Heterogeneity-Aware Scheduling on SoCs for Autonomous Vehicles

Aporva Amarnath, Subhankar Pal, Hiwot Tadese Kassa, Augusto Vega, Alper Buyuktosunoglu, Hubertus Franke, John-David Wellman, Ronald Dreslinski and Pradip Bose

Abstract—Fully autonomous vehicles (AVs) must meet stringent real-time performance and safety-criticality constraints of multiple applications simultaneously in highly dynamic environmental conditions. To enable such a system, carefully selected accelerators and general-purpose cores on Systems-on-Chips (SoCs) are required. However, schedulers that are agnostic to this heterogeneity not only lead to inefficient utilization of hardware resources in the SoC, but can also increase the time to complete the AV's mission.
Our paper addresses this challenge by proposing a heterogeneity-aware, multi-level scheduler called HetSched. HetSched leverages run-time information about the underlying heterogeneous SoC, along with the applications' real-time constraints to improve an AV's mission time. Our evaluation demonstrates a reduction in mission time of 1.7× for a highly congested scenario, when compared against state-of-the-art (SOTA) schedulers. Furthermore, when used as part of an SoC design space exploration loop, in comparison to the SOTA schedulers, HetSched reduces the number of accelerators required by an SoC to safely complete the AV's mission by 1.9× on average.

Index Terms—Heterogeneous SoC, autonomous vehicles, real-time scheduler, design space exploration.

1 Introduction

Fully autonomous vehicles that can perceive the environment and respond faster than human drivers can greatly improve road safety. Current vehicles with self-driving modes are characterized only at an autonomy level of 2 [1], *i.e.* they require regular human monitoring and intervention while the vehicle is performing autonomous operations. To enable level 4 and 5 fully autonomous systems that require no human intervention, we need to resort to hardware-software co-designed platforms that can meet stringent real-time and safety requirements. Such a system should be able to process multiple simultaneous tasks with a high volume of data under varying environmental conditions, in order to safely complete an AV's mission.

Heterogeneous systems-on-chips (SoCs) [2], [3] equipped with multiple acceleration engines specifically cater to ultra-efficient execution of AV application kernels. Although hardware acceleration is key towards the goal of fully autonomous systems, it increases the *heterogeneity* in the system, *i.e. high variability in task execution time*. For example, in our experiments, an accelerator can execute tasks up to 300× faster than the slowest processing element (PE) in the SoC. This calls for new scheduling approaches that can exploit such heterogeneity in the hardware.

However, prior real-time schedulers, designed for heterogeneous systems, are built to cater only to real-time constraints of individual applications, without leveraging the heterogeneity in the SoC to guide scheduling decisions for multiple applications. This leads to missed deadlines for the running applications, and higher overall mission time for the AV, that may need to be compensated with an over-provisioned SoC design.

In order to address this mismatch, we propose **HetSched**, a multi-level scheduler that leverages the heterogeneous characteristics of the underlying hardware platform and the applications' runtime characteristics and constraints to satisfy the growing throughput demand of AVs. Specifically, our contributions are:

- We propose *fine-grained heterogeneity-aware schemes*, namely, **heterogeneous and hybrid ranking**, that dynamically prioritize tasks based on the task execution times on different PE types in the SoC, and the real-time constraints of the tasks.
- We propose a **task pruning** optimization that helps *prune non-safety-critical tasks* that cannot meet their deadlines in favor of safety-critical tasks that are required to meet all real-time constraints for the safe completion of the AV's mission.
- We demonstrate significant reductions in mission time, energy consumption and SoC area, obtained with HetSched in a design space exploration (DSE) loop.

We show that HetSched on average reduces the accelerator resource requirement of an efficient SoC to safely complete AV missions by 1.9×, while improving mission time by 1.7× in comparison to prior state-of-the-art schedulers [4], [5].

A. Amarnath, S. Pal, H. Kassa and R. Dreslinski are with University of Michigan, Ann Arbor. A. Vega, A. Buyuktosunoglu, H. Franke, J.-D. Wellman and P. Bose are with IBM Research, Yorktown Heights.

Corresponding author: Aporva Amarnath (aporvaa@umich.edu)

2 Background and Motivation

2.1 Autonomous Vehicle Applications

AV applications are composed of compute intensive tasks like perception, planning, control and communication [2], [3]. These tasks are commonly deployed on hardened logic in the form of accelerators. *E.g.* Lin *et al.* [2] propose the use of accelerators for perception tasks along with their CPU and GPU implementations to meet real-time requirements of an AV.

Such AV applications can typically be represented as directed-acyclic graphs (DAGs) that are statically known, but the arrival and execution of which are dynamic, and only determined during the AV's operation. Each DAG is associated with a *real-time deadline*, determined by the speed of the AV. Furthermore, each DAG is also assigned a *safety-criticality constraint*, based on the risk involved to life and property from the AV. ISO 26262 identifies four such levels, denoted by ASIL A, B, C, and D for AV automotive functions [6]. ASIL A represents operations which will not result in any injuries, while D represents operations that can result in the highest degree of automotive hazard. Following this scheme, we classify the DAGs as follows:
Critical DAGs: DAGs that belong to ASIL B, C and D and can lead to hazards are assigned *Crit=2*. *E.g.* perception of a stop sign during an AV's mission. We pessimistically consider that missing a deadline for *any* critical DAG leads to mission failure.
Non-critical DAGs: All other DAGs belonging to ASIL A are assigned *Crit=1*. *E.g.* object recognition on a blind-spot camera while traveling straight on a single-lane road, which if not executed before its deadline will not lead to any hazards.

We also consider three congestion scenarios, namely *rural*, *semi-urban* and *urban*, based on the frequency of DAGs with *Crit=2* encountered during the mission. *E.g.* an AV encounters more crosswalks while driving in an urban area as opposed to a

TABLE 1
Comparison of HetSched with Prior Real-Time Schedulers

Prior Art	Input	Ranking Type, Parameters	Safety-Aware	Hetero-Aware	Task-Pruning
2lvl-EDF [4]	Multiple Static DAGs	Dynamic Earliest deadline first	×	×	×
ADS [5]	Multiple Static DAGs	Dynamic Earliest finish time first	✓	×	×
HetSched	Multiple Static DAGs	Dynamic Deadline, PE variation	✓	✓	✓

rural area. Finally, we consider metrics like the AV's *mission time*, and *energy consumed* by the SoC to evaluate our AV systems. These metrics are influenced by the maximum speed achievable by an AV operating in diverse environmental conditions, while meeting deadlines for all *Crit*=2 DAGs.

2.2 Related Work on Real-Time Schedulers

Autonomous vehicles must execute applications within stringent real-time constraints. Prior work propose the use of heterogeneous SoCs [2], [3] and schedulers built for heterogeneous systems [7], [8] to help meet performance constraints of *individual* applications. However, to enable level 4 and 5 AVs, we require schedulers that can schedule for mixed criticalities and multiple DAGs simultaneously [9]. In this regard, we describe prior art in this section and compare against HetSched in Table 1.

2lvl-EDF: Wu and Ryu [4] present the best speed fit EDF scheduler that prioritizes tasks according to the earliest deadline and assigns tasks to the PE tasks can complete the fastest on.

ADS: Xie *et al.* [5] propose an adaptive dynamic scheduler (ADS), to optimize the execution time of a DAG while considering safety and criticality of the system. ADS ranks tasks based on earliest finish time policies [7] and dynamically prioritizes tasks with higher criticality levels (safety-aware).

While, ADS is safety-aware, 2lvl-EDF is not. Moreover, both schemes neither exploit the variation in the timing profile of tasks on the heterogeneous SoC (not hetero-aware) nor predict when to prune non-critical tasks. HetSched caters to stringent AV requirements, *i.e.* to meet real-time deadlines for all critical DAGs, to reduce overall mission time, and to improve hardware efficiency by employing schemes that are safety-aware, hetero-aware and allow for task pruning of non-critical tasks.

3 HetSched: Heterogeneity-Aware Scheduling

We propose HetSched as a multi-level scheduler that exploits the heterogeneous nature of SoCs and the applications' constraints to improve mission time and hardware efficiency for AVs. Specifically, HetSched consists of two levels: `Meta-Sched` and `Task-Sched`, as depicted in Figure 1 (left). `Meta-Sched` is responsible for DAG pre-processing, whereas `Task-Sched` performs the actual task-to-PE assignment and hardware telemetry. The two levels communicate with each other using a *ready queue* and a *completed queue*. HetSched also uses profiling information of tasks and data movement costs across *all* possible PEs in the SoC to aid in the scheduling decision. We describe the operations of `Meta-Sched` and `Task-Sched` in this section.

Dependency Tracking: `Meta-Sched` resolves dependencies, denoted as an edge between two nodes in a DAG, to determine *ready* tasks. A task is determined as *ready* when all its parent nodes in the DAG have completed execution.

Task Prioritization: Once a task is deemed ready, a rank is assigned to it (*higher rank implies higher priority*). For this, we first derive the real-time and safety constraints of a task from that of the DAG it belongs to. The *Crit* of a task is assigned as the *Crit* of the DAG. Due to the presence of multiple paths and tasks in a DAG, the sub-deadline (SD) of a task is determined by the DAG's deadline ($Deadline_{DAG}$) and a weighted ratio of the task's execution time relative to its path's execution time. We call this ratio the task's sub-deadline ratio (SDR). We consider

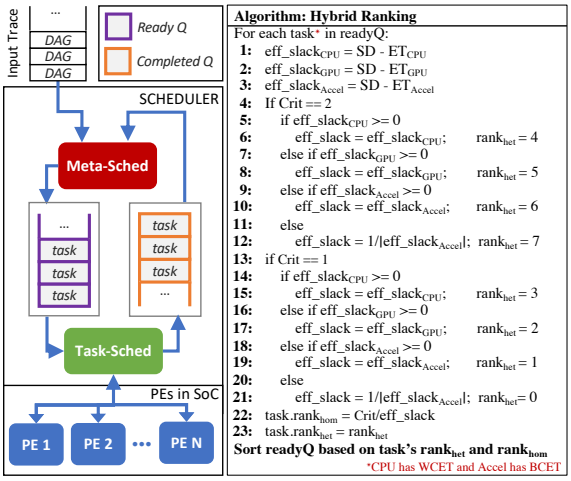

Fig. 1. *Left:* HetSched showing `Meta-Sched` (DAG processor) and `Task-Sched` (task-to-PE scheduler), and their interaction through the ready and completed queues. *Right:* Algorithm for hybrid ranking of tasks using both $rank_{hom}$ and $rank_{het}$ heuristics.

two properties to calculate the SDR of the task, namely the worst-case execution time of the task ($WCET$), *i.e.* the time to execute on the slowest PE, and the path time (PT), calculated as the sum of the $WCET$s of the tasks in that path. The path with the largest PT in the DAG is designated as the critical path, and its PT is the critical path time (CPT). Additionally, a task's SDR is determined based on whether the task lies on a path intersecting the critical path or not. If it does, then the path is divided into two segments, the critical path segment with a critical-path segment time ($CPST$), and the non-critical path segment with non-critical-path segment time ($NCPST$). The SDR of tasks on such a path is calculated as follows:

$$SDR = \frac{WCET}{NCPST} \times \frac{CPT - CPST}{CPT} \quad (1)$$

Note that if the task's path does not intersect the critical path, SDR is computed with $NCPST$=PT and $CPST$=0. If the task's path is the critical path, $SDR = WCET/CPT$. Finally, the SD of the task is calculated as:

$$SD = SDR \times Deadline_{DAG} \quad (2)$$

Using the SD of a task, we compute a heuristic called effective slack of the task (*Eff_Slack*) that is representative of the urgency to execute the task. Quantitatively, it is the estimated time remaining until the deadline after completing the task on a PE, and is computed as shown below:

$$Eff\_Slack = SD - EET \quad (3)$$

where, SD is the task's sub-deadline and EET is the task's estimated execution time. Based on the manner in which ready tasks can be prioritized using their *Eff_Slack* and *Crit*, we present three ranking mechanisms.

Homogeneous Ranking: In this ranking scheme, we calculate $rank_{hom}$ as *Crit*/*Eff_Slack*, where the *Eff_Slack* is determined by using the $WCET$ of the task as the EET in Equation 3. Therefore, this ranking policy prioritizes critical tasks that have earlier deadlines over non-critical tasks that have later deadlines.

Heterogeneous Ranking: This ranking approach, $rank_{het}$, improves upon $rank_{hom}$ by accounting for the variation in the execution time of different PEs on the SoC, as well as by adopting contrasting approaches based on the task's *Crit*. The goal of the scheme is to allow for critical tasks to be prioritized on fast PEs and non-critical tasks to be run on slow PEs without blocking the fast PEs. For this, we first calculate *Eff_Slack* for

TABLE 2
System description of SoC for workload evaluation.

Workload	System	Description
Synthetic	Sys_A	8 single-core Arm Cortex-A57 CPUs 2 NVIDIA Maxwell GPUs 1 CNN/FFT accelerator [10], [11]
End-to-end	Sys_B	N_C† Single-core Arm Cortex-A57 CPUs N_G† NVIDIA Maxwell GPUs N_{traA}† tracking accelerators [2] N_{locA}† localization accelerators [2] N_{detA}† detection accelerators [2]

† N_C,N_G,N_{traA},N_{locA},N_{detA} are determined using design space exploration

TABLE 3
Timing and power profile of evaluated kernels on each PE type.

Suite	Task Type	Execution Time			Power (mW)		
		CPU	GPU	Accel	CPU	GPU	Accel.
Mini-ERA	2D Convolution	583*	349*	180*	634	2225	48
	Viterbi Decoder	1021*	20*	-	864	1228	-
	2D FFT	3193*	97*	4*	1036	6364	4
ADSuite	Object Detection	3531†	156†	96†	3654	467	28
	Object Tracking	1825†	17†	2†	5600	12790	590
	Localization	165†	95†	10†	6133	4457	22
	Mission Planning	1†	-	-	3534	-	-
	Motion Planning	8†	-	-	4222	-	-

- implementation not available *in micro-seconds †in milli-seconds

each PE type that the task can execute on, as shown in Figure 1 (right) (lines 1-3) using Equation 1, 2 and 3. We then use the task's *Crit* and *Eff_Slack* of each PE type to calculate $rank_{het}$. The $rank_{het}$ heuristic serves two purposes: (i) prioritization of non-critical tasks that can meet their deadlines on more than one type of PE over those that can meet their deadlines only when executed on a fast PE; and (ii) prioritization of critical tasks with earlier deadlines over those with later deadlines, while considering the PEs that it can execute on.

Hybrid Ranking: To differentiate between tasks of the same $rank_{het}$ in the heterogeneous scheme, we additionally, introduce the scheme of "hybrid ranking" that prioritizes tasks based on both $rank_{hom}$ and $rank_{het}$. However, unlike the homogeneous ranking scheme that uses $WCET$ for *Eff_Slack* to calculate $rank_{hom}$, we assign *Eff_Slack* based on the PEs that the task can execute on to meet its deadline. This method, shown in Figure 1 (right), is similar to how $rank_{het}$ is determined. This ranking mechanism allows for tasks having the same $rank_{het}$ to be prioritized using $rank_{hom}$ *i.e.* earliest deadline first. Hybrid ranking also allows for tasks that cannot be executed on all PE types to be prioritized using $rank_{hom}$ and $rank_{het}$ for eligible PE types. Note that this case is not depicted in Figure 1 (right).

Task Assignment: `Task-Sched` assigns ordered ready-tasks to eligible PEs, *i.e.* PEs that can execute the task, in a non-blocking manner. `Task-Sched` schedules a task to the PE that will result in the earliest estimated finish time (EFT) for the task. The EFT of a task is calculated using the execution time of the task on the eligible PEs, current busy status of the PE, and all tasks ahead of this task in the ready queue that are potentially scheduled to the same PE. Once a task is completed, `Task-Sched` pushes it into the completed queue for `Meta-Sched` to track dependencies and to obtain the data movement cost of its dependent tasks.

Deadline Tracking and Task Pruning: `Meta-Sched` can also elect to prune DAGs, *i.e.* not execute them at all any further, thus eliminating *Crit=1* tasks that will not meet their deadlines to favor *Crit=2* tasks. If the *Eff_Slack* on the fastest PE of a *Crit=1* task is negative, the task is not pushed into the ready queue. The rest of the DAG that the task belongs to is then pruned by `Meta-Sched`. We consider a non-preemptive scheduling model and do not prune tasks that are being executed on PEs.

Hardware Telemetry: Runtime information used by HetSched, gathered from hardware monitors in the SoC managed by `Task-Sched`, includes the busy status of PEs, estimated completion time for the busy PEs, execution time of a completed task, and energy consumed by a completed task.

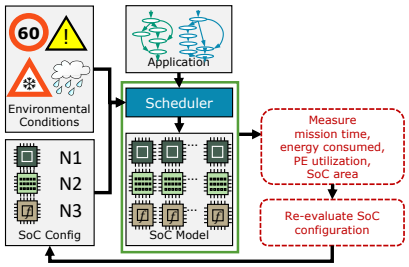

Fig. 2. Use of a real-time scheduler in a design space exploration loop to determine the best SoC design for AVs by considering the product of mission time, energy consumption and SoC area over the PE utilization.

4 System Methodology

4.1 System Simulation and Benchmarks

We profile (offline) a set of AV kernels on a Jetson TX1 board. We then simulate a heterogeneous SoC with multiple Cortex-A57 CPUs, Maxwell GPUs with 256 CUDA cores, and fixed-function accelerators for certain tasks. We consider two types of workloads for evaluation: a synthetic workload [12] and an end-to-end autonomous driving application [2]. The hardware description of the SoC used for these workloads and the task profiling results are shown in Table 2 and Table 3, respectively. For our evaluation, we derive the CPU and GPU area from publicly available technical specifications, and area for the accelerators from [2], [10], [11].

Synthetic Application Tasks. Our synthetic applications are comprised of three types of tasks: 2D Fast Fourier Transform (`fft`), 2D convolution (`conv`) and Viterbi decoding (`decoder`), taken from the Mini-ERA benchmark suite [12], which simulates a simplified autonomous vehicle.

End-to-End Autonomous Driving Tasks. We also consider an end-to-end autonomous driving application [2], referred to as ADSuite. It comprises of tasks like object detection, object tracking, localization, mission planning and motion planning.

Simulation Platform: Using the power and timing profile of tasks shown in Table 3, HetSched is evaluated on the STOMP scheduler platform [13]. STOMP is a queue-based discrete-event simulator used for early-stage evaluation of task scheduling mechanisms in heterogeneous platforms.

4.2 Trace Generation

For the synthetic workload, we generate traces of DAGs composed of 5 to 10 tasks each. Each DAG's critical path time (CPT) is set as its deadline. For ADSuite, we study scenarios that can lead to the execution of different sets of kernels and derive multiple DAGs. Following the discussion in [2], the deadline for each DAG is set to be 400 ms.

For each of the two workloads, we generate 1,000-DAG traces for the three congestion scenarios (urban, semi-urban and rural) with *Crit*=2 DAG fractions of 50% (urban), 20% (semi-urban) and 10% (rural). We then run these traces at varying DAG arrival rates, emulating different AV speeds, to determine the best mission time, where all *Crit*=2 DAGs meet their deadlines.

4.3 Design Space Exploration with Scheduler in the Loop

We use two state-of-the-art real-time schedulers, ADS [5] and 2lvl-EDF [4], as our baselines. We determine an efficient SoC configuration considering the use of ADS, 2lvl-EDF and HetSched as the scheduler in an optimization loop, as shown in Figure 2. We sweep the design space of the quantity of each PE type in Sys_B SoC, and evaluate ADSuite on each of the explored SoC configurations for the urban scenario. For each scheduler, we determine the "best" SoC configuration as the one that has the least product of mission time, energy consumption and SoC area

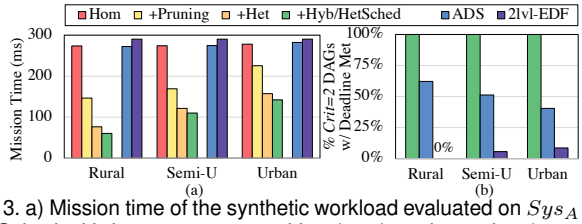

Fig. 3. a) Mission time of the synthetic workload evaluated on Sys_A using HetSched with homogeneous-ranking (`Hom`), task pruning (`Pruning`), heterogeneous-ranking (`Het`) and hybrid-ranking (`Hyb`), and baseline schedulers - ADS and 2lvl-EDF, for different congestion scenarios. b) % of *Crit=2* DAGs that meet deadline using HetSched, ADS and 2lvl-EDF when the AV operates at the maximum speed achievable by HetSched.

over the PE utilization for a successful mission, *i.e.* all *Crit=2* DAGs meet their deadlines. Although the composite metric is used to determine the best SoC configuration, we also evaluate the schedulers for each metric individually.

5 Evaluation

We evaluate each feature in HetSched and then compare it against state-of-the-art schedulers, ADS [5] and 2lvl-EDF [4].

5.1 HetSched Feature Evaluation

We incrementally evaluate the improvement in mission time of each of the features in HetSched, in comparison to the base version of HetSched with homogeneous ranking (`Hom`) for the synthetic workloads, as shown in Figure 3 (a). Note that mission time is only reported on successful completion of an AV mission *i.e.* all *Crit*=2 DAGs meet their deadlines. The task pruning feature that prunes non-critical tasks and DAGs that are unlikely to meet deadlines can provide 1.2-1.9× speedup in comparison to `Hom`. Additionally, employing the heterogeneous ranking mechanism that considers the variation in the execution profile of the SoC can provide 1.8-3.6× speedup over `Hom`. Lastly, using the hybrid ranking scheme provides a speedup of 4.6× over `Hom` for the rural scenario and 2.0× for the urban scenario. Moreover, HetSched with task pruning and hybrid ranking provides a speedup of 2.0-4.5× and 2.0-4.8× over ADS and 2lvl-EDF, respectively. Consequently, ADS and 2lvl-EDF are only able to meet deadlines for 40-62% and 0-9% of *Crit=2* DAGs, respectively, when operated at the maximum safe speed achieved by HetSched that meets 100% of all *Crit=2* DAG deadlines, as shown in Figure 3 (b).

5.2 SoC Design Optimization for Autonomous Driving

We determine the best SoC configuration for a highly congested urban scenario for ADS, 2lvl-EDF and HetSched, as shown in Figure 4. Note that, for brevity, the sweep for tracking and localization accelerators are not shown. The best SoC configurations determined using ADS, 2lvl-EDF and HetSched are $(8, 8, 8, 2, 2)$, $(8, 6, 8, 2, 2)$ and $(4, 4, 4, 2, 2)$, respectively, where (N_{detA},N_{traA},N_{locA},N_G,N_C) denotes N_{detA} detection accelerators, N_{traA} tracking accelerators, N_{locA} localization accelerators, N_G GPUs and N_C CPU cores. By exploiting the heterogeneous characteristic of the SoC and real-time requirements of AV applications (using hybrid and heterogeneous ranking, task pruning), the best SoC designed using HetSched has 2× and 1.8× lower number of accelerators, over the SoC derived using ADS and 2lvl-EDF, respectively, thereby increasing PE utilization by 4×. Moreover, for this SoC design, HetSched is able to achieve (1.7×, 1.7×) and (2.0×, 1.9×) better mission time and energy, respectively, over (ADS, 2lvl-EDF) for the urban scenario.

5.3 Scalability Discussion

Increase in *Crit=2* DAGs: As the congestion level increases, the number of *Crit=2* DAGs also increases. Although the mission time for HetSched with homogeneous ranking remains almost same, benefits from task pruning, and heterogeneous and hybrid ranking decreases with increasing congestion level (Figure 3 (a)).

Acc.	GPU	CPU Core 2	4	8
2	2	2.9	3.9	6.3
	4	4.5	5.6	7.9
	8	6.4	7.3	9.2
4	2	1.7	2.2	3.3
	4	2.6	3.1	4.1
	8	4.3	4.8	5.9
8	2	1.5	1.8	2.5
	4	1.7	1.9	2.4
	8	6.0	3.1	3.3

(a)

Acc.	GPU	CPU Core 2	4	8
2	2	2.9	3.9	6.3
	4	4.5	5.6	7.8
	8	6.3	7.2	9.1
4	2	1.7	2.2	3.3
	4	3.1	3.8	5.1
	8	4.2	4.7	5.8
8	2	1.5	1.8	2.5
	4	4.5	5.3	7.0
	8	5.9	6.7	8.2

(b)

Acc.	GPU	CPU Core 2	4	8
2	2	>1.0	1.4	2.4
	4	1.7	2.2	3.2
	8	2.7	3.1	4.0
4	2	1.0	1.3	2.1
	4	1.4	1.7	2.4
	8	2.5	2.9	3.7
8	2	2.5	3.2	4.6
	4	8.2	10.0	13.8
	8	4.4	5.1	6.4

(c)

Fig. 4. Normalized product of mission time, energy, SoC area over PE utilization for varying SoC configurations for ADSuite using (a) ADS, (b) 2lvl-EDF and (c) HetSched, in an urban scenario. We vary the count of CPUs, GPUs and accelerators. The best SoC configuration for each scheduler is denoted with the bordered green box.

Increase in number of PEs: As the number of PEs increase, the time to evaluate the fastest finish time in `Task-Sched` increases. For the SoC sizes evaluated in this work, the scheduler decisions for the 1000-DAG ADSuite missions are determined in less than 10% of the total mission time.

Increase in number of PE types: As the heterogeneity level of an SoC increases, *i.e.* increase in number of PE types and larger variability in the execution time of tasks on the PE types, HetSched achieves larger wins over baseline schedulers that do not exploit this characteristic of the heterogeneous SoC. For brevity, we have excluded this analysis from the paper.

6 Conclusion

We propose HetSched, a multi-level scheduler that exploits the highly-heterogeneous nature of the underlying systems-on-chips (SoCs) in conjunction with the characteristics of autonomous vehicle (AV) applications. The goal is to improve the overall mission time, while reducing the hardware requirement to enable fully autonomous vehicles. The paper demonstrates that HetSched can reduce the number of accelerators required in an SoC to complete AV's missions safely by 1.9×, while improving mission time by 1.7× on average, when compared against state-of-the-art schedulers for an end-to-end autonomous application.

References

[1] "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," https://www.sae.org/standards/content/j3016_201806/, 2018.

[2] S.-C. Lin *et al.*, "The architectural implications of autonomous driving: Constraints and acceleration," in *ASPLOS*, 2018.

[3] E. Talpes *et al.*, "Compute solution for Tesla's full self driving computer," *Micro*, 2020.

[4] P. Wu and M. Ryu, "Best Speed Fit EDF Scheduling for Performance Asymmetric Multiprocessors," *MPE*, 2017.

[5] G. Xie *et al.*, "Adaptive dynamic scheduling on multifunctional mixed-criticality automotive cyber-physical systems," *TVT*, 2017.

[6] "Road vehicles - functional safety - part 1: Vocabulary." https://www.iso.org/standard/68383.html, 2018.

[7] H. Topcuoglu *et al.*, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *TPDS*, 2002.

[8] K. Chronaki *et al.*, "Task scheduling techniques for asymmetric multi-core systems," *TPDS*, 2016.

[9] E. A. Capota *et al.*, "Towards mixed criticality task scheduling in cyber physical systems: Challenges and perspectives," *JSS*, 2019.

[10] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *JSSC*, 2016.

[11] M. Seok *et al.*, "A 0.27 V 30MHz 17.7 nJ/transform 1024-pt complex FFT core with super-pipelining," in *ISSCC*, 2011.

[12] "Mini-ERA: Simplified version of the main ERA workload," https://github.com/IBM/mini-era.

[13] A. Vega *et al.*, "STOMP: A tool for evaluation of scheduling policies in heterogeneous multi-processors," *arXiv:2007.14371*, 2020.