Energy-Efficient Adaptive System Reconfiguration for Dynamic Deadlines in Autonomous Driving

Saehanseul Yi*, Tae-Wook Kim†, Jong-Chan Kim†‡ and Nikil Dutt*
* Department of Computer Science, University of California, Irvine, USA
† Graduate School of Automotive Engineering, Kookmin University, Korea
‡ Department of Automobile and IT Convergence, Kookmin University, Korea
{saehansy, dutt}@uci.edu, {dsd8135, jongchank}@kookmin.ac.kr

Abstract—The increasing computing demands of autonomous driving applications make energy optimizations critical for reducing battery capacity and vehicle weight. Current energy optimization methods typically target traditional real-time systems with static deadlines, resulting in conservative energy savings that are unable to exploit additional energy optimizations due to dynamic deadlines arising from the vehicle’s change in velocity and driving context. We present an adaptive system optimization and reconfiguration approach that dynamically adapts the scheduling parameters and processor speeds to satisfy dynamic deadlines while consuming as little energy as possible. Our experimental results with an autonomous driving task set from Bosch and real-world driving data show energy reductions up to 46.4% on average compared with traditional static energy optimization methods, demonstrating great potential for dynamic energy optimization gains by exploiting dynamic deadlines.

I. INTRODUCTION

Due to the enormous amount of computing required for autonomous vehicles, its inherent high energy consumption has become one of the major hurdles when designing such computing systems. For example, even a single vehicle computer with multiple central processing units (CPUs) and graphics processing units (GPUs) alone reportedly consumes more than 2 kW of power, reducing an electric vehicle’s driving range up to 12% [1]. With this challenge, some energy optimization methods have been proposed that at the same time try to satisfy the stringent real-time requirements of automotive systems [2], [3]. However, since they commonly assume just (fixed) static deadlines, they do not reflect recent autonomous driving applications with time-varying dynamic deadlines. Such applications include the localization system with its dynamic latency constraint as a function of velocity [4] and the truck platooning system where its control response times are adjustable to varying driving conditions [5].

With the above motivation, we aim to develop an energy-efficient software optimization method and a runtime framework that can specifically exploit the dynamic nature of deadlines found in many autonomous driving applications. This study specifically focuses on minimizing CPU energy consumption by using dynamic voltage and frequency scaling (DVFS). Although conventional automotive microcontrollers often lack such features, recent application processors for autonomous driving mostly support DVFS.

Fig. 1: Example dynamic deadlines

To demonstrate our basic idea, Fig. [a] shows an example time history of a vehicle’s velocity and its corresponding dynamic deadline, assuming a velocity-deadline mapping function

\[ d(v) = \frac{-v + \sqrt{v^2 + 2a_{\text{max}}\lambda}}{a_{\text{max}}} \]

where \( a_{\text{max}} \) represents the minimum time for a vehicle to advance a fixed distance \( \lambda \) assuming its maximum acceleration \( a_{\text{max}} \). It is especially useful in truck platooning where trucks maintain a fixed longitudinal gap between them across various driving velocities [6] such that safe control decisions can be made more efficiently in terms of a maximum travel distance (i.e., \( \lambda \)) between sensing and actuation rather than by a rigid timing constraint. Hence, in the figure, the faster the vehicle runs, the shorter the deadline gets. The minimum deadline depicted by a red line is determined by the maximum velocity enforced by traffic regulations. Here, our basic idea is to trade off the area between the time-varying dynamic deadline and the minimum deadline to reduce energy consumption by adaptively slowing down the CPU to the extent that guarantees the dynamic deadline, rather than adhering to the static deadline [2], [3] as in the traditional energy optimization methods.

In the automotive industry, complex control applications composed of multiple independent real-time tasks are commonly modeled with directed acyclic graphs (DAGs) whose...
nodes are periodic tasks and edges are read-write dependencies between tasks. Fig. 2 shows an example DAG from Bosch for their reference autonomous driving system from sensors to actuators. Upon such a DAG, its dynamic deadlines are imposed by its worst-case end-to-end delays from sources to sinks, which is collectively decided by the periods of individual tasks. Then, our objective is to minimize the average power consumption while guaranteeing such dynamic deadlines. When doing that, we must satisfy (i) the system schedulability constraint (i.e., task periods) as well as (ii) the end-to-end deadline constraint. At first, we solve the problem by assuming a static deadline constraint. For that, we formulate it as a geometric programming (GP) problem, which is a special form of non-convex optimizations that can be efficiently solved by a transformation into a convex problem.

To extend the above optimization method to time-varying dynamic deadlines, we partition the feasible deadline range into a number of discrete modes, where the system is separately optimized for each mode, assuming each mode’s shortest deadline, respectively. Then, our runtime framework provides a safe mode change protocol that changes each task’s period when the vehicle slows down or speeds up crossing across different modes. Our mode change protocol is designed not to miss any deadline if the mode change is from a shorter deadline to a longer one. However, we found that extra delays are unavoidable in the opposite direction (i.e., longer to shorter deadlines). Even in that case, however, we provide a mode change delay analysis method from which we can reserve enough safety margins to hide away the extra delays.

Our experimental results show that our approach reduces the average energy consumption up to 46.4% in various real-world driving scenarios compared with the conventional method based on static deadlines. Moreover, our extensive simulation experienced no deadline miss due to our safe mode change protocol and delay analysis method. To the best of our knowledge, our work is one of the first attempts for optimizing energy consumption in computing systems for autonomous driving, explicitly focusing on dynamic deadlines.

This study’s contributions can be summarized as follows:

- We formulate an optimization problem for energy-efficient autonomous driving systems with time-varying dynamic deadlines and provide a GP-based optimal solution.
- We provide a safe mode change protocol that guarantees analyzable (if any) overheads, which can be safely manipulated in the design time.

The remainder of this paper is organized as follows: The next section describes the background and our problem. Section III presents our offline system optimization method. Section IV explains the dynamic system reconfiguration approach. Section V discusses practical issues. Section VI provides the evaluation results. Section VII presents related work. Finally, Section VIII concludes the paper.

1As an initial effort for the energy optimization based on dynamic deadlines, this study employs a rather simple system model, which will be extended in our future work. Its practical consideration will be discussed in Section V.
variables. Thus, a complete system configuration \( \pi \) can be described by a vector of tuples
\[
\pi = ((p_1, s_1), \ldots, (p_n, s_n)).
\] (6)

We assume the earliest deadline first (EDF) scheduling algorithm such that the L\&L utilization bound [13] can be used to test the system schedulability as
\[
U(\pi) = \sum_{i=1}^{n} \frac{e_i}{p_i s_i} \leq 100\%,
\] (7)
which has \( s_i \) in the denominator reflecting the effective WCET that is inverse-linearly proportional to \( s_i \). Based on the above schedulability test, we can guarantee every task’s periodicity.

B. Dynamic Deadlines

In our system model, when referring to deadlines, they always mean the end-to-end deadlines from sensors to actuators, not the per-task implicit deadlines that are equal to periods. Thus, even when the system is schedulable, satisfying every task’s \( p_i \), it does not mean deadlines will be guaranteed. To formally define our notion of deadlines, let us assume \( n_s \) sensor tasks (i.e., source nodes) and \( n_a \) actuator tasks (i.e., sink nodes) in \( G \). Then we say there are \( n_s \times n_a \) unique flows, each of which has at least one path that is a sequence of adjacent tasks fully connecting a flow. The set of paths in \( G \) is denoted by
\[
P = \{\delta_1, \delta_2, \ldots, \delta_P\},
\] (8)
where each path \( \delta_i \) denotes an ordered set of task indices following the path. For example, Fig. 2 has three \( (3 \times 1) \) flows and eight paths. Then deadlines are imposed upon the paths such that newly arrived sensor data at time \( t_1 \) propagates through the DAG until it first gets out of an actuator task at time \( t_2 \) within a given deadline \( d \) (i.e., \( t_2 - t_1 \leq d \)). In the automotive industry, the above notion is commonly referred to as reaction time constraints [14].

Fig. 3 shows continuous dynamic deadlines as the vehicle velocity changes, where vertical dashed lines depict discrete sensor arrivals. At each \( k \)-th sensor data arrival at time \( t[k] \), its dynamic deadline \( d[k] \) is decided as a red point by the vehicle velocity \( v[k] \) with a given mapping function (e.g., \( d(v) \) in Eq. (1a)). Thus, each sensor data arrival can be denoted by \((t[k], d[k])\) for \( k \geq 1 \). Although many variables representing other physical states can be considered, this study focuses on the velocity-dependent deadlines as an initial effort toward a more general framework.

To efficiently manage dynamic deadlines, we employ a multi-mode approach, where a feasible deadline range is partitioned into \( m \) discrete modes, where each mode guarantees the shortest deadline within its deadline range. For notational convenience, the modes are denoted by the per-mode shortest deadlines \( \{d_1, d_2, \ldots, d_m\} \).

In Fig. 3 its deadline domain is partitioned into six equal length modes, and at each sensor data arrival, the system mode is decided, possibly triggering mode changes. While the system is in a particular mode, the mode’s shortest deadline is guaranteed, as depicted by the thick blue line.

C. Power Model

We use the following popular power model from [12], [15]:
\[
P(s) = P_s + P_d(s) = \beta + \alpha s^\gamma,
\] (9)
where \( P_s \) is the static power and \( P_d(s) \) is the dynamic power parameterized by a speed factor \( s \). The static power is expressed as \( \beta \) independent of other parameters, whereas the dynamic power depends on \( s \) while \( \alpha \) and \( \gamma \in [2, 3] \) are CPU-dependent parameters. In this work, we do not use CPU sleep states to reduce the static power, so our focus is to minimize the average dynamic power by minimizing \( s \) as long as satisfying the dynamic deadlines.

For that, the average dynamic power can be calculated as follows: For each task \( t_i \), its instantaneous dynamic power \( \alpha s_i^\gamma \) is maintained during task’s effective execution time \( e_i/s_i \) affected by the speed factor. Since the power pattern repeats by its period \( p_i \), the average power consumed in a unit time while executing \( t_i \) can be calculated as
\[
P_i(p_i, s_i) = \frac{\alpha s_i^\gamma \times \frac{e_i}{s_i}}{p_i} = \frac{\alpha s_i^\gamma e_i}{p_i},
\] (10)
which is a function of \( p_i \) and \( s_i \). Then, by summing up \( n \) such average powers with the static power and the idle-time CPU power at \( s_{min} \), the total average power for the system is given as a function of a system configuration \( \pi \) as
\[
P(\pi) = \beta + \alpha \sum_{i=1}^{n} \frac{s_i^{-1} e_i}{p_i} + \alpha s_{min}^\gamma \left(1 - \sum_{i=1}^{n} \frac{e_i}{p_i s_i}\right).
\] (11)

D. Problem Description

With the above system model, multi-mode deadline constraints, and energy model, our problem can be described as follows: Given a DAG \( G \) of \( n \) periodic tasks and \( m \) discrete deadline constraints, our problem is to find the following two matrices:
\[
\begin{pmatrix}
p_1^1 & p_2^1 & \cdots & p_n^1 \\
p_1^2 & p_2^2 & \cdots & p_n^2 \\
\vdots & \vdots & \ddots & \vdots \\
p_1^m & p_2^m & \cdots & p_n^m
\end{pmatrix}
\] and
\[
\begin{pmatrix}
s_1^1 & s_2^1 & \cdots & s_n^1 \\
s_1^2 & s_2^2 & \cdots & s_n^2 \\
\vdots & \vdots & \ddots & \vdots \\
s_1^m & s_2^m & \cdots & s_n^m
\end{pmatrix},
\] (12)
where each \( p_i^j \) and \( s_i^j \) represent \( t_i \)'s optimal period and speed factor at the \( j \)-th mode, respectively, in terms of system-wide energy efficiency. Besides, the solution should satisfy safe mode changes such that the system can freely go back and forth between modes without violating the dynamic deadline requirements. For that, a safe runtime mode change protocol
should be developed along with a mode change delay analysis method.

III. PERIODS AND SPEED FACTORS OPTIMIZATION

This section formulates and solves the multi-mode system optimization problem. We begin by finding the optimal configuration assuming a single fixed mode without considering mode changes (Section III-A). Then, we extend the optimization method such that it can guarantee safe mode changes (Section III-B).

A. Single-Mode Formulation

This section explains how we can formulate a single-mode optimization as a baseline for the multi-mode system optimization. As the objective function, the average power in Eq. (11) is used, without the constants \( \alpha \) and \( \beta \), where \( \pi \) denotes two sets of decision variables \( p_i \) and \( s_i \) with constrained domains as \( p_i > 0 \) and \( s_{min} \leq s_i \leq 1 \), respectively.

Then, we have two explicit constraints: (i) schedulability constraint as already discussed in Eq. (7) and (ii) deadline per-task utilization invariability, which we add a new constraint called coexistence during a mode change, schedulability violations can occur even if each mode is schedulable in isolation [16]. Thus, we add a new constraint called per-task utilization invariability, meaning every task’s utilization \( u_i = e_i/(p_i s_i) \) is invariant across modes as

\[
\frac{e_i}{p_i s_i^1} = \frac{e_i}{p_i s_i^2} = \cdots = \frac{e_i}{p_i s_i^m} = u_i^* \quad (\forall i = 1, 2, \ldots, n),
\]

where \( u_i^* \) denotes identical utilization for \( \tau_i \) across modes. In that manner, even in the transient interval, the system’s instantaneous utilization is maintained unchanged, which in turn guarantees the system schedulability [17]. Now, we use \( u_i^* \)’s as our decision variables, replacing \( s_i^r \)’s, which can be decided later by \( s_i^r = e_i/(p_i u_i^*) \). Then our multi-mode optimization can be formulated as follows:

\[
\begin{align*}
& \text{minimize} \quad P(\hat{\pi}) = \sum_{j=1}^{m} \sum_{i=1}^{n} \frac{e_i^\gamma}{(p_i^j)^\gamma(u_i^*)^{\gamma-1}} + s_{min}^{\gamma}(1 - \sum_{i=1}^{n} u_i^*) \\
& \text{subject to} \quad U(\pi) = \sum_{i=1}^{n} u_i^* \leq 1, \\
& \quad \quad \quad D_\delta(\pi) = \sum_{i=1}^{n} 2p_i^\delta \leq \delta^d \quad (\forall j \in [1, m], \forall \delta \in \mathbb{P}),
\end{align*}
\]

(16)

C. Geometric Programming-based Optimization

Our multi-mode optimization problem can be efficiently solved by GP, which is a mathematical optimization method for solving specially formed optimization problems through logarithmic transformations into convex ones. As a result, GP always finds the (true, globally) optimal solution when the problem is feasible [9]. To use GP, the objective function and inequality constraints should be constructed by the special form polyonimal, as in

\[
\begin{align*}
& f(x) = \sum_{k=1}^{K} c_k x_1^{a_{1k}} x_2^{a_{2k}} \cdots x_n^{a_{nk}},
& \text{with decision variables } x_i, \text{ non-negative coefficients } c_k, \text{ and real-valued exponents } \{a_{11}, \cdots, a_{nk}\}. \quad \text{Our objective functions and constraints are in polynomial forms except for the idle-time CPU power terms in the rightmost part of } P(\pi) \text{ in Eq. (14) and } P(\hat{\pi}) \text{ in Eq. (16).}
\end{align*}
\]

However, it can be removed without affecting optimality as long as \( \exists s_i \neq s_{min} \). The optimal solutions in such cases always have 100% system utilization without any idle time. To prove it intuitively, assume the system utilization \( U < 100\% \), if we pick a certain task \( \tau_i \) and decrease \( s_i \), thus increasing \( e_i \), until \( U \) reaches 100%, the average power of \( \tau_i \) will decrease and the idle power term will disappear, eventually saving more energy than the original configuration. However, when we cannot reduce processor speeds (\( \forall i : s_i = s_{min} \)), the optimal case would have \( U < 100\% \) as \( p_i \) increases. Such cases only occur when the deadline is extremely long after all \( s_i \)’s are bounded by \( s_{min} \). We are not considering such extreme cases in this work.
IV. SAFE MODE CHANGE FOR SYSTEM RECONFIGURATION

For safe system reconfigurations with dynamic deadlines, the followings should be respected even during mode changes:

- **Periodicities.** Every task period before and after the mode change instant should be guaranteed, which can be satisfied by the per-task utilization invariability constraint introduced in Section III-B.

- **Deadlines.** Unfortunately, however, the above periodicities do not guarantee dynamic deadlines, which span across multiple tasks possibly with different modes during a mode change.

Thus, this section focuses on developing a safe mode change protocol in terms of end-to-end dynamic deadlines based on the already guaranteed per-task periodicity.

Assume the system is switching from an old mode to a new mode, represented by each mode’s shortest deadlines, respectively, by

\[ d^{\text{old}} \rightarrow d^{\text{new}}. \]  \hfill (17)

When \( d^{\text{old}} < d^{\text{new}} \), it is termed as **relaxing deadlines** and in the opposite as **shrinking deadlines**. System configurations for each mode are denoted by \( \pi^{\text{old}} \rightarrow \pi^{\text{new}} \), that is,

\[
((p_{1}^{\text{old}},s_{1}^{\text{old}}),\ldots,(p_{n}^{\text{old}},s_{n}^{\text{old}})) \rightarrow
((p_{1}^{\text{new}},s_{1}^{\text{new}}),\ldots,(p_{n}^{\text{new}},s_{n}^{\text{new}}))
\]  \hfill (18)

in its expansion form. A mode change is triggered by a sensor data arrival at time \( t_0 \) with its dynamic deadline falling above or below the old range. Then, we consider the following mode change methods, as depicted in Fig. 4.

- **ALAP (As Late As Possible)** individually triggers per-task mode changes after the new data make progress to every incoming edge of it. The actual mode changes will happen at the nearest period boundary after the trigger.

- **AEAP (As Early As Possible)** immediately triggers every task regardless of the new data’s progress. The mode change completes by \( t_0 + \max_{\delta}(p_{i}^{\text{old}}) \) in the worst case when the longest period of \( \pi^{\text{old}} \) began right before \( t_0 \).

Then we deal with the two cases: (i) relaxing and (ii) shrinking deadlines with the above methods, respectively.

(i) **Relaxing deadline.** In this case, we use ALAP such that the mode changes do not adversely affect the already ongoing progress of old sensor data. Besides, we need to ensure that the new sensor data do not violate \( d^{\text{new}} \). Note that the new sensor data may progress through tasks possibly with different modes, which happens due to different speeds of different paths. For example, in Fig 4(a), \( \tau_2 \) has two incoming edges, where the upper path requests the mode change while the slower lower path still retains the old mode. Then the upper path \( \tau_1 \rightarrow \tau_2 \rightarrow \tau_4 \rightarrow \tau_5 \) can have a mixture of both modes while handling the new sensor data. Thus, the worst-case delay for the new sensor data during ALAP mode changes can be calculated as in the following:

\[ D^{\text{new}}(\pi^{\text{old}} \rightarrow \pi^{\text{new}}) = \max_{\delta \in \mathbb{D}} \left( \sum_{i=1}^{\delta} 2 \max(p_{i}^{\text{old}},p_{i}^{\text{new}}) \right) \leq \max_{\delta \in \mathbb{D}} \left( \sum_{i=1}^{\delta} 2p_{i}^{\text{new}} \right) = d^{\text{new}}, \]  \hfill (19)

which is less than \( d^{\text{new}} \) since \( \forall i : p_{i}^{\text{old}} \geq p_{i}^{\text{new}} \) that is true when relaxing deadlines. This is because, when the task utilization is fixed, the value of \( p_i s_i \) should remain the same across modes. Therefore, \( p_i \) should increase monotonically to decrease \( s_i^{-1} \) in Eq. (11), reducing the average power in the next longer deadline mode.

(ii) **Shrinking deadlines.** In this case, which basically makes the situation more challenging, we use AEAP to quickly finish mode changes, minimizing possible extra delays. Delays for the old sensor data are naturally kept less than \( d^{\text{old}} \) by the same rationale in Eq. (19) since \( \forall i : p_{i}^{\text{old}} \geq p_{i}^{\text{new}} \) when shrinking deadlines. However, regarding the new sensor data, it can suffer extra delays if any task’s old period instance that began before the new sensor data arrival persists long enough such that the new data’s progress is unexpectedly delayed by that persisting old task instance. Algorithm 1 calculates the worst-case delay considering such negative effects for each path \( \delta \), which is an ordered set of task indices in each path. Among the calculated delays, we can find the longest. The algorithm gradually accumulates delays by tasks in \( \delta \). Line 1 indicates that it is unavoidable for the new sensor data to be *waited* by the old period at the first task. Then we have two cases for the remaining tasks: (i) its mode is already changed before the data progress arrives (Line 4) and (ii) an old instance persists (Line 6). In the former, we simply accumulate the new delay component \( 2p_{i}^{\text{new}} \) in the latter, the persisting old (long) period \( p_{i}^{\text{old}} \) hides away the accumulated delay up to then, resetting it to \( p_{i}^{\text{old}} + p_{i}^{\text{new}} \).

By the above analyses, we claim that when relaxing deadlines with ALAP, there is no deadline miss for both the already ongoing progress and new ones. When shrinking deadlines with
AEAP, the already ongoing progress rather benefits from it, whereas new sensor data can suffer extra delays. However, we can analyze the worst-case extra delays, which can be used when planning appropriate safety margins in design time.

V. PRACTICAL CONSIDERATIONS

A. Extension to Discrete CPU Frequency Levels

Our optimization method yields speed factors in the continuous range between $s_{\text{min}}$ and 1. However, because most CPUs, in practice, support only a predefined set of discrete frequency levels, we need to adapt the resulting speed factors to the discrete domain. One possible approach is to emulate the exact speed factors by modulating between two neighboring discrete frequency levels \cite{18}, \cite{19} to obtain the near-optimal energy reduction similar to the continuous frequency solution. However, using such intra-task DVFS entangles other practical considerations such as extra time and energy overhead associated with excessive frequency transitions \cite{20} and possible transient faults \cite{21}, \cite{22}.

In light of this, we propose a more conservative but safer method that uses the closest frequency level that is higher than the corresponding optimal speed factor. Then every task’s actual utilization is less than or equal to the ideal utilization in accordance with the per-task utilization invariance constraint. Thus, even though the approximated system will consume more energy than the continuous one, it safely ensures schedulability and end-to-end delay constraint during mode changes.

B. Extension to Multicore CPUs

Our system model assumes a singlecore CPU, which is still dominant in the automotive industry due to the safety concerns and software complexity caused by multicore CPUs. Nonetheless, high performance multicore CPUs have recently gained acceptance in the industry, necessitating the extension of our method to multicore CPUs. Although this issue is not within this paper’s scope, this section briefly discusses some ideas for the extension.

One possible approach is to incorporate one of the global multicore scheduling algorithms that still provide utilization-based schedulability analysis. Then, only minor modifications are necessary in the optimization formulation, and also the mode change protocols can be easily reused. However, this extension approach requires multiple tasks executing in parallel to have different CPU speed levels, which is difficult to implement because most CPUs do not support per-core DVFS in practice.

Another approach, which seems more practical, is to use the RT-Gang scheduling framework \cite{23} into our method. RT-Gang may eliminate many multicore issues including per-core DVFS and unpredictable inter-core memory bandwidth contention by its one-gang-at-a-time policy. Here, a gang is a predefined group of tasks that is allowed to run in parallel. After the gang grouping, the system model is transformed as in our singlecore model, where despite the gang grouping problem, our optimization method and mode change protocol can be used without modification. For the remaining gang grouping, we leave it for future work.

VI. EXPERIMENTS

A. Experimental Setup

Workload. We use the DAG in Fig. 2 with ten tasks, where their WCETs are listed in Table. I. They are derived from the actual measurements \cite{24} on Nvidia Jetson TX2 platform (Denver cores) and scaled considering typical high-performance computing systems for autonomous driving. Unfortunately, these industry-level applications are IP protected, so we were not able to run them on a real platform.

Energy Model. We empirically found the energy parameters for Eq. (11), as $\alpha=842.04$, $\beta=232.81$, and $\gamma=2.64$, on the same hardware platform in \cite{24}.

Discrete frequency levels. We use 12 evenly spaced frequencies between 345 MHz and 2 GHz from the same hardware platform in \cite{24}.

Scenarios. We use real-world driving scenarios from the comma.ai driving dataset \cite{7}, where we picked ten 60-second driving logs with their velocity from 0 km/h to 114 km/h as depicted in Fig. 5.

Dynamic Deadlines. We converted velocities into deadlines using Eq. (1) with $\lambda=20$ m and $\alpha_{\text{max}}=2.5$ ms$^{-2}$. The shortest deadline (617 ms) is obtained using the highest velocity (114 km/h), while the longest deadline (2945 ms) is bounded by $s_{\text{max}}=0.17$. Thus, the number of modes is chosen arbitrarily, $m=24$, partitioning the range with equal length.

Optimization. For the GP solver, we use CVX \cite{25} on a laptop with a quad-core Intel Core i7@2.6 GHz CPU, where our optimization takes about 1.46 seconds.

Simulation. We implemented a simulator supporting the EDF scheduling and our mode change protocol, by which we can precisely estimate the exact task schedules, end-to-end delays, and energy consumption.

B. Evaluation

The goal of our evaluation is twofold: (i) illustrate the benefit of using multiple frequency levels in dynamic deadline situation over conventional methods that utilize only one or two frequencies (ii) evaluate the effectiveness of our mode change protocol. It is of no use if it compromises safety. The following three methods are compared in the evaluation:

- Baseline: $\forall i : s_i = 1$;
- Static: optimized for the minimum deadline (617 ms);
Fig. 5: Real-world driving scenarios from comma.ai dataset

| Task                    | Camera Grabber | Lidar Grabber | CAN  | SPM  | Lane detection | Detection | Loc  | EKF  | Planner | DASM |
|-------------------------|----------------|---------------|------|------|----------------|-----------|------|------|--------|------|
| WCET (ms)               | 0.25           | 2.75          | 0.15 | 6.95 | 10.55          | 29.00     | 73.70| 1.10 | 3.10   | 0.325|

Fig. 6: Dynamic power optimization results

- **Multi mode (OURS):** method in Subsection III-B

Fig. 7 shows the dynamic power optimization results of each method. For Single mode, we solved the problem in Subsection III-A repeatedly for each deadline. In the leftmost mode with the shortest deadline, the three methods except the baseline method show an equal result (58.9%). However, as the deadline increases, the dynamic power plunges to 1.0% in the rightmost mode with the longest deadline. Another interesting observation is that our multi-mode results do not reveal visible performance degradation compared with the single-mode results, meaning that the per-task utilization invariability constraint does not critically affect the optimization results. On top of that, there is a 75% reduction in overall optimization time when using multi-mode formulation thanks to fewer decision variables.

Fig. 7 shows the simulation results with one of our driving scenarios. Only in this experiment, we partition the deadline range into six equal length modes for better visualization. As the deadline changes, as depicted by the green dashed line, mode changes are sporadically triggered, depicted by the rising and falling edges of the blue line that roughly follows the dynamic deadline. Note the small red staircase shapes at each falling edge, which depicts the extra delays when shrinking deadlines analyzed by Algorithm 1. More specifically, their height represents the extra delay, while their width represents the transient interval for each mode change. As shown in the figure, we do not have any deadline miss in this scenario even after considering the extra mode change delays. Additionally, the bottom plot shows how the normalized dynamic power varies (between 1.0% and 13.3%) depending on the mode changes.

Fig. 8 shows the energy consumption, including static and dynamic power, in driving scenarios using continuous frequency. Though Static achieves a significant energy reduction of 32% from baseline, Multi mode reduces further by timely
utilizing lower frequencies. Scenarios #3, #4, and #6 show relatively low energy reductions since they maintain the vehicle within a high-velocity range most of the time, making the system remain in the short deadline modes. In the others, more than half of the total energy was saved. On average, our method achieved 69.3% and 54.8% energy reductions compared with the baseline and the static methods, respectively.

Fig. 9 shows the energy consumption using discrete frequencies as discussed in Section V-A. A discrete frequency that is greater than or equal to the optimal one is assigned to each task. Therefore, we expect the energy consumption will increase in both methods. Static achieves 24% of energy reduction, which is smaller than the previous case. Multi mode still shows a significant reduction from Static, however, the energy gap between them becomes smaller. In scenarios #3, #4, and #6, the reduced gap is very notable. This is caused by the nonlinearity in the power model in terms of speed factor. The difference between the optimal and mapped discrete frequency yields bigger difference in power consumption in the higher frequency range. On average, there was a 31.8% increase in energy consumption when mapped to discrete frequencies. Our method achieved 59.3% and 46.4% energy reductions on average compared with the baseline and the static methods, respectively.

Throughout the experiments with the industry-level applications and real-life driving scenarios, we could not find any deadline violation with our mode change protocols. However, as we discussed in Section IV the extra delays when shrinking deadlines are unavoidable and they could be an issue depending on vehicle’s maximum acceleration and mode lengths. In that case, the violations can be avoided by selecting a shorter deadline mode considering possible mode change delays analyzed in Section IV(i.e. a safety margin) in design time analysis. We leave the choice of optimal margin for our future work since it requires a thorough analysis.

VII. RELATED WORK

Dynamic deadlines. Recent studies [26], [27] presented motivating examples of dynamic deadlines in autonomous driving, where object detection systems are commonly proposed that adapt themselves to varying deadlines, demonstrating the unique potential of autonomous driving systems. More specifically, Lee and Nirjon [26] support dynamic deadlines with selective subgraph executions by considering varying time budgets. Heo et al. [27] support dynamic deadlines by selectively executing multiple forward propagation paths of a neural network with different execution times. Both studies trade dynamic deadlines (or slacks) for improving the object detection accuracy. However, little work has been done with dynamic deadlines for the energy optimization of autonomous driving.

DVFS-based energy optimization. There have been many efforts to develop energy-efficient real-time systems, most of which, however, assume only static deadlines. Broadly, there are two frequently used energy saving approaches in hard real-time systems: DVFS and dynamic power management (DPM). In DVFS approaches, there is a body of literature to find speed levels through offline optimization [12], [28]–[30]. These approaches try to find critical speed factors under a static deadline constraint, which is the lowest frequency satisfying the given static timing constraint. In contrast, our method finds the critical speed for each deadline through multiple modes with different timing constraints and employs a safe mode change protocol to freely go back and forth between them. Another body of real-time DVFS approaches tries to reclaim dynamic slacks [31], [32], which is not to be confused with our dynamic deadlines; instead, they define dynamic slacks as the difference between the worst-case and the actual execution times. Note that dynamic slack reclaiming does not conflict with our approach and could be used together for further energy reductions.

DPM-based energy optimization. In DPM approaches, cores are switched off during idle periods to reduce energy consumption. It is useful when the system is underutilized. However, each idle period should be large enough to offset the energy overhead of frequent switching on and offs. Many scheduling methods have been proposed to create large idle periods [33]–[36]. Lee et al. [33] proposed leakage control EDF scheduling, but it requires additional hardware to calculate the sleep period. This impractical assumption was avoided in [35] by using a simpler sleep period calculation method and the enhanced race-to-halt (ERTH) algorithm. With ERTH, tasks run at full speed to secure longer sleep time. DPM and DVFS approaches are not orthogonal and can be used in conjunction. However, satisfying the timing constraints during mode changes with DPM approaches requires non-trivial considerations. Thus, we leave integrating DPM for our future work.

VIII. CONCLUSION

This study is motivated by emerging autonomous driving applications with time-varying dynamic deadlines, where the computing system’s excessive energy consumption is a major concern. Unlike traditional energy optimization methods assuming rigid static deadlines, our solution approach tries to utilize the dynamic slack obtained by adaptively relaxing deadlines considering the vehicle’s physical state. For that, our GP-based optimization method proactively exploits the dynamic deadlines to find energy-efficient multi-mode system configurations. Moreover, our safe mode change protocol enables adaptive system reconfiguration between the predefined modes. Our experimental results show an average of 46.4% energy reduction from the previous method with static deadlines, demonstrating the great potential toward energy-efficient autonomous driving systems.

Based on the theoretical foundation presented in this paper, in the future, we plan to extend our work to incorporate other energy saving techniques into our adaptive reconfiguration and to consider practical issues in COTS CPUs including multicore processors. Also, we plan to extend our system model considering other accelerators such as GPUs.

ACKNOWLEDGMENT

This work was supported partially by NSF grant CCF-1704859 and partially by the Ministry of Land, Infrastructure,
and Transport (MOLIT), Korea, through the Transportation Logistics Development Program (20TLRP-B147674-03, Development of Operation Technology for V2X Truck Platooning). J.-C. Kim is the corresponding author.

REFERENCES

[1] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, “The architectural implications of autonomous driving: Constraints and acceleration,” in Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, 2018, pp. 751–766.

[2] S. Kehr, E. Quiñones, D. Langen, B. Bödkeker, and G. Schäfer, “Parsc: energy-aware and robust parallelization of autosar legacy applications,” in 2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). IEEE, 2017, pp. 343–352.

[3] G. Xie, H. Peng, Z. Li, J. Song, Y. Xie, R. Li, and K. Li, “Reliability enhancement toward functional safety goal assurance in energy-aware autonomous cyber-physical systems,” IEEE Transactions on Industrial Informatics, vol. 14, no. 12, pp. 5447–5462, 2018.

[4] T. G. Reid, S. E. Houts, R. Cammarata, G. Mills, S. Agarwal, A. Vora, and G. Pandey, “Localization requirements for autonomous vehicles,” SAE International Journal of Connected and Automated Vehicles, vol. 2, no. 3, Sep 2019.

[5] L. Davis, “Stability of adaptive cruise control systems taking account of vehicle response time and delay,” Physics Letters A, vol. 376, no. 40-41, pp. 2658–2662, 2012.

[6] A. Divila and M. Nombela, “Safte-safe road trains for the environment reducing fuel consumption through lower aerodynamic drag coefficient,” SAE Technical Paper, Tech. Rep., 2011.

[7] comma.ai, “comma.ai driving dataset.” [Online]. Available: https://archive.org/details/comma-dataset

[8] A. Hamann, D. Darsari, and F. Wurst, “Waters industrial challenge,” 2019. [Online]. Available: https://www.ecrts.org/forum/viewtopic.php

[9] S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi, “A tutorial on geometric programming,” Optimization and engineering, vol. 8, no. 1, p. 67, 2007.

[10] M. Grant and S. Boyd, “CVX: Matlab software for disciplined convex programming, version 2.1,” Mar. 2014. [Online]. Available: http://cvxr.com/cvx

[11] Z. Guo, A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, and N. Guan, “Energy-Efficient Real-Time Scheduling of DAGs on Clustered Multi-Core Platforms,” in IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). IEEE, 2020, pp. 174–187.

[12] T. S. ISO 15502-1, “Sae j2735: Vocabulary for real-time systems.” [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_view.aspx?catalogue_id=26247

[13] T. Ishihara and H. Yasuura, “Voltage scheduling problem for dynamically variable voltage processors,” in Proceedings. 1998 International Symposium on Low Power Electronics and Design (IEEE Cat. No.98TH8379), 1998, pp. 197–202.