared resources. For this purpose,

the minimal resources required to guarantee ADAS performance, separate from the resources used by other multi-domain applications, should be allocated to ADAS in isolation. So, we built LXC-based resource isolation environment to separate the essential resources needed for ADAS from those utilized by other multi-domain applications.

Before deriving the resource requirement for ADAS, we investigated the potential for optimization within Autoware[2], an open-source autonomous driving software, which is used by ADAS, to reduce the minimum required resources. Through analysis of Autoware, we observed that the cumulative delay caused by external factors such as scheduling mechanisms accounted for around 66% of the E2E response time on average. To optimize the minimal resource requirement for ADAS, we conducted preliminary optimization by removing delay caused by external factors. There have been previous studies such as [3] that improve ADAS performance by finding the optimal spin rate of each node without changing the Autoware. However, [3] has the disadvantage of finding the spin rate of each node through exhaustive search. Subsequently, using the optimized ADAS, the minimum resource requirement were experimentally derived using the algorithm presented in this paper.

With the derived minimum resources, we confirmed that the ADAS performance is guaranteed in the environment where multi-domain applications are concurrently executed through the experiment results that both the E2E response time and collision ratio decrease to levels similar to those achieved when ADAS is executed alone.

Contribution **:** In our research, we conducted study to guarantee the perfor-

mance of ADAS on integrated ECU hosting multi-domain applications simultaneously. First, we performed quantitative observations indicating that the performance of ADAS is affected by shared resource contention. Second, we built the resource isolation environment using LXC to separate the minimum necessary resources for ADAS from those utilized by other multi-domain applications. Third, before deriving the minimal resource requirement for ADAS, we optimized ADAS to improve its performance to minimize the resources required by ADAS. With the optimized ADAS, we experimentally derived the minimum resource requirement that guarantees ADAS performance. Lastly, we experimentally confirmed that the ADAS performance is guaranteed when multi-domain applications are executed concurrently on integrated ECU.

The rest of the paper is organized as follows. Section 2 explains the motivation behind this research. In Section 3, it explains the ADAS optimization and the minimum resource derivation method for guaranteeing ADAS performance. Section 4 presents the experimental results of our research. Finally, Section 5 concludes the paper and discuss future research plan.

2 Motivation

2.1 Performance Degradation of ADAS Due to Shared Resources

We conducted experiment to see if ADAS has performance degradation issues due to contention for shared resources. Through Autoware and SVL [4], an autonomous driving simulator, we evaluated ADAS performance using zigzag scenario(Figure 1) in which stopped vehicles are zigzag-evaded at a maximum speed of 7.5 m/s. Each experiment was performed 100 times, and the experiment types and core mappings are illustrated in Figure 2. As performance metric, we measured the E2E response time from the start of receiving LiDAR sensing topics from SVL to the completion of vehicle control in ADAS node. We also recorded the occurrence of collision with stationary vehicles. Detailed information regarding the experiments is provided below.

- ADAS : Performance evaluation without resource interference
- ADAS + SCW : Performance evaluation when CPU contention occurs by running Synthetic CPU Workload (SCW), which performs arithmetic calculations with best effort on the same core as ADAS
- ADAS + SMW : Performance evaluation when memory BW contention occurs by running Synthetic Memory Workload (SMW), which performs memory access with best effort, in the cluster different from the one in which ADAS is running

As expected, when synthetic workloads were executed concurrently, an increase in ADAS's E2E response time and collision ratio was observed. The ex-

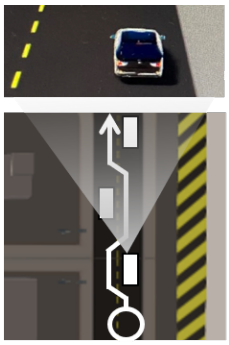


Figure 1: Zigzag scenario

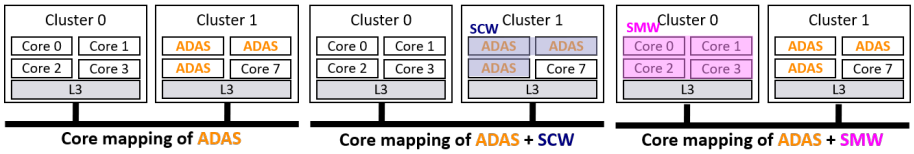


Figure 2: Core mapping for shared resources contention experiment

perimental results are presented in Table 1. It was found that CPU contention led to 66% increase in ADAS's average E2E response time, while memory BW contention resulted in 19% increase. In addition, when ADAS was performed alone without resource contention, the vehicle moved according to the scenario without collision, but in the case of resource contention, we observed that the vehicle collided with the stopped vehicle due to the vehicle direction calculated based on the old sensing data as the E2E response time increased. This put emphasis on the need to mitigate the impact of contention for shared resources to guarantee the performance of ADAS. Consequently, it implies that the minimum resources required to guarantee ADAS performance should be isolated

from the resources used by other multi-domain applications. Therefore, it is necessary to derive the minimum resource requirement that guarantee the performance of ADAS.

Table 1: ADAS performance affected by shared resource contention

Workload	E2E Response Time	Collision Ratio
ADAS (No Cont.)	180.62ms	0%
ADAS + SCW (CPU Cont.)	300.15ms	100%
ADAS + SMW (memory BW Cont.)	215.46ms	5%

2.2 Optimizing ADAS for Reduced Resource Requirement

Before deriving the minimum resource requirement, we examined the potential for optimization within Autoware, to reduce the resources allocated to ADAS in isolation. In Autoware, the E2E process consists of chain of internal nodes, starting from the start of receiving LiDAR sensing topics from SVL to vehicle control. The E2E response time is calculated as the sum of response times of nodes within the chain, along with the cumulative delay(hereinafter alignment delay) caused by external factors such as scheduling mechanisms. In order to assess the impact of alignment delay on the E2E response time, we examined the alignment delay excluding the response times of nodes within the chain, as shown in Figure 3. As a result, it was observed that alignment delay accounted for 66% of the E2E response time on average. By optimizing and eliminating unnecessary alignment delay, the minimal resource requirement for ADAS can be reduced, which means that other multi-domain applications

can use more resources. That is, in order to reduce the minimum resource requirement of ADAS, ADAS optimization is required.

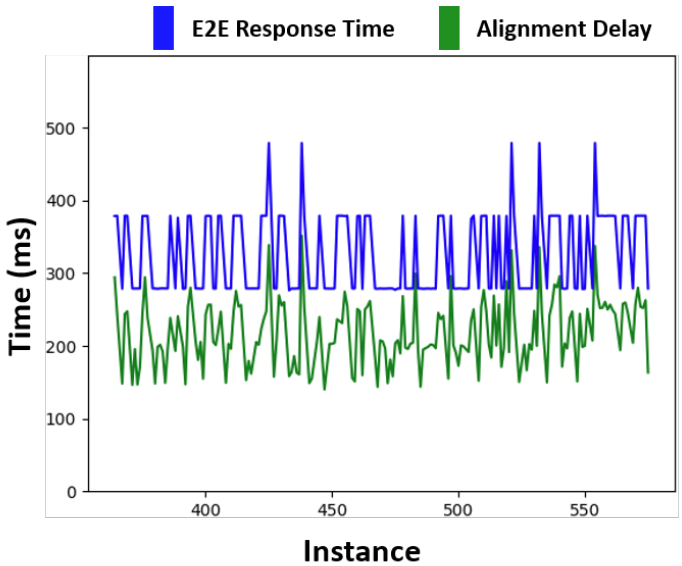


Figure 3: The proportion of alignment delay in default ADAS

3 Optimizing and Minimum Resource Allocation for ADAS

To guarantee the performance of ADAS, we conducted research on isolating the minimal required resources for ADAS from the resources used by other multi-domain applications. Firstly, we built the resource isolation environment. Before deriving the minimum resource requirement, we optimized ADAS to reduce the minimum resource requirement. Then, we conducted research on deriving the method for determining the minimal resource requirement for ADAS.

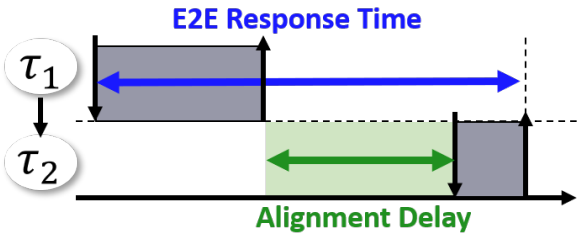
3.1 Building LXC-based Environment for Concurrent Execution of Multi-Domain Applications

In order to guarantee performance of ADAS on integrated ECU, it becomes essential to reduce the negative effects caused by contention for shared resources. To achieve this objective, it is crucial to allocate the minimal resources required to guarantee ADAS performance, separate from the resources utilized by other multi-domain applications. So we needed the environment that would isolate the resources allocated to ADAS from the resources of multi-domain applications. Since the hypervisor code loaded in Exynos Auto V9 was black-box according to corporate policy, it was difficult to troubleshoot problems related to resource isolation. To achieve efficient resource management, the previously deployed black-box hypervisor was replaced with the open-source LXC which provides OS-level virtualization that allows multiple isolated Linux systems to run on top of a single Linux kernel.

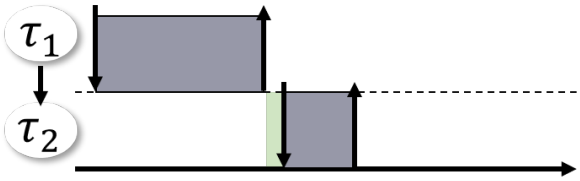
3.2 ADAS Optimization via Event-Driven

Through section 2, it was confirmed that 66% of the average E2E response time of the existing ADAS is attributed to alignment delay. To improve the performance of ADAS by reducing alignment delay, we conducted research on ADAS optimization. Autoware is a software based on Robot Operating System(ROS). Nodes based on ROS generally sleep so that tasks run every predefined period through rate.sleep(), and repeatedly perform periodic-polling-based data processing in the spinOnce() function to handle currently received data. In other words, each node in the chain processes data only when it wakes up from sleep, not immediately when the data is generated. This means that if there is misalignment between nodes, as shown in Figure 4 (a), it can cause the alignment delay, severely delaying the response time of tasks and increasing the E2E response time.

To eliminate the previously identified alignment delay issue, we modified Autoware to event-driven-based system where each node processes data at the moment it is generated using the spin() function. The spin() function operates without sleep, waking up the node when data to be received is generated and performing the necessary processing on the received data. By using spin () instead of spinOnce() combined with sleep(), we resolved the problem of alignment delay occurrence. Consequently, we reconfigured nodes to operate based on the spin() function, transforming the original periodic-polling-based Autoware into event-driven system. In other words, as shown in Figure 4 (b), we reduced alignment delay, leading to significant reduction in the E2E response time and improvement in the performance of ADAS.



(a)



(b)

Figure 4: (a) Periodic-polling ADAS (b) Event-driven ADAS

3.3 Minimum Resource Requirement to Guarantee ADAS Performance

Through section 2, it was confirmed that ADAS was affected by contention for shared resources, specifically CPU and memory BW, resulting in driving failures. This observation highlights the need to derive the minimum resource requirement that guarantee the performance of ADAS to ensure its functionality.

To determine the minimum CPU requirement for ADAS, the ADAS experiment introduced in section 2 was used. Starting with the allocation of all CPU to ADAS, the number of CPU was gradually reduced while moni-

toring the occurrence of vehicle collision in the zigzag scenario or until only one CPU was available. In other words, the smallest number of CPU in the ADAS experiment where no collision occurred indicates the minimum CPU requirement for guaranteeing ADAS performance. In terms of allocating CPU to ADAS, we used CPUSET.

To determine the minimum memory BW requirement for ADAS, we used ADAS experiment using the derived minimum number of CPU. Starting with the allocation of the memory BW budget of all CPU involved in ADAS as the memory BW specification, binary search approach was used to gradually reduce the memory BW budget allocated to the CPU until vehicle collision occurred in the zigzag scenario. To prevent the binary search from going on indefinitely, a condition is added to stop the binary search when a search range less than predefined threshold is reached. In other words, the smallest memory BW in the ADAS experiment where no collision occurred indicates the minimum memory BW requirement for guaranteeing ADAS performance. To limit the CPU's memory BW, memguard[5], a technique based on controlling CPU stalls through LLC miss budget, was used.

4 Experiment Results

In this section, we share experimental results on ADAS optimization and the algorithm for finding the minimum resources required by ADAS using Autoware and SVL. The experiments were conducted using evaluation board consisting of two 4-core CPU clusters, Exynos Auto V9. Evaluation board used lpddr4x and the maximum memory BW specification was 68.2GB/s. To reduce the overhead of SVL, the autonomous driving simulation was configured to run on an external PC instead of the evaluation board.

4.1 ADAS Optimization

We conducted the ADAS experiment introduced in section 2 comparing the conventional periodic-polling-based ADAS with the ADAS modified to event-driven approach. Table 2 shows the results. Compared to the previous experiment where alignment delay was present, the average alignment delay decreased by 71.9%, resulting in 47.0% reduction in average E2E response time while keeping the sum of the response times of the nodes similar. Furthermore, the collision ratio was improved from 36% to 0%, confirming that scenario that required more than 3 CPUs in the periodic-polling-based ADAS are sufficient with 3 CPUs in the event-driven-based ADAS. This means, by optimizing the ADAS, the number of CPU resources required by the ADAS has decreased.

4.2 Performance Guarantee of ADAS

We experimentally determined the minimum resource requirement to guarantee the performance of ADAS by investigating the impact of changes in

Table 2: ADAS performance improvement by event-driven method

Workload	E2E Response Time	Alignment Delay	Collision Ratio
Periodic-Polling ADAS	340.63ms	224.01ms	36%
Event-Driven ADAS	180.62ms	63.00ms	0%

resources on ADAS performance. To experimentally confirm the minimum CPU requirement for ADAS, we used the ADAS experiment and the proposed algorithm. Figure 5 shows the results. We can observe that collision occur when the number of CPU is less than 3, indicating that the minimum CPU resource requirement for ADAS is 3.

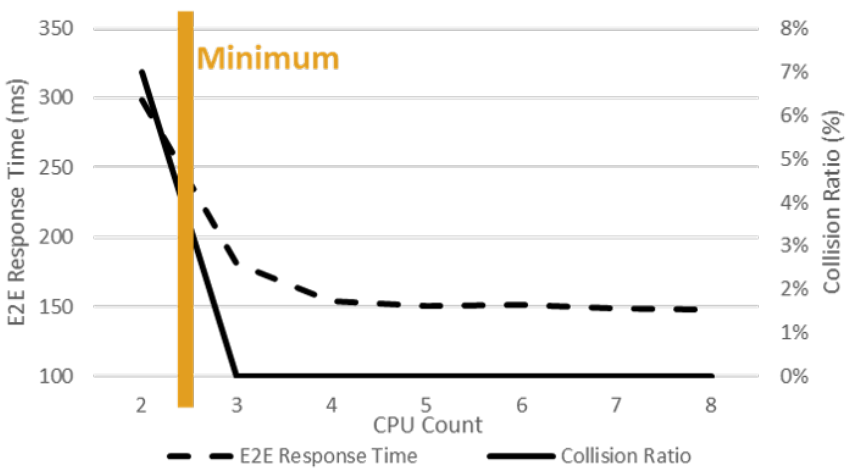


Figure 5: Deriving minimum CPU requirement of ADAS

We conducted experiments to investigate whether allocating the minimum CPU resources to ADAS while isolating them from the CPU used by multi-domain applications would lead to performance degradation. We used the ADAS experiment introduced in section 2 , and the experimental setup and

core mapping are shown in Figure 6. The specific details of the experiments are as follows.

- ADAS : Performance evaluation using the minimum number of CPU without resource interference
- ADAS + SCW + ISO. : Performance evaluation using the minimum number of CPU when running Synthetic CPU Workload (SCW), which performs arithmetic calculations with best effort on different cores from ADAS, and CPU contention was minimized through isolation

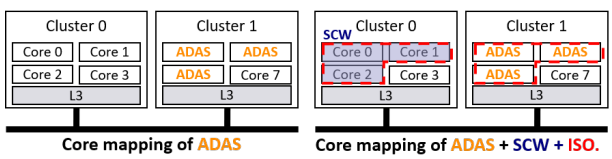


Figure 6: Core mapping for CPU isolation

Through Table 3, it can be observed that the synthetic cpu workload has minimal impact on ADAS performance. When CPU isolation is applied, it is evident that both the E2E response time and collision ratio decrease to levels similar to those achieved when ADAS is executed alone.

Table 3: Performance isolation with minimum CPU

Workload	E2E Response Time	Collision Ratio
ADAS (No Cont.)	180.62ms	0%
ADAS + SCW + ISO(CPU Iso.)	180.88ms	0%

To experimentally determine the minimum memory BW requirement for ADAS, we used the ADAS experiment using the derived minimum number of CPU, along with the algorithm proposed in section 3 (with a threshold of 1GB). The results are presented in Figure 7. The binary search terminated when the search interval became smaller than the predefined threshold, and it can be observed that the smallest memory BW budget that avoids collision is 1.598 GB/s. Therefore, we can confirm that the minimum memory BW resource requirement for ADAS is 1.598 GB/s.

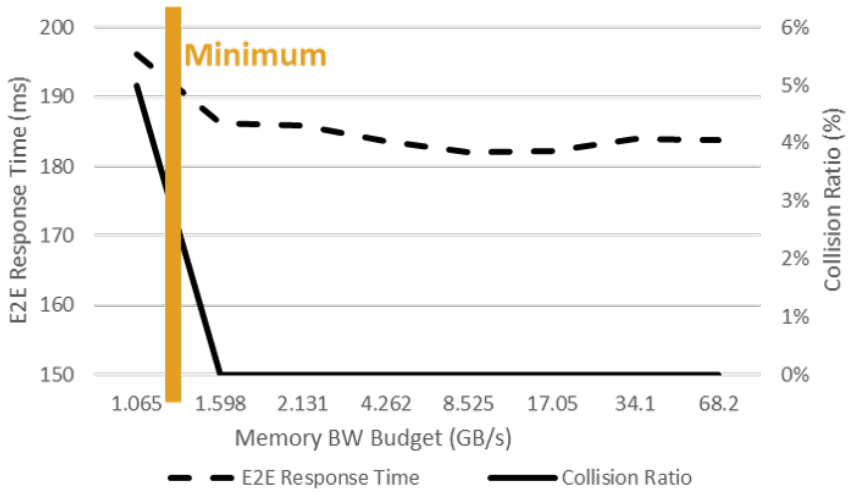


Figure 7: Deriving minimum memory BW requirement of ADAS

We performed experiments to determine whether there is a degradation in ADAS performance when the minimum CPU and memory BW are isolated and allocated to ADAS, separate from the CPU and memory BW used by multi-domain applications. The ADAS experiment was used, and the experiment types and core mappings are shown in Figure 8. The specific details of the

experiments are as follows.

- ADAS : Performance evaluation using the minimum number of CPU and memory BW without resource interference
- ADAS + SMW + ISO. : Performance evaluation using the minimum number of CPU and memory BW when Synthetic Memory Workload (SMW), which performs memory access with best effort, was executed in the cluster different from the ADAS cluster, and memory BW contention was minimized through isolation

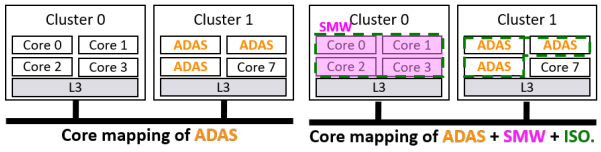


Figure 8: Core mapping for CPU and memory BW isolation

By referring to Table 4, it can be observed that synthetic memory workload has minimal impact on ADAS performance. When CPU and memory BW isolation is implemented, it is evident that the E2E response time and collision ratio decrease to levels similar to that achieved when ADAS operates alone.

Table 4: Performance isolation with minimum CPU and memory BW

Workload	E2E Response Time	Collision Ratio
ADAS (No Cont.)	186.24ms	0%
ADAS + SMW + ISO(memory BW Iso.)	187.27ms	0%

5 Conclusion

We conducted the study to guarantee the performance of ADAS on integrated ECU hosting multi-domain applications. First of all, we quantitatively observed the performance degradation of ADAS due to resource contention. Then, we built LXC-based resource isolation environment. Prior to deriving the minimum resource requirement for ADAS, we optimized its performance. With the optimized ADAS, we empirically derived the minimum resource requirement that guarantees the performance of ADAS. Lastly, we confirmed that performance guarantee of ADAS was attained using the derived minimum resources when running multi-domain applications concurrently. In future work, we plan to study on flexible isolation to maximize performance and completely eliminate mutual interference.

References

[1] Marco Di Natale and Alberto Luigi Sangiovanni-Vincentelli. Moving from federated to integrated architectures in automotive: The role of standards, methods and tools. Proceedings of the IEEE, 98(4):603–620, 2010.

[2] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems. In 2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS), pages 287–296, 2018.

[3] Byungkyu Park and Chang-Gun Lee. System optimization of ros-based open source autonomous driving platform on embedded board environment. In Proceedings of the 6th International Conference on Algorithms, Computing and Systems, pages 1–7, 2022.

[4] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, Eugene Agafonov, Tae Hyung Kim, Eric Sterner, Keunhae Ushiroda, Michael Reyes, Dmitry Zelenkovsky, and Seonman Kim. Lgsvl simulator: A high fidelity simulator for autonomous driving. In 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), pages 1–6, 2020.

[5] Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. Memguard: Memory bandwidth reservation system for efficient per-

formance isolation in multi-core platforms. In 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), pages 55–64, 2013.

요약(국문초록)

ECU는 유지 보수하기가 어렵고 wiring과 같은 하드웨어 비용이 들기 때문에, 기존 분산되어 있던 ECU들이 통합되고 있다. ECU들이 통합됨에 따라, 다중 도메인 응용들이 ADAS와 하드웨어 자원을 공유하여 서로에게 영향을 주게 되었다. 우리는 다중 도메인 응용을 동시에 호스팅하는 통합 ECU에서 ADAS의 성능을 보장하는 연구를 진행하였다. 먼저, 우리는 ADAS의 성능이 공유 자원에 대한 경합에 의해 영향을 받는 것을 정량적으로 관찰했습니다. 이를 통해, ADAS 성능 보장을 위해서 ADAS가 요구하는 최소한의 필요 자원이 다른 다중 도메인 응용이 사용하는 자원들과 격리되어 ADAS에 할당되어야 하는 필요성을 확인하였다. 그렇기 때문에, ADAS에 필요한 최소한의 자원을 다른 다중 도메인 응용에서 사용하는 자원과 격리할 수 있도록 LXC를 사용하여 자원 격리 환경을 구축했습니다. ADAS의 최소 자원 요구 사항을 도출하기 전, 최소 자원 요구 사항을 줄이기 위하여 ADAS 최적화를 진행하였다. 이 후, ADAS 성능을 보장하는 최소 자원 요구 사항을 본 논문에서 제시하는 알고리즘을 이용하여 실험적으로 도출하였다. 마지막으로, 다중 도메인 응용이 동시에 실행되는 환경에서 도출된 최소 자원으로 ADAS의 성능이 보장됨을 실험을 통해 확인하였다.

주요어 : 자율주행, 성능보장, 최소자원
학 번 : 2021-25483