EASYR: Energy-Efficient Adaptive System Reconfiguration for Dynamic Deadlines in Autonomous Driving on Multicore Processors

SAEHANSEUL YI, University of California, Irvine
TAE-WOOK KIM and JONG-CHAN KIM, Kookmin University
NIKIL DUTT, University of California, Irvine

The increasing computing demands of autonomous driving applications have driven the adoption of multicore processors in real-time systems, which in turn renders energy optimizations critical for reducing battery capacity and vehicle weight. A typical energy optimization method targeting traditional real-time systems finds a critical speed under a static deadline, resulting in conservative energy savings that are unable to exploit dynamic changes in the system and environment. We capture emerging dynamic deadlines arising from the vehicle's change in velocity and driving context for an additional energy optimization opportunity. In this article, we extend the preliminary work for uniprocessors [66] to multicore processors, which introduces several challenges. We use the state-of-the-art real-time gang scheduling [5] to mitigate some of the challenges. However, it entails an NP-hard combinatorial problem in that tasks need to be grouped into gangs of tasks, gang formation, which could significantly affect the energy saving result. As such, we present EASYR, an adaptive system optimization and reconfiguration approach that generates gangs of tasks from a given directed acyclic graph for multicore processors and dynamically adapts the scheduling parameters and processor speeds to satisfy dynamic deadlines while consuming as little energy as possible. The timing constraints are also satisfied between system reconfigurations through our proposed safe mode change protocol. Our extensive experiments with randomly generated task graphs show that our gang formation heuristic performs 32% better than the state-of-the-art one. Using an autonomous driving task set from Bosch and real-world driving data, our experiments show that EASYR achieves energy reductions of up to 30.3% on average in typical driving scenarios compared with a conventional energy optimization method with the current state-of-the-art gang formation heuristic in real-time systems, demonstrating great potential for dynamic energy optimization gains by exploiting dynamic deadlines.

CCS Concepts: • **Computer systems organization → Real-time systems**; *Real-time system specification;*

Additional Key Words and Phrases: Directed acyclic graph (DAG), dynamic voltage and frequency scaling (DVFS), voltage-frequency island (VFI), earliest deadline first (EDF), energy-efficient, real-time

This work was supported partially by NSF grant CCF-1704859 and partially by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (2022R1A2C1013197).

Authors' addresses: S. Yi and N. Dutt, University of California, Irvine, CA, 92697, USA; email: saehansy@uci.edu; T.-W. Kim and J.-C. Kim (corresponding author), Kookmin University, Seoul, Korea; emails: {dsd8135, jongchank}@kookmin.ac.kr, dutt@ics.uci.edu.

1539-9087/2023/04-ART52 $15.00
https://doi.org/10.1145/3570503

ACM Reference format:
Saehanseul Yi, Tae-Wook Kim, Jong-Chan Kim, and Nikil Dutt. 2023. EASYR: Energy-Efficient Adaptive System Reconfiguration for Dynamic Deadlines in Autonomous Driving on Multicore Processors. *ACM Trans. Embedd. Comput. Syst.* 22, 3, Article 52 (April 2023), 29 pages.
https://doi.org/10.1145/3570503

1 INTRODUCTION

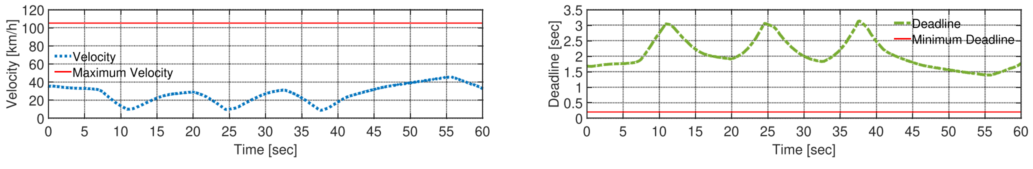
Due to the enormous amount of computing required for autonomous vehicles, their inherent high energy consumption has become one of the major hurdles when designing such computing systems. For example, Gawron et al. [35] transformed a commercial **Electric Vehicle (EV)** into a **Connected and Automated Vehicle (CAV)** by installing sensors such as camera, radar, and LiDAR in addition to computers and a short-range communication device to evaluate the energy consumption and greenhouse gas emission of the future CAVs. According to their report, the computer system is taking 80% of the added power consumption. The computer system of a CAV composed of multiple **Central Processing Units (CPUs)** and graphics processing units reportedly consumes more than 2 kW of energy, reducing an EV's driving range up to 12% [50]. The current energy consumption of EVs is around 200 Wh/km on average of a total of 245 cars [27]. That is 10-kW energy consumption at the speed of 50 km/hour (about 30 mph). Therefore, even though the current CAV has not reached full automation, its computer system can consume up to 20 % of the vehicle's power consumption. As autonomous driving technologies evolve, more sensors will be added, requiring more processing and computing power, so the portion of computing systems in energy consumption is expected to increase. In addition to that, according to a recent article regarding vehicle computing [52], the U.S. national energy consumption of EVs is approximately equal to the total energy consumption of 15 representative technology companies' data centers, each of which can consume 12 terawatt-hours, taking the significant portion of the overall energy consumption. Since the volume of EV energy consumption is huge, even a small improvement would be meaningful in terms of global carbon emission reduction. With this challenge, some energy optimization methods have been proposed that simultaneously try to satisfy the stringent real-time requirements of automotive systems [47, 63]. 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 [56] and the truck platooning system where its control response times are adjustable to varying driving conditions [30].

With the preceding 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 focuses on minimizing CPU energy consumption using **dynamic voltage and frequency scaling (DVFS)**. Although conventional automotive microcontrollers often lack such features, recent application processors for autonomous driving mostly support DVFS.

To demonstrate our basic idea, Figure 1 shows an example time history of a vehicle's velocity and its corresponding dynamic deadline, assuming a velocity-deadline mapping function

$$d(v) = \frac{-v + \sqrt{v^2 + 2\lambda a^{max}}}{a^{max}}, \tag{1}$$

which represents the minimum time for a vehicle at its initial velocity v to advance a fixed distance λ assuming its maximum acceleration a^{max}. It is especially useful in truck platooning where trucks maintain a fixed longitudinal gap between them across various driving velocities [29] such that



(a) Example Time History of Vehicle Velocity [23]

(b) Corresponding Dynamic Deadlines (λ=20 m and $a^{\max}$ =2.5 ms^{-2})

Fig. 1. Example dynamic deadlines.

safe control decisions can be made more efficiently in terms of a maximum travel distance (i.e., λ) 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 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 [47, 63] 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. Figure 2(a) shows an example DAG from Bosch [39] 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 latency 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 [18], which is a special form of non-convex optimization that can be efficiently solved by a transformation into a convex problem.

To extend the preceding 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 while guaranteeing safety.

The basic formulation for uniprocessors was done in our previous work [66]. In this work, we achieve the same goal in multicore processors with **Voltage-Frequency Island (VFI)** architecture, where all CPU cores in the same island share the same voltage-frequency level. VFI is most common in commercial processors because the complex design for supporting per-core DVFS cannot justify the potential energy gain [41].

Transitioning from single-core to multicore introduces several challenges: inter-core interference and multicore real-time schedulability. In addition to the NP-hard task-to-core assignment, the inter-core interferences and complex architecture of multicore processors make timing analysis notoriously difficult. Because of that, it led to strict timing constraints based on a highly pessimistic estimation of **Worst-Case Execution Time (WCET)**, resulting in less number of schedulable tasks and low system utilization. Recently, **Real-Time Gang (RT-Gang)** scheduling was proposed to address the inter-core interference and schedulability issues. In RT-Gang, the tasks are grouped into a gang which is the smallest scheduling unit. It enforces a one-gang-at-a-time policy, which makes the interference smaller and manageable since only a predefined set (i.e., a gang) of tasks can occupy the processor at a time.

Moreover, it enables the use of well-studied unicore timing analysis and real-time schedulers to increase system schedulability. Even though RT-Gang's strict policies hinder exploiting the full capacity of hardware, we choose RT-Gang as it improves predictability and schedulability which are crucial in a real-time systems context.

However, it entails another problem: gang formation. It can be described as partitioning a DAG of tasks into multiple gangs, which decides the minimum end-to-end delay of a given DAG. Minimizing end-to-end delay is important because the shorter the delay, the more room for slowing down the processor until the end-to-end delay meets the deadline. The gang formation is an NP-hard combinatorial problem, and thus we developed a greedy heuristic to handle it.

Our experimental results show that our gang formation heuristic performs 32% better on average than the state-of-the-art heuristic. Our multi-mode optimization reduces the average total energy consumption by up to 54.9% in various real-world driving scenarios compared with the maximum energy consumption configuration and 30.3% compared with the conventional method using **Dynamic Power Management (DPM)**. 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 to optimize the energy consumption in multicore systems for autonomous driving, explicitly focusing on dynamic deadlines.

This study's contributions can be summarized as follows:

- We developed a greedy heuristic to obtain gang formation that could lead to better energy savings.
- We formulate an optimization problem for energy-efficient autonomous driving systems with time-varying dynamic deadlines equipped with a multicore processor 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.
- Our experiments use a realistic autonomous driving task set from industry and actual measurements from the target computing system in addition to randomly generated task sets for extensive analysis for comparison with conventional real-time energy optimization methods.

The rest of the article is organized as follows. Section 2 describes the background and our problem. Section 3 describes our gang formation heuristic. Section 4 presents our offline system optimization method. Section 5 explains the dynamic system reconfiguration approach. Section 6 provides the evaluation results. Section 7 presents related work. Finally, Section 8 concludes the article.

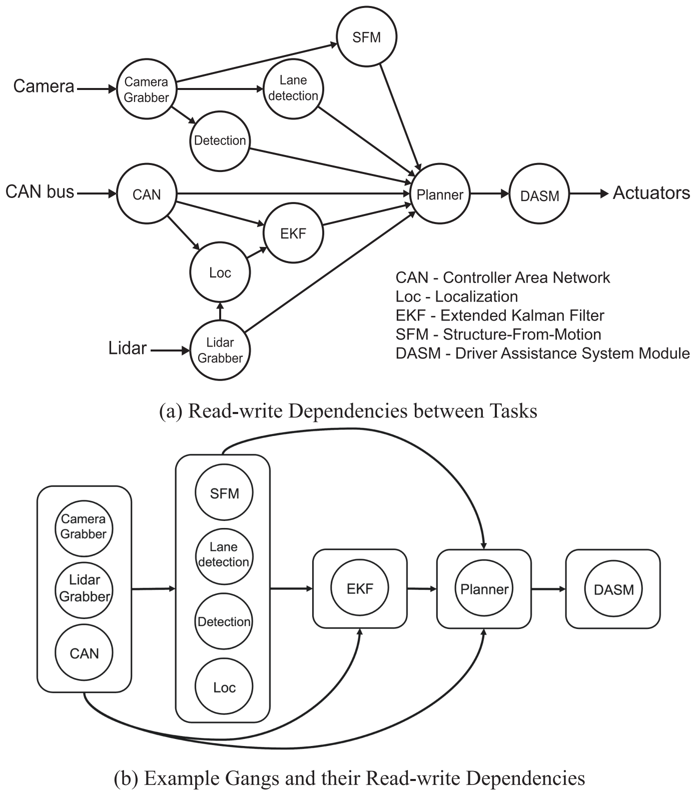
2 BACKGROUND AND PROBLEM DESCRIPTION

2.1 System Model

This study assumes a multicore processor with m homogeneous cores and VFI architecture. The CPU clock frequency is shared across the cores, which is expressed as a speed factor S in the range of

$$0 < S_{min} \leq S \leq 1, \tag{2}$$

where S_{min} denotes the minimum speed factor used for the CPU idle time, whereas the upper bound 1 (i.e., 100%) indicates the maximum processing speed. The system executes a set of n


Camera
Camera Grabber
SFM
Lane detection
Detection
CAN bus
CAN
Planner
DASM
Actuators
EKF
Loc
Lidar
Lidar Grabber
CAN - Controller Area Network
Loc - Localization
EKF - Extended Kalman Filter
SFM - Structure-From-Motion
DASM - Driver Assistance System Module

(a) Read-write Dependencies between Tasks

(b) Example Gangs and their Read-write Dependencies

Fig. 2. A DAG from Bosch as a reference autonomous driving system in the WATERS Industrial Challenge 2019 [39].

implicit-deadline periodic gangs of tasks

$$V = \{w_1, w_2, \ldots, w_n\}, \tag{3}$$

where each gang contains a minimum of one to m tasks such that every gang member will have at least one dedicated CPU core to avoid scheduling inside gangs [5]. The read-write dependencies between gangs are inherited from the gang members. For instance, Figure 2(a) shows a DAG with 10 tasks from sensors to actuators, where its nodes and directed edges represent the task set and dependencies, respectively. Then, Figure 2(b) shows an example of five gangs and their inherited dependencies. Tasks communicate with each other with asynchronous message passing. Due to the multi-rate nature, oversampling or undersampling can happen to communication buffers, where newly arrived data always overwrite existing ones. This task model has been commonly used in many studies for automotive systems [28, 33].

Depending on how tasks are grouped into gangs, so-called a gang formation, the end-to-end latency may vary because the dependency between gangs also changes. Finding a gang formation with the minimum end-to-end latency is known to be NP-hard and will be discussed in later sections.

Given a gang formation, each gang w_i is characterized by its period P_i, per-gang speed factor S_i, and WCET function E_i:

$$w_i = (P_i, S_i, E_i), \tag{4}$$

where a gang of tasks shares the same P_i and maintains S_i for E_i period of time. The speed of processor can only be changed at context switching between gangs to avoid energy and timing overhead of excessive frequency transitions [11]. This inter-task DVFS is typically known to have negligible latency overhead due to the small number of switchings. Therefore, we do not consider DVFS overhead in this work. For E_i, it takes an input S_i and outputs the maximum WCET among gang members at a given speed factor:

$$E_i(S_i) = \max\left(e_x^{mem} + \frac{e_x^{comp}}{S_i}\right), \forall x \in w_i, \tag{5}$$

where e_x^{mem} is a speed-independent portion of WCET of gang member x, and e_x^{comp} is the portion that scales inverse-linearly with S_i. Many studies assume the entire WCET to be inverse-linearly proportional to S_i [16] since it provides a safe upper bound. However, it can be largely inaccurate [12] because memory and I/O operations are less affected by S_i. Thus, we use the task model from the work of Aydin et al. [8], which considers speed-independent components into account. For notational convenience, we define *speed-independent ratio* r_i as follows for discussions in later sections.

$$r_i = e^{mem}/(e^{comp} + e^{mem}) \tag{6}$$

Among gang parameters in Equation (4), only P_i and S_i are design variables of our optimization problem. Thus, we define *system configuration* Π as

$$\Pi = ((P_1, S_1), \ldots, (P_n, S_n)), \tag{7}$$

which is a vector of tuples of each gang's period and speed factor. There could exist multiple system configurations under various deadline constraints, which will be further discussed in the next section.

Finally, we use the **Earliest Deadline First (EDF)** scheduling with the one-gang-at-a-time policy [5] such that the scheduling unit is a gang instead of a task. In other words, a DAG of tasks is transformed into another graph where each node is a gang by using gang formation algorithm. Then each gang is assigned its period and scheduled dynamically based on their implicit deadline with a preemptive EDF scheduler, occupying the entire multicore processor at a time. The L&L utilization bound [51] can be used to test the system schedulability whether all the gangs can be scheduled without violating their respective implicit deadline (i.e., before the next period starts). With the preceding gang WCET model, our L&L utilization bound is as follows:

$$U(\Pi) = \sum_{i=1}^{n} \frac{E_i(S_i)}{P_i} \leq 100\%, \tag{8}$$

where the sum of gang utilizations (the gang WCET over period) should not exceed the upper bound 100%. The upper bound can vary depending on the scheduler used.

2.2 Dynamic Deadlines

In our system model, when referring to deadlines, they always mean the *end-to-end deadlines* from sensors to actuators, not the implicit deadlines that are equal to periods. Thus, even when the system is schedulable, satisfying every gang's P_i, it does not mean that 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

$$\mathbb{P} = \{\delta_1, \delta_2, \ldots, \delta_{|\mathbb{P}|}\}, \tag{9}$$

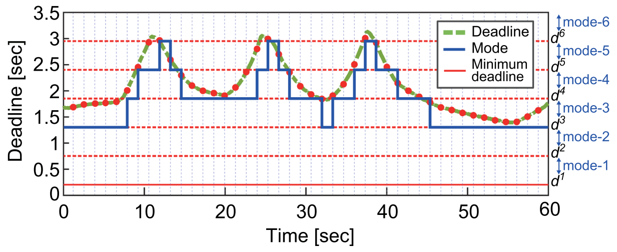


Fig. 3. Dynamic deadlines with discrete mode changes.

where each path δ_i denotes an ordered set of task (or gang) *indices* following the path. For example, in Figure 2(a), the original DAG has three (3×1) flows and eight paths. Similarly, for the corresponding gang formation in Figure 2(b), there exists a linear ordering of gangs that preserves task dependency for each path δ from the original DAG. 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 preceding notion is commonly referred to as *reaction time constraints* [6].

Figure 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 Equation (1)). 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

$$\{d^1, d^2, \ldots, d^m\}. \tag{10}$$

In Figure 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.

The mode's shortest deadline is the minimum *end-to-end latency* from sensors to actuators and is formally defined as follows. We borrow a widely used model from Davare et al. [28], which expresses the worst-case delay of a path δ as an accumulation of periods (P_is) and worst-case response times ($WCRT$s) of every task in δ, like the rightmost part in

$$D_\delta(\Pi) = \sum_{i \in \delta} 2P_i \approx \sum_{i \in \delta} (P_i + WCRT_i), \tag{11}$$

where $D_\delta(\Pi)$ denotes the approximated worst-case delay of a path δ assuming a system configuration Π. We approximate the original delay model to a linear form $\sum_{i \in \delta} 2P_i$ for simplicity. Among the per-gang delay components $2P_i$, one P_i is for the waiting time until the task reads the sensor data (*waiting delay*) and the other is for processing the data (*processing delay*). Then, given tasks and gang formation, the *end-to-end latency* is defined as the longest D_δ among paths in $\mathbb{P}$.

2.3 Gang Formation

Gang scheduling in real-time systems brings many benefits, such as mitigating inter-core interference, tighter WCET estimation, and increased system utilization so that the system can accept more tasks given end-to-end deadlines. However, it entails another NP-hard integer programming problem: the gang formation. This can be described as grouping tasks into multiple gangs while minimizing the end-to-end latency in Equation (11). We employ the state-of-the-art gang scheduling [5] that enforces one-gang-at-a-time policy and restricts co-scheduling inside a gang (i.e., the number of tasks in a gang does not exceed the number of cores) for minimizing inter-core interference. Depending on how gangs are formed, they may not fully utilize the multicore processor (e.g., the shorter task creates a core idle period until the longer task finishes), which leads to an increased delay and failure to meet the end-to-end deadline. More importantly, in our dynamic deadline context, the minimum end-to-end latency decided by a gang formation is closely related to power savings, as the gap between the delay and the deadline provides opportunities to slow down the CPU. Therefore, it is important to create a gang formation that minimizes the makespan, and a couple of heuristics have been proposed [3, 4].

2.4 Power Model

We use the following single-core power model from Bambagini et al. [11] and Bhuiyan et al. [15] as a base for the multicore model:

$$\mathcal{P}(S) = \mathcal{P}_s + \mathcal{P}_d(S) = \beta + \alpha S^\gamma, \tag{12}$$

where $\mathcal{P}_s$ is the static power and $\mathcal{P}_d(S)$ is the dynamic power parameterized by a speed factor S. The static power is expressed as β, independent of other parameters, whereas the dynamic power depends on S while α and $\gamma \in [2, 3]$ are CPU-dependent parameters. In this work, we do not utilize CPU sleep states to reduce the static power, so our focus is to minimize the average dynamic power by minimizing S as long as the dynamic deadlines are satisfied, although we also include and compare static power in the evaluation.

For that, the average dynamic power can be calculated as follows: for each gang w_i, its instantaneous dynamic power αS_i^γ is maintained during gang WCET (Equation (5)). Since the power pattern repeats by its period P_i, the average power consumed in a unit time while executing w_i can be calculated as

$$\mathcal{P}_i(P_i, S_i) = \alpha S_i^\gamma \times \frac{E_i(S_i)}{P_i}, \tag{13}$$

which is a function of P_i and S_i. Then, by summing up n such average powers and the idle-time CPU power at S_{min}, the average dynamic power of core k is given as a function of a system configuration Π as

$$\mathcal{P}^k(\Pi) = \alpha \sum_{i=1}^{n} S_i^\gamma \frac{E_i(S_i)}{P_i} + \alpha S_{min}^\gamma \left(1 - \sum_{i=1}^{n} \frac{E_i(S_i)}{P_i}\right). \tag{14}$$

For the homogeneous multicore processors, the total average power consumption can be described as the sum of individual core's power consumption as in the work of Basmadjian and de Meer [14]:

$$\mathcal{P}(\Pi) = \sum_{k}^{m} \mathcal{P}^k(\Pi), \tag{15}$$

where m is the number of cores in the homogeneous multicore processor.

2.5 Problem Description

Given a DAG G of n periodic tasks and m discrete deadline constraints, our problem is decomposed into three parts: (i) gang formation, (ii) period and speed factor optimization, and (iii) safe mode change protocol. First, we obtain a gang formation $W = \{w_1, w_2, \ldots, w_k\}$ that can minimize the end-to-end latency of the given DAG considering task WCET and dependency. Second, given a

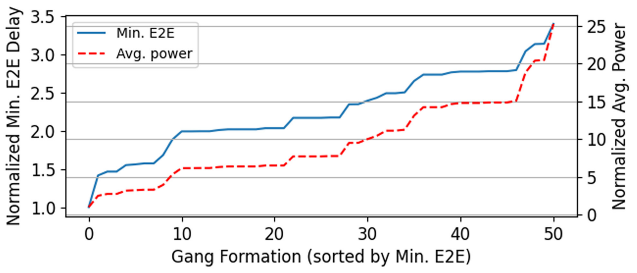


Fig. 4. Gang formation problem space. A shorter end-to-end latency results in a lower power consumption.

gang formation, we find the gang periods and speed factors that minimize average power consumption while satisfying the end-to-end deadline constraint for each mode. The solution can be described by following two matrices:

$$\begin{pmatrix} P_1^1 & P_2^1 & \cdots & P_k^1 \\ \vdots & \vdots & \ddots & \vdots \\ P_1^m & P_2^m & \cdots & P_k^m \end{pmatrix} \text{ and } \begin{pmatrix} S_1^1 & S_2^1 & \cdots & S_k^1 \\ \vdots & \vdots & \ddots & \vdots \\ S_1^m & S_2^m & \cdots & S_k^m \end{pmatrix}, \tag{16}$$

where P_i^j and S_i^j represent gang i's optimal period and speed factor at the j-th mode, respectively. In other words, each row represents mode j with system configuration Π_j that guarantees the end-to-end latency shorter than d_j in Equation (10). Finally, 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 is proposed in Section 5.

3 GANG FORMATION HEURISTIC

Given a DAG of n tasks, their WCETs, and homogeneous m cores, we want to group tasks into an arbitrary number of gangs while minimizing the end-to-end latency defined in Equation (11). The reason for minimizing end-to-end latency here is that it is closely related to minimizing power. In Figure 4, we exhaustively searched gang formation space for an example five-task DAG. All possible gang formations are sorted by their minimum end-to-end latency and numbered sequentially over the x-axis. The average power consumption on the right y-axis is calculated using the period optimization in the next section, under the same end-to-end deadline for all gang formations. We discovered that the power consumption increases monotonically as the minimum end-to-end latency increases. This is because the bigger the gap between the minimum end-to-end latency and the deadline, the more room for the CPU to slow down, resulting in smaller power consumption.

Intuitively, to get better gang formation, it is desirable to pack gangs as densely as possible to fully utilize m cores while trying to avoid grouping dependent tasks to the same gang. If tasks in dependency constraints are in the same gang, the gang has to be repeated to propagate data to the consumer task which was not possible in the previous execution because it is synchronized to start with the provider task by gang scheduling. This adds an additional delay to the overall end-to-end latency.

The gang formation proves an NP-hard integer programming problem [3] like bin packing. Therefore, Ali et al. [4] proposed the **Virtual Gang Heuristic (VGH)**. In VGH, a DAG consists of tasks on m homogeneous cores and is partitioned into multiple gangs minimizing the completion time of all the gangs. It introduces the notion of family to avoid grouping dependent tasks. A family of tasks τ_i consists of all tasks that are connected with τ_i's ancestor or descendant. VGH strongly restricts grouping tasks in a family relationship to avoid the repetition of gang execution.

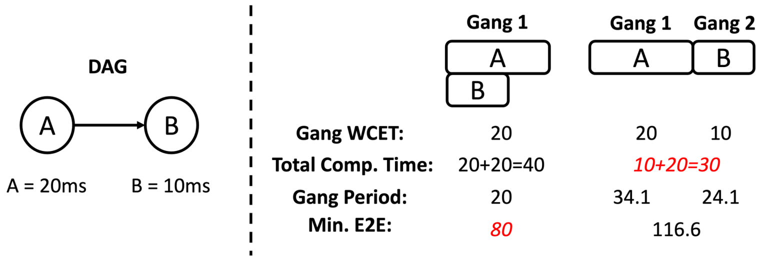


Fig. 5. Gang formation example with different objectives (total completion time vs. minimum end-to-end latency).

However, VGH does not fit well for our problem because it has a different objective. It minimizes the total completion time of gangs that is equal to the sum of gang WCETs along the paths. Yet our objective is to minimize end-to-end latency considering the worst data propagation pattern, which is analyzed in Equation (11). A simple two-task example is depicted in Figure 5. In the example, tasks A and B are in a dependency constraint. For VGH, it makes sense to avoid packing A and B together in the same gang because it would increase the total completion time. However, the gang periods tend to increase significantly as we add more gangs because of the utilization bound in Equation (8). Basically, the utilization bound implies that the period should be large enough to have time for the execution of other gangs. In this case, A's period (34.1 ms) is significantly greater than its WCET (20 ms) to give task B (10 ms) a chance to execute. Therefore, for the minimum end-to-end latency that is based on gang periods, it is beneficial to pack A and B together despite the dependency relationship.

Moreover, VGH does not consider WCET changes over processor speed, especially with the speed-independent ratio r_i defined in Equation (6). Specifically, the gang WCET is defined as the longest task's WCET in the gang, and if its r_i is high, the dominant task can be changed to another one as depicted in Figure 6. As the speed factor increases, task 1's WCET shrinks slower than task 2's and task 1 becomes the dominant task in the gang. In worse cases, the gap between the grouped tasks' WCETs becomes big enough to significantly decrease the multicore utilization, which can be deemed as the quality degradation of gang formation over S_i changes. This would not happen without r_i, as the tasks will scale at the same ratio. At certain S_i, the gang WCET might change significantly and the gang formation obtained at $S_i = 1$ may not perform well. To address this issue, we try to find the *base speed factor* (S_{base}) and obtain a gang formation from the scaled task WCETs that minimizes the degradation of the gang formation quality over S_i changes. For example, the gang formation with $S_{base} = 0.5$ might degrade less than the one with $S_{base} = 1.0$ as the possible S_i change is smaller (0.5-s_{min} vs. 1.0-s_{min}).

As a result, VGH can be improved in three ways for our problem. First, the strict family concept should be alleviated, as there are advantageous cases when tasks in dependency constraints are in the same gang, as we have seen in the previous example. Second, an estimation of gang periods must be taken into account, as our objective is calculated based on periods, not WCETs. Third, the gang WCET changes over S_i should be considered, as the heuristic solution may perform significantly worse at different S_i.

We developed a greedy heuristic, **Dynamic Deadline Gang Heuristic (DDGH)**, as shown in Algorithm 1. Line 1 scales the task WCETs with the base speed factor (S_{base}). Line 2 sorts tasks by its WCET in descending order. Sorting tasks helps to group tasks with similar WCETs, trying to fully utilize all of the cores during gang execution. Line 3 initializes an empty gang formation. As a greedy approach, a single task is picked (line 6) and then assigned to a gang (line 12) at a

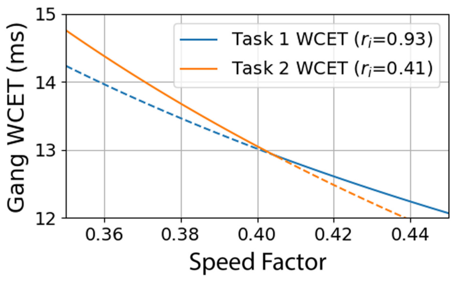


Fig. 6. An example gang WCET curve with $r_i = e_i^{mem}/e_i$. Solid lines represent the dominant task at the time.

ALGORITHM 1: Dynamic Deadline Gang Heuristic

```
Require: Task WCETs ($e$), speed-independent ratio ($r$), base speed factor ($S_{base}$), paths in the DAG ($\mathbb{P}$)
Ensure: Gang formation ($gangs$)
 1: $new\_wcet = scale\_wcet\_at\_sf(e, r, S_{base})$
 2: $q = sort\_task\_by\_wcet(new\_wcet)$
 3: $gangs = ()$
 4: while not $q.empty()$ do
 5:     $candidates = ()$
 6:     $task = q.pop()$
 7:     for $g \in \{gangs + \emptyset\}$ do
 8:         $new\_gangs = add\_gang\_member(gangs, g, task)$
 9:         $delay = calc\_delay\_proxy(new\_gang, e, \mathbb{P})$
10:         $candidates.push(delay)$
11:     $best\_g = min(candidates)$
12:     $gangs = add\_gang\_member(gangs, best\_g, task)$
    return $gangs$
```

time. Lines 7 through 10 create a temporary gang formation (*new_gangs*) to add the current task to an existing gang for evaluation. The iteration with $g = \emptyset$ means creating a new gang with the current task. Finally, line 11 decides the best gang (*best_g*) for the current task and updates the gang formation (*gangs*). The process is repeated until the queue (q) is empty.

For evaluating temporary gang formations, *calc_delay_proxy* function in Algorithm 2 is used to provide a proxy of end-to-end latency. It is a proxy because it is difficult to estimate the exact end-to-end latency, which is the optimization result in the next section. Instead, we exploit the knowledge of optimization solution form to just estimate the relative end-to-end latencies between different gang formations. Lines 2 and 3 calculate gang WCETs (E) for a given gang formation. Lines 5 through 9 iterate over each path in the DAG and sum up our metric in line 8. The metric is the core part of the heuristic and consists of gang WCET and the sum of gang WCETs. With both terms multiplied, it acts as a proxy for the gang period, which is a basic component in end-to-end latency. The first term, gang WCET, contributes to minimizing the individual gang's WCET. The period is typically proportional to WCET, so the smaller the gang WCET, the smaller the period. The second term, the sum of gang WCETs, is another critical part to estimate the period. When a new gang is added, one of the gang periods must be greater than the sum of gang WCETs, as there should exist enough time for other gangs to execute. This is implied in the utilization bound in Equation (8) ($U = \sum E_i/P_i < 1$). It also increases the other gang's period, as we have seen with the example in Figure 5, so the heuristic is discouraged to create a new gang by this term. Therefore, DDGH does not strictly prohibit the grouping of tasks in precedence relationships. Instead, it uses

ALGORITHM 2: Evaluate Temporary Gang Formations

```
Require: Gang formation ($gangs$), task WCETs ($e$), paths in the DAG ($\mathbb{P}$)
Ensure: Proxy of end-to-end latency
function CALC_DELAY_PROXY($gangs$, $e$, $\mathbb{P}$)
    for $gang \in gangs$ do
        $E(gang) = get\_gang\_wcet(gangs)$
    $max\_delay = 0$
    for $\delta \in \mathbb{P}$ do
        $max\_delay = 0$
        for $task \in \delta$ do
            $delay+ = E(task\_to\_gang(task)) * sum(E)$
        if $max\_delay < delay$ then
            $max\_delay = delay$
    return $max\_delay$
```

a proxy delay to assign tasks flexibly and discourages creating a new gang, so gangs are packed as densely as possible.

4 PERIOD AND SPEED FACTOR OPTIMIZATION

This section formulates and solves the offline multi-mode system optimization problem assuming that gang formation is given and fixed. We begin by finding the optimal configuration for a single mode without considering safety constraints in the transient phase between the modes (Section 4.1). Then, we extend the optimization method such that it can go back and forth between the modes without violating safety constraints (Section 4.2) followed by the optimization solver we used (Section 4.3). Finally, a realistic condition of discrete CPU frequencies is discussed (Section 4.4).

4.1 Single-Mode Formulation

This section explains how we can formulate a single-mode optimization as a baseline for multi-mode system optimization. As the objective function, the average power in Equation (14) is used, without the constants α and β, where Π denotes two sets of decision variables P_is and S_is 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 Equation (8), and (ii) deadline constraint of which we already devised an end-to-end latency analysis model in Equation (11).

Then, our single-mode formulation is given as follows:

$$\begin{aligned} \underset{\Pi}{\text{minimize}} \quad & \mathcal{P}(\Pi) = \sum_{i=1}^{n} \frac{S_i^{\gamma} E_i(S_i)}{P_i} + S_{min}^{\gamma}\left(1 - \sum_{i=1}^{n} \frac{E_i(S_i)}{P_i}\right) \\ \text{subject to} \quad & U(\Pi) = \sum_{i=1}^{n} \frac{E_i(S_i)}{P_i} \leq 1 \\ & D_{\delta}(\Pi) = \sum_{i \in \delta} 2P_i \leq d \quad (\forall \delta \in \mathbb{P}). \end{aligned} \tag{17}$$

4.2 Multi-Mode Formulation Considering Mode Changes

Naively, the method in Section 4.1 can be repeatedly used to find every row of the multi-mode solution matrices in Equation (16). However, we cannot directly use this approach for a multi-mode system since it does not guarantee a safe transition between modes. Specifically, when old- and new-mode gangs coexist during a mode change, schedulability violations can occur even if each mode is schedulable in isolation [21]. Thus, we add a new constraint called *utilization invariability*, meaning that every gang's utilization (i.e., $u_i^j = E_i(S_i^j)/p_i^j$) is invariant across modes as

$$\frac{E_i(S_i^1)}{P_i^1} = \frac{E_i(S_i^2)}{P_i^2} = \cdots = \frac{E_i(S_i^m)}{P_i^m} = u_i^* \ (\forall i = 1, 2, \ldots, k), \tag{18}$$

where u_i^* denotes identical utilization for w_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 [1]. Now, we use u_i^*s as our decision variable replacing P_i^js with $E_i(S_i^j)/u_i^*$. Then our multi-mode optimization can be formulated as follows:

$$\begin{aligned} \underset{\hat{\Pi}}{\text{minimize}} \quad & P(\hat{\Pi}) = \sum_{j=1}^{m}\sum_{i=1}^{n}(S_i^j)^{\gamma} u_i^* + S_{min}^{\gamma}\left(m - \sum_{j=1}^{m}\sum_{i=1}^{n} u_i^*\right) \\ \text{subject to} \quad & U(\hat{\Pi}) = \sum_{i=1}^{n} u_i^* \leq 1, \\ & D_{\delta}^j(\hat{\Pi}) = \sum_{i\in\delta'} \frac{2E_i(S_i^j)}{u_i^*} \leq d^j \quad (\forall j \in [1, m], \forall \delta' \in \mathbb{P}'), \end{aligned} \tag{19}$$

where $\hat{\Pi}$ denotes the newly defined multi-mode system configuration with P_i^js and u_i^*s. The objective function is the sum of average power in each mode, after eliminating the α and β from Equation (14) for the notational simplicity. The first constraint is the system schedulability now expressed by u_i^*s. The second constraint is for the dynamic deadlines across m modes.

Note that the average power of a mode from multi-mode solution could be worse than the one from single-mode solution. To achieve a mode's optimal average power, each gang should freely choose its period and speed factor. However, if we fix the utilization value across modes, the period and speed factor are not independent anymore but constrained by the relationship $u_i^* = E_i(S_i^j)/p_i^j$. This adverse effect of utilization invariability is further discussed in the experiment section.

4.3 GP-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 [18]. To use GP, the objective function and inequality constraints should be constructed by the special form *posynomial*, as in $f(x) = \sum_{k=1}^{K} c_k x_1^{a_{1k}} x_2^{a_{2k}} \cdots x_n^{a_{nk}}$, with decision variables x_is, non-negative coefficients c_ks, and real-valued exponents $\{a_{11}, \ldots, a_{nK}\}$. Our objective functions and constraints are in posynomial forms except for the idle-time CPU power terms in the rightmost part of $\mathcal{P}(\Pi)$ in Equation (17) and $\mathcal{P}(\hat{\Pi})$ in Equation (19).

However, the idle power terms 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 gang w_i and decrease S_i, thus increasing E_i, until U reaches 100%, the average power of w_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_is are bounded by S_{min}. We are not considering such extreme cases in this work.

4.4 Extension to Discrete CPU Frequency Levels

Our optimization method yields speed factors in the continuous range between S_{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 [17, 45] 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 [61] and possible transient faults [26, 67].

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 utilization invariability constraint. Thus, even though the approximated system will consume more energy than the continuous one, it safely ensures schedulability and end-to-end latency constraints during mode changes.

5 SAFE MODE CHANGE FOR SYSTEM RECONFIGURATION

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

- *Periodicities*: Every gang period before and after the mode change should be guaranteed, which can be satisfied by the utilization invariability constraint introduced in Section 4.2.
- *Deadlines*: Unfortunately, however, the preceding periodicities do not guarantee dynamic deadlines, which span across multiple gangs 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 already guaranteed per-gang periodicity.

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

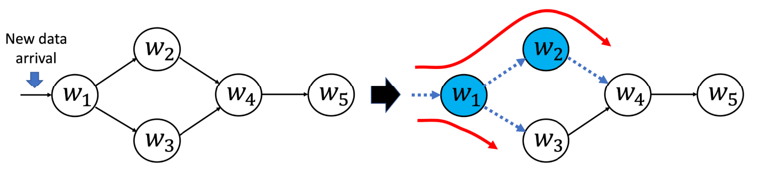
$$d^{old} \rightarrow d^{new}. \tag{20}$$

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

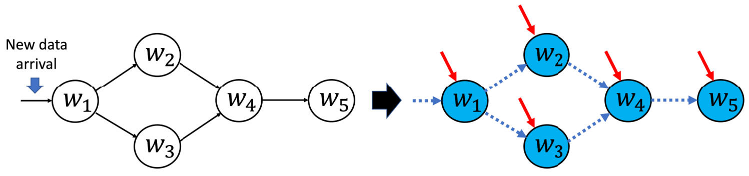
$$((P_1^{old}, S_1^{old}), \ldots, (P_n^{old}, S_n^{old})) \rightarrow ((P_1^{new}, S_1^{new}), \ldots, (P_n^{new}, S_n^{new})) \tag{21}$$

in its expansion form. The system mode change is triggered by a sensor data arrival at time t_0 with its dynamic deadline falling above or below the old range. In terms of individual gangs, their mode changes only happen at period boundaries. In other words, when a gang period has ended and the new one is about to start, the scheduler adjusts its internal structure to the new mode's period and speed factor for that gang. This is because if we change the mode for a gang when its period has not finished yet, the instantaneous utilization (E_i/P_i) for the gang disturbs the overall schedulability (Equation (8)), which defeats the purpose of utilization invariability in our optimization constraints and may lead to a deadline violation. Then, we consider the following mode change methods, as depicted in Figure 7:

- *ALAP* (*As Late As Possible*) individually triggers per-gang 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.



(a) As Late As Possible (ALAP)



(b) As Early As Possible (AEAP)

Fig. 7. ALAP: Tasks gradually change modes (white → blue) as the new data progress along paths (curved red arrows) possibly at different speeds. AEAP: The new sensor data arrival immediately triggers (straight red arrows) every task's mode change.

- *AEAP* (*As Early As Possible*) immediately triggers every task regardless of the new data's progress. The mode change completes by $t_0 + \max_i(P_i{}^{old})$ in the worst case when the longest period of Π^{old} began right before t_0.

We deal with the following two cases: (i) relaxing and (ii) shrinking deadlines with the preceding methods, respectively.

Case (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^{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 Figure 7(a), τ_4 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 $w_1 \rightarrow w_2 \rightarrow w_4 \rightarrow w_5$ 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:

$$D^{new}(\Pi^{old} \rightarrow \Pi^{new}) = \max_{\forall \delta \in \mathbb{P}'} \left(\sum_{i \in \delta} 2 \cdot \max(P_i{}^{old}, P_i{}^{new}) \right) \leq \max_{\forall \delta \in \mathbb{P}'} \left(\sum_{i \in \delta} 2 P_i{}^{new} \right) = d^{new}, \tag{22}$$

which is less than d^{new} since $\forall i : P_i{}^{old} \leq P_i{}^{new}$ that is true when relaxing deadlines. This is because when the gang utilization (u_i^*) is fixed across modes, if S_i decreases in Equation (8), P_i should increase to offset the change. Therefore, P_i increases monotonically to decrease S_i^{γ} in Equation (19), reducing the average power in the next longer deadline mode.

Case (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^{old} by the same rationale in Equation (22) since $\forall i$: $P_i{}^{old} \geq P_i{}^{new}$ when shrinking deadlines. However, regarding the new sensor data, it can suffer extra delays if any gang'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 gang instance. Algorithm 3 calculates the worst-case delay considering such negative effects for each path δ, which is an ordered set of gang indices in each path. Among the calculated delays, we can find the longest. The algorithm gradually accumulates delays by gangs in δ. Line 1 indicates that

Table 1. Workload Information at Maximum Speed ($s_i = 1$)

Task	Cam. Grabber	Lidar Grabber	CAN	SFM*	Lane Det.*	Obj Det.*	Loc.*	EKF*	Planner	DASM
WCET(ms)	0.25	25.7	0.6	30.6	27.1	294.8	175.5	1.6	21.0	1.9
$\boldsymbol{r_i}$	0.5	0.5	0.5	0.15	0.45	0.29	0.20	4e-9	1e-15	1e-15

*From the Chauffeur benchmark suite [53]. WCETs of unmarked applications are derived from the work of Krawczyk et al. [48].

ALGORITHM 3: Finding the Worst-Case Delay for AEAP

```
Require: $\{(P_1^{old}, \ldots, P_n^{old}), (P_1^{new}, \ldots, P_n^{new}), \delta\}$
Ensure: The worst-case delay of new sensor data for $\delta$
1: $D \leftarrow P_{\delta[1]}^{old} + P_{\delta[1]}^{new}$        ▷ $\delta[1]$ denotes its first element
2:
3: for $i \in \delta \setminus \{\delta[1]\}$ do
4:     if $D + 2P_i^{new} > P_i^{old} + P_i^{new}$ then
5:         $D \leftarrow D + 2P_i^{new}$
6:     else
7:         $D \leftarrow P_i^{old} + P_i^{new}$
   return $D$
```

it is unavoidable for the new sensor data to be *waited* by the old period at the first gang. Then we have two cases for the remaining gangs: (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 $2P_i^{new}$. In the latter, the persisting old (long) period P_i^{old} hides away the accumulated delay up to then, resetting it to $P_i^{old} + P_i^{new}$.

By the preceding analyses, we claim that when relaxing deadlines with ALAP, there is no end-to-end 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. For example, instead of using direct mapping from velocities to modes, the system can change its mode a little earlier at a lower velocity to offset the extra delays.

6 EXPERIMENTS

6.1 Experimental Setup

Workload. When we need an exemplar DAG, we use the one in Figure 2 with 10 tasks, where their WCETs and the speed-independent ratio r_is are listed in Table 1 [48] and their gang formation in Table 2. Unfortunately, WATERS industrial challenge applications are IP protected, so we were not able to run them and obtain r_is. We borrowed available applications from the Chauffeur autonomous driving benchmark [53] and measured their WCETs and r_is on the Nvidia Jetson TX2 platform (A57 cores). We did a frequency sweep to get r_i and applied curve fitting. All r_is have R-squared greater than 0.99 in their curve fitting. The WCETs of unmarked applications in Table 1 are derived from actual measurements on the same Jetson platform from Krawczyk et al. [48], and we made an educated guess about their r_is.

DAG Generation. We use the GGen random graph generator [24] to generate multi-source and -sink graphs in a layer-by-layer method: 500 DAGs for each input size n = 5, 10, 20 and edge probability p = 0.5, 0.25, and 0.125. All generated DAGs are non-isomorphic to each other (i.e., structurally different). WCETs are generated with a uniform distribution, and for each WCET, three sets of r_is are generated from different ranges: low [0.0, 0.5], high [0.5, 1.0], and mixed [0.0, 1.0].

Table 2. Gang Formation of WATERS Workloads on Quad-Core CPU Using the Heuristic from Ali et al. [4]

Gang	1	2	3	4	5
Tasks	SFM, Lane Det., Obj Det., Loc.	Cam. Grabber, Lidar Grabber, CAN	Planner	DASM	EKF
WCET at $\forall S_i = 1$	294.8	25.7	21.0	1.9	1.6
r_i **at** $\forall S_i = 1$	0.29	0.5	1e-15	1e-15	4e-9

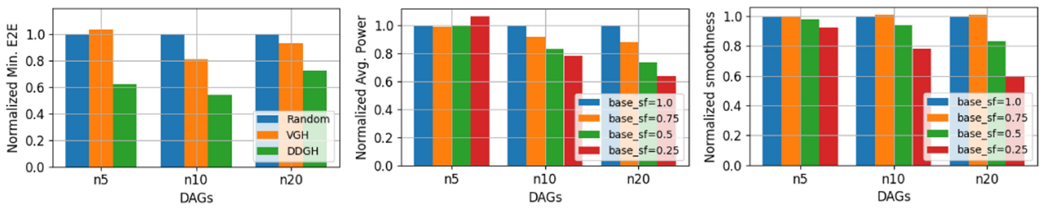
Power Model. We empirically found the power parameters for Equation (12), as $\alpha = 842.04$, $\beta = 232.81$, and $\gamma = 2.64$, on the same hardware platform in the work of Krawczyk et al. [48].
Discrete Frequency Levels. We use 12 evenly spaced frequencies between 345 MHz and 2 GHz from the same hardware platform [48] for real-world driving data simulation.
Driving Scenarios. We use real-world driving scenarios from the comma.ai driving dataset [23], where we picked ten 60-second driving logs with their velocity from 0 km/hour to 114 km/hour as depicted in Figure 13.
Dynamic Deadlines. Each DAG has its shortest and longest end-to-end latency on the target platform when $\forall S_i = 1$ and $\forall S_i = s_{min}$, respectively. For the fair comparison between DAGs, we converted velocities into deadlines for each DAG using Equation (1) by picking values for λ when $a^{max} = 2.5$ ms^{-2} such that the shortest latency of the DAG becomes the shortest deadline at the highest velocity (114 km/hour) in our scenarios, whereas the longest latency of the DAG which is bounded by $s_{min} = 0.17$ becomes the longest deadline. Then, the number of modes is chosen arbitrarily to 10, partitioning the range with equal length.
Optimization. For the GP solver, we use the MATLAB CVX [37] convex programming package. On a laptop with a quad-core Intel Core i7@2.6-GHz CPU, it took 3.07, 7.97, and 23.11 seconds for $\{n = 5, p = 0.5\}$, $\{n = 10, p = 0.25\}$, and $\{n = 20, p = 0.125\}$ task sets, respectively.
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 latencies, and energy consumption. Our simulator takes 0.1-ms timesteps in each iteration and in each step the EDF scheduler routine is invoked. Gangs are submitted to the scheduler queue at the start of the simulation, and they resubmit themselves at the end of their period. The EDF scheduler (1) decreases the current gang's remaining WCET, (2) checks if the current gang's WCET is equal to or below 0 and schedules a new gang from the queue if needed, (3) sets the speed factor and period for the new gang, and finally (4) maintains a queue in the earliest deadline first order.

6.2 Evaluation

We first present gang formation results, by which we decide gang formations for each DAG for the rest of the evaluation. For the period and speed factor optimization, we compare power savings in individual modes with our multi-mode optimization method. Then, we vary the parameters, such as the number of tasks, the number of edges, and the speed-independent ratio r_i to evaluate their effect on our power optimization. Our multi-mode solution is then evaluated with real-world driving scenarios in a simulator with our safe mode change protocol. Finally, the efficacy of our safe mode change protocol is evaluated with an end-to-end deadline violation example. The following three methods are compared in the evaluation:

- ***Baseline***: $\forall i : S_i = 1$.
- ***DPM (conventional)***: $\forall i : S_i = 1$ while executing a gang, and otherwise the CPU is in sleep state.
- ***Multi-mode (OURS)*:** The method in Section 4.2.

One of the traditional methods of DVFS-based power optimization in real-time systems is to set a static deadline, assumed to be the shortest deadline that could possibly occur, and find the critical speed to save dynamic power consumption. The power savings of such a method largely depends



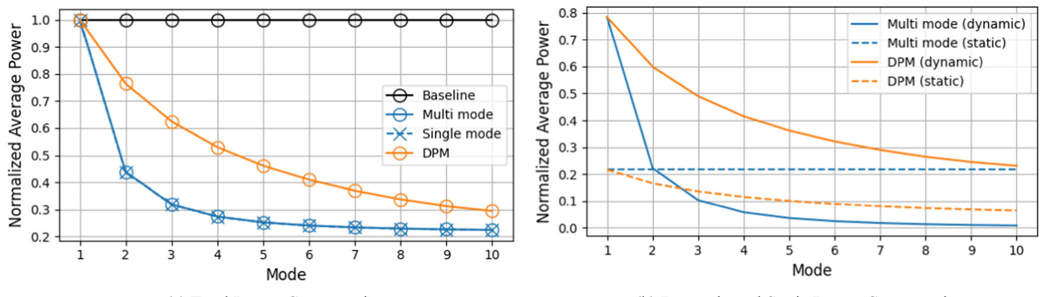
(a) Minimum End-to-end Latency (b) Average Mode Dynamic Power (c) Gang WCET Curve Smoothness

Fig. 8. Gang formation heuristic comparison and base speed factor variation experiments.

on how far away the static deadline is from the minimum deadline that the given system is able to handle and thus the result could be subjective. Therefore, we set the minimum deadline as the static deadline, labeled as *Baseline*, representing the maximum possible power savings from the traditional static deadline optimization methods. *DPM* [58] is another conventional method that we are comparing to, which could reduce both dynamic and static power consumption. In general, it is believed to have a better power efficiency than DVFS in periodic real-time systems [10]. Although there are many power-efficient scheduling algorithms for DPM, they do not consider safe mode change and thus cannot be used directly in our setting. Therefore, we use our safety-guaranteed solution and mode change for DPM experiments. Specifically, they have the same gang periods as our multi-mode solution such that end-to-end deadlines are guaranteed. In each mode, a gang will be given the time for execution based on Equation (5). However, the gang will be executed at the full speed, and the remaining slack is used for sleeping to cut off the static power consumption. For simplicity, we do not consider the overhead of changing CPU sleep states.

6.2.1 Gang Formation Heuristics. In Figure 8(a), we evaluate three gang formation heuristics: random, VGH, and our DDGH with 500 DAGs for each task set (n = 5, 10, 20) and mixed values of r_is. The results are normalized to random formation results for each DAG and then averaged. The random method randomly picks a task and assigns it to a gang, including both existing and new gangs, with an equal probability. VGH is the current state-of-the-art method, but it aims to minimize the total completion time, exhibiting slight improvement for end-to-end latency compared to the random formation. It is worse than random when a DAG is small and simple as in the $n = 5$ task set. When the length of paths in a DAG is not too long, it is advantageous to pack dependent tasks together as we have seen in the example (Figure 5), which is not possible with VGH because of the family policy. However, as the paths get longer as the number of tasks increases in $n = 10$ task set, VGH's family policy takes a positive effect and performs better than the random method. With the $n = 20$ task set, even though the gang formation problem becomes significantly difficult due to combinatorial explosion, our DDGH still performs better than VGH because of its flexibility in grouping dependent tasks and better end-to-end latency estimation. Overall, DDGH performs 32% better on average compared to VGH in all task sets.

Next, we evaluated the effect of the base speed factor (S_{base}) in Algorithm 2. In Figure 8(b), we sampled four values of S_{base} from 0.25 to 1.0 to see the trend. For each S_{base}, we apply multi-mode optimization to the gang formation to obtain power consumption in each mode. The number of modes was set to 10, each of which has a corresponding end-to-end deadline for which all the S_{base} gang formations were optimized equally. Since it requires a holistic evaluation across modes, the average mode power consumption is used for evaluation. Our initial hypothesis was that the sweet spot would exist around $S_{base} = 0.5$ so that the gang WCET curve change is minimized toward both ends of the mode range: the shortest and the longest modes where S_i changes are maximum. However, we discovered that the lower S_{base} is, the less degradation it gets by S_i changes, resulting



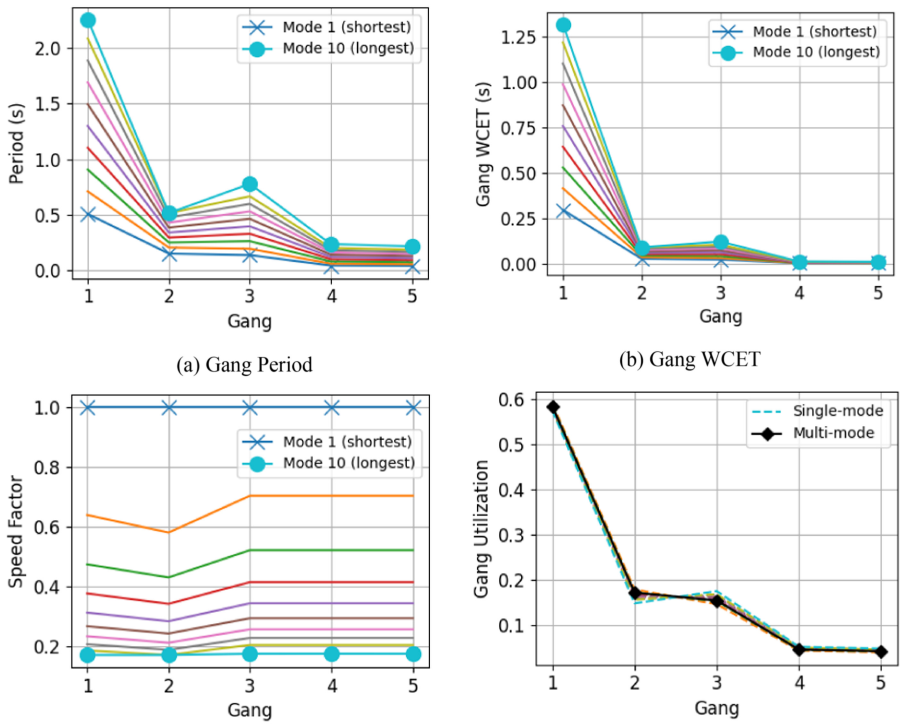
(a) Total Power Consumption (b) Dynamic and Static Power Consumption

Fig. 9. Power optimization results including both dynamic and static power. Mode 1 has the shortest deadline.

in a smaller average mode power. The reason is that when task WCETs are scaled to a lower S_{base} value, DDGH identifies the tasks whose WCETs are significantly scaled up and tends to group them together. In this way, even if there is a change in S_i, the overtaking of a dominant task happens less frequently and the quality of gang formation is maintained stable over S_i changes. We measured the smoothness of gang WCET curve in S_{base} solutions for each task set. The smoothness is defined as the e_{comp} differences between the two tasks when dominant task switching happens as e_{comp} is the only component affected by S_i. The larger the difference is, the greater the impact on gang formation quality and the resulting end-to-end latency, so a smaller value is desired for smoothness. For the $n = 5$ task set, because the DAGs are simple, gang WCET changes rarely occur, so the smoothness did not change much and our S_{base} method did not have a major impact. However, for larger task sets, smoothness declines steeply, exhibiting a similar trend with the average mode power consumption. As a result, for $n = 10$ and $n = 20$ with $S_{base} = 0.25$, the power consumption is reduced by 21.8% and 36.1%, respectively, compared to $S_{base} = 1.0$ gang formation. Following this trend, S_{base} is set to S_{min} for the rest of the experiments that involve DDGH.

6.2.2 Multi-Mode Average Power Optimization. Figure 9(a) shows normalized average power in each mode, averaged across 500 randomly generated 10-task DAGs with mixed values of r_is. The modes are numbered in order from the shortest to the longest deadline, equally partitioning the range from minimum to maximum end-to-end latency of each DAG. For the comparison between single-mode and multi-mode optimization, the only difference is the utilization invariability constraint. However, as it can be seen in the figure that the two curves are almost perfectly overlapped, showing its effect on power consumption is negligible. Even if the freedom of gang utilization variable is limited in multi-mode optimization, the other decision variable P_i is flexibly adjusted to lower S_i as much as possible, still reaching the near-optimal power consumption. In the first mode with the shortest deadline, both DPM and our multi-mode approach show the maximum power consumption, as there is no slack for slowing down nor shutting off the CPU cores. As the deadline gets longer, the slacks also become longer. To exploit them, DPM inactivates the CPU cores to save static power, as can be seen in Figure 9(b). Since the ratio of execution burst at full speed to the size of slack gets smaller, the average dynamic power of DPM also decreases, but not as fast as our multi-mode method. Even though the CPU cores are always on consuming static power, the dynamic power reduction is much greater than DPM resulting in overall smaller average power of 22.4% against the maximum power consumption compared to 30.8% of DPM in the longest deadline mode.

6.2.3 Gang Parameter Details in Each Mode. Figure 10 depicts the exemplar DAG's gang parameters in each mode after applying our multi-mode optimization. Only the first and the last



(a) Gang Period (b) Gang WCET

(c) Gang Speed Factor (d) Gang Utilization

Fig. 10. Gang parameters in each mode: gang period, WCET, speed factor, and utilization.

mode are marked for readability. Basically, each of these parameters tends to scale linearly from the shortest mode. The gang period (Figure 10(a)) and WCET (Figure 10(b)) are positively correlated, as it can be seen starting from the bottom of the figures that they increase similarly as the mode goes higher. Note that for gangs 2 and 3, their WCETs are similar, but gang 3 is more sensitive to the speed factor due to its low speed-independent ratio r_i. Eventually, in the longest deadline mode, gang 3 has a longer WCET and period than gang 2. For speed factors (Figure 10(c)), all gangs try to maintain a similar value. This is because the exponent (γ) of the speed factor in power Equation (14) is typically greater than 2, so the power is more sensitive to an increase in speed factor than a decrease. Therefore, if all WCETs scale at the same rate with S_i (i.e., all r_is are the same across tasks), the solver cannot favor a specific S_i, which may lead to an increase in another gang's S_i and worse overall power savings. As a result, it tries to find a uniform value for all S_is. In Figure 10(c), however, the speed factors of gangs 1 and 2 are lower than others. Given the utilization and period, these gangs with higher r_i can afford lower S_i due to the low sensitivity of WCET to S_i. Finally, the gang utilization (Figure 10(d)) is maintained the same across modes in the multi-mode solution. Note that even though the optimal gang utilization from single-mode solutions depicted in dashed lines does not deviate too far from the multi-mode solution, they could possibly violate the utilization bound during mode change. However, the effect of the deviation on power savings is negligible, as can be seen in Figure 9.

6.2.4 Effect of the Number of Edges and Tasks. When generating random DAGs, we tried to have a similar shape to the exemplar DAG, meaning that the ratio of the number of edges to

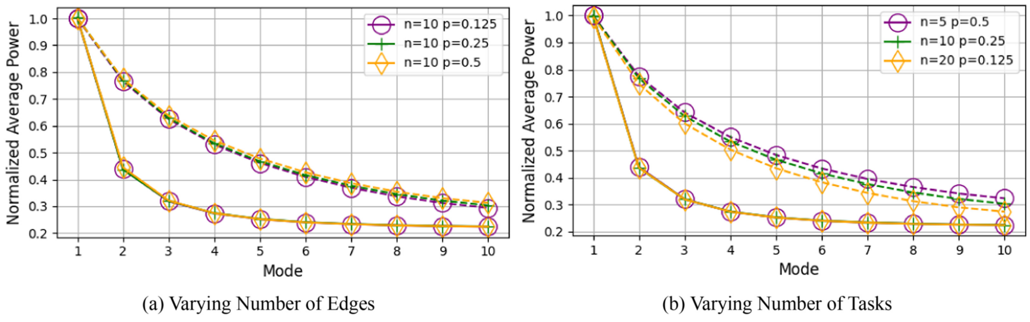


Fig. 11. Power optimization results with varying numbers of tasks and edges. Mixed values of r_is are used. The dashed lines are DPM counterparts.

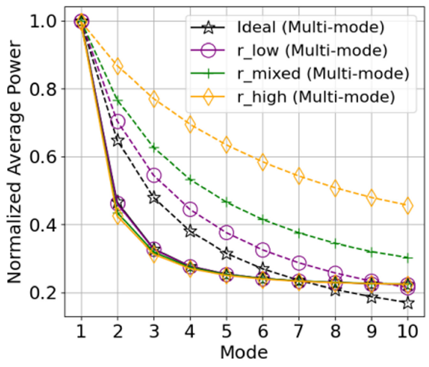


Fig. 12. Power optimization results with varying speed-independent ratio r_is. The dashed lines are DPM counterparts.

the number of nodes should be similar. This can be controlled by the edge probability parameter provided by GGen, which resulted in different values for each task set (n = 5, 10, 20) as specified in Section 6.1. In Figure 11(a), we first fix the number of tasks and only vary the edge parameter to see the effect on our multi-mode optimization compared to the DPM method in dashed lines. Each task set's result is averaged across 100 randomly generated DAGs with mixed values of r_is. As can be seen in the figure, the number of edges does not affect the amount of power savings in our multi-mode approach, meaning that it is not an important parameter of our optimization. Likewise, in Figure 11(b), we vary the number of tasks with selected edge probabilities, and they result in the same power savings as in Figure 11(a). In both of the preceding experiments, our multi-mode optimization achieves strictly better power savings than the DPM method.

6.2.5 Effect of Speed-Independent Ratio r_i. So far, we have been using mixed values of speed-independent ratio r_is in our experiments. This experiment investigates the effect of r_is on our multi-mode optimization by comparing four different sets of r_is. In addition to low, high, and mixed sets specified in Section 6.1, we added the ideal set where all r_is are set to zero. In Figure 12, the DPM counterparts in dashed lines vary significantly with r_is, whereas our multi-mode method shows a stable power-saving result. The power consumption of DPM increases as r_i gets higher because when r_i is high, the WCET does not scale well, resulting in longer execution time and hence more static power consumption. However, our multi-mode method can flexibly adjust the speed of the processor to save dynamic power regardless of the r_is. It is interesting to note that

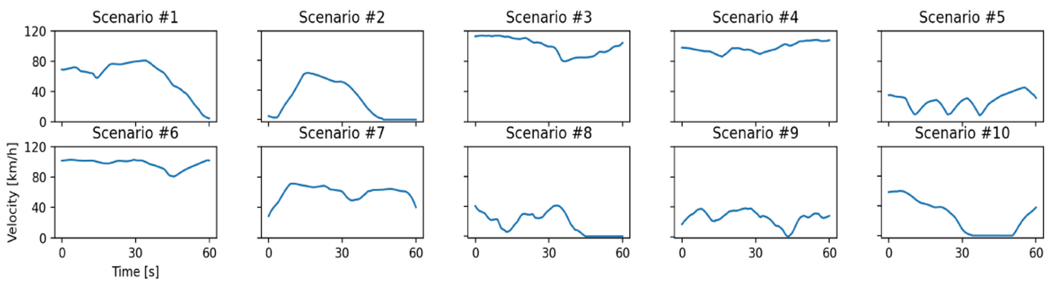


Fig. 13. Real-world driving scenarios from the comma.ai dataset.

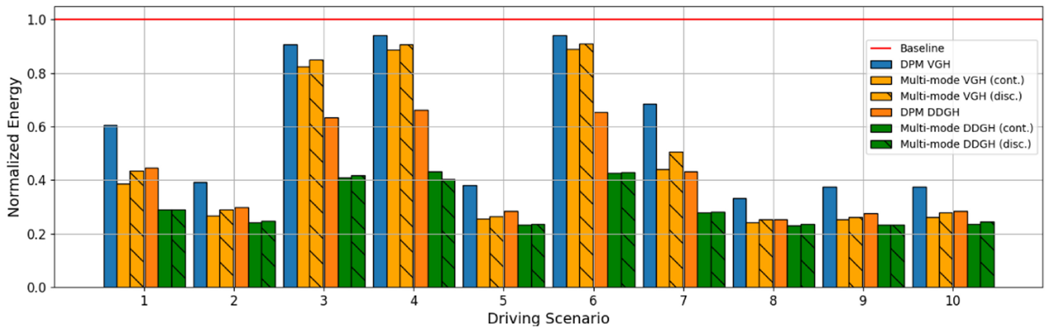


Fig. 14. Energy consumption with various driving scenarios.

DPM outperforms our multi-mode optimization in long deadline modes with the ideal r_i set. Yet in reality, it is unrealistic to have the ideal set of r_is. The exemplar DAG's r_is are close to the low r_i set in the figure, which could perform equal to or better than DPM in every mode.

6.2.6 Energy Optimization with Real-World Driving Scenarios. Figure 14 shows the energy consumption, including static and dynamic power, in driving scenarios using continuous and discrete frequencies. The exemplar DAG with its r_is and our multi-mode optimization were used in this experiment. For each driving scenario, the first three bars are the energy consumption of DPM and multi-mode methods using VGH, whereas the latter three bars are using our DDGH. We start our analysis with VGH solutions first and then discuss DDGH ones later. With continuous frequency, although DPM VGH achieves a significant energy reduction of 40.6% on average from baseline, the multi-mode VGH reduces further by timely utilizing lower frequencies. For discrete frequencies, we use the frequency that is greater than or equal to the optimal one assigned to each gang, as discussed in Section 4.4. Therefore, we expect that the energy consumption will increase slightly in our multi-mode method, whereas the DPM method is not affected because it uses only the maximum frequency. As a result, the energy consumption is increased by 6.2% on average for the multi-mode solution with discrete frequencies, but it still shows a significant reduction of 20.4% from DPM VGH. 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. On average, our multi-mode VGH method achieved 52.9% and 30.7% energy reductions compared with the baseline and the DPM VGH method, respectively. In the others, more than half of the total energy was saved.

On top of that, our DDGH can minimize energy consumption further by providing shorter end-to-end latency gang formation for both DPM and multi-mode optimization. The shorter end-to-end latency means that there is more slack from the end-to-end deadline, and thus a lower CPU frequency can be used in case of multi-mode optimization, or for the longer period the core is turned off for the DPM method. As a result, DDGH enables 27.8% and 25.5% improvement from VGH solutions for the DPM and the multi-mode method, respectively. For high-velocity scenarios #3, #4, and #6, the improvement by DDGH is significant because it enables the lower CPU frequencies for the same deadline. For other scenarios, the improvement is relatively small because they are already exploiting low CPU frequencies and the power consumption difference between these modes is small as depicted in Figure 9(a). For discrete frequencies, the energy consumption is increased by 0.9% on average. However, we noticed that for scenario #4, the energy consumption is actually decreased from its continuous frequency counterpart. The high-velocity scenario #4 has small changes in velocity, and the continuous solution utilizes a total of two modes for the entire 60 seconds. With discrete frequencies, slightly higher frequencies are mapped, which as a result result adds one more available mode whose deadline falls into scenario #4's deadline range. This is an exceptional case related to the mode granularity, and if we increase the number of modes, the discrete frequency solution will consume more energy as expected. On average, our method achieves 54.9% and 30.3% total energy reductions compared with the baseline and the traditional optimization with the current state-of-the-art gang formation heuristic (DPM VGH), respectively.

6.2.7 Safe Mode Change. In Section 5, we analyzed that the end-to-end deadline violation could only occur when the deadline gets shorter in shrinking deadline situations. The old instances in longer periods can cause extra delays in response time, and we provided an algorithm to predict the worst-case delay. To evaluate our algorithm and mode change safety, because we could not find any violation throughout the driving scenario experiments, we generated an artificial deadline graph with a sharp slope as in Figure 15 in addition to worst-case EDF scheduling. It is possible for this to happen in the real driving scenario, but it is not included in the dataset we used. The artificial driving scenario is basically a strong acceleration for a continuous 10 seconds with the worst data propagation patterns. The same exemplar DAG and its multi-mode solution were used for this experiment. As the deadline changes, as depicted by the blue line, mode changes are sporadically triggered, depicted by the rising and falling edges of the orange line that roughly follows the dynamic deadline. Note the small red staircase shapes at each falling edge, which depict the extra delays when shrinking deadlines analyzed by Algorithm 3. More specifically, their height represents the extra delay, whereas their width represents the transient interval for each mode change. Not only should the mode deadline curve (orange) not go above the dynamic deadline (blue) but also the extra delay curve should not (red staircase). In Figure 15(a), an end-to-end violation occurred at the end of the staircase at $t = 52.1$ marked with a circle. In Figure 15(b), we shifted the mode deadline curve to the left by applying a small safety margin that triggers mode change a little earlier such that the red staircase shape is not overlapped with the deadline curve. As a result, the end-to-end deadline violation was avoided.

7 RELATED WORK

Dynamic deadlines. Recent studies [40, 49] 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 [49] support dynamic deadlines with selective subgraph executions by considering varying time budgets. Heo et al. [40] 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 object detection accuracy. However, little work has been done with dynamic deadlines for the energy optimization of autonomous driving.

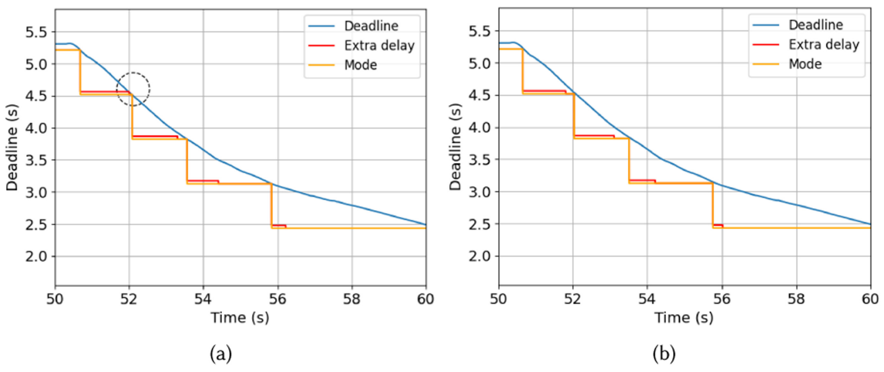


Fig. 15. An example of end-to-end deadline violation with the AEAP method.

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 DPM. In DVFS approaches, there is a body of literature to find speed levels through offline optimization [11, 25, 38, 57]. 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 [13, 46], 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.

For multicore systems, many approaches use the partitioned EDF approach such that a set of tasks is statically assigned to a CPU core [31, 54]. These works consider VFI, where all cores in the same island share the same clock frequency. Then, a global frequency for all cores that meet timing constraints is calculated for maximum energy efficiency. However, they do not consider the task precedence or end-to-end deadline, necessitating a different optimization formulation. Previous works on energy-efficient multicore real-time systems [9, 20, 38, 62, 68] assume that each CPU core can operate at different frequency levels. A task can be scheduled on any core at any time with a per-task speed factor. Although these techniques can achieve maximum energy efficiency in theory, they are not applicable to VFI processors due to hardware limitations. Xie et al. [64] combine non-DVFS and DVFS methods to minimize energy in heterogeneous real-time systems. The non-DVFS method is basically an energy-efficient scheduling algorithm for heterogeneous systems when tasks are assigned to processors with different power characteristics. After obtaining the schedule using the non-DVFS method, DVFS is applied to reclaim the slack. Our problem assumes a homogeneous processor, so without the non-DVFS method, the slack reclaiming is equivalent to a single end-to-end deadline optimization in our experiment (baseline).

DPM-Based Energy Optimization. As the static power consumption of processors has become more significant [44] in the past decade, DPM has been attracting more attention. In DPM approaches, cores are switched off during idle periods to reduce static power consumption. It is useful when the system is underutilized because each idle period should be large enough to offset the energy overhead of frequent switching. In general, however, it is known to have a better power efficiency than the DVFS-based methods in periodic real-time systems [10]. Many scheduling methods have been proposed to create large idle periods [7, 43, 59, 65]. Lee et al. [65] proposed leakage control EDF scheduling, but it requires additional hardware to calculate the sleep period. This impractical assumption was avoided in the work of Awan and Petters [7] by using a simpler sleep period calculation method and the ERTH (enhanced race-to-halt) algorithm. With ERTH, tasks run at full speed to secure longer sleep time.

There are fewer studies on energy-efficient DPM on multicore processors because of the difficulty of modeling idle intervals. Reghenzani et al. [55] study multi-level DPM with various energy-efficiency states. However, they also assume that the system is under a static deadline and find the best processor sleep state depending on the length of an idle period. Chen et al. [19] combined DPM and DVFS approaches on a multicore processor. They present a DVFS+DPM formulation for energy optimization and solve for static scheduling under a static deadline using mixed-integer linear programming. In addition to the scheduling under a static deadline, they assume a non-VFI architecture, so their approach cannot be applied in our setting. Hu et al. [42] also used a combined DPM and DVFS approach to minimize energy consumption while satisfying both response time and reliability constraints. Their Processor-Merging algorithm iteratively turns off a core at a time to minimize the number of cores used and then reclaims the remaining slack using DVFS. It assumes a workflow execution model where a task is triggered by the preceding task, whereas we assume multi-rate periodic execution, which is commonly used in many studies for automotive systems. The difference in the execution model results in a different scheduler and schedulability tests. Therefore, it cannot be directly applied to our problem.

Gang Scheduling. Traditional gang scheduling was proposed to minimize the makespan of parallel tasks by reducing synchronization overheads through scheduling interacting threads together as a gang [34]. Then, it is picked up by the real-time community to support parallel tasks because of increasing computing demands and the need for parallel processing [2, 22, 32, 36, 60]. These early works on real-time gang scheduling focus on schedulability improvement such that more tasks can be accepted with minimal deadline miss. In practice, however, predictability becomes an issue due to inter-core interference between co-running tasks that is difficult to predict. Recently, to mitigate the interference issue, RT-Gang scheduling was proposed [5] for periodic real-time tasks. RT-Gang is a more restrictive form of gang scheduling because of its one-gang-at-a-time policy restricting the co-scheduling of gangs to prevent interference. Thus, RT-Gang scheduling can make the interference manageable, which is suitable for real-time systems where predictability is important. Even though RT-Gang considers task precedence and successfully minimizes the end-to-end deadline, it does not consider energy consumption or system reconfiguration such as DVFS. Our work combines gang scheduling and adaptive system reconfiguration for energy efficiency.

8 CONCLUSION

In this work, we presented EASYR: Energy-Efficient Adaptive System Reconfiguration for dynamic deadlines in autonomous driving on multicore processors. Our work 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 EASYR approach utilizes the dynamic slack obtained by adaptively relaxing deadlines considering the vehicle's physical state. Toward that end, EASYR's GP-based optimization method proactively exploits the dynamic deadlines to find energy-efficient multi-mode system configurations. To enable our framework on multicore processors, we use RT-Gang scheduling to eliminate the challenges introduced by multicore processors: predictability and schedulability, which are important in the context of real-time systems. The gang scheduling entails the gang formation problem where we provided a greedy heuristic that outputs 32% shorter end-to-end delay compared to the state of the art. Moreover, EASYR's safe mode change protocol enables adaptive system reconfiguration between the predefined modes. Our experimental results demonstrate that EASYR achieves an average of 30.3% energy reduction over the conventional DPM method with the current state-of-the-art gang formation heuristic, demonstrating the great potential toward energy-efficient autonomous driving systems.

Based on the theoretical foundation presented in this article, in the future we plan to extend EASYR's system model considering heterogeneous CPU clusters and other accelerators such as graphics processing units. Additionally, instead of just velocity, we plan to consider other factors that can dynamically affect end-to-end deadlines such as adaptive redundant execution for fault tolerance. Other future extensions would be to jointly support both real-time and non-real-time tasks, in addition to supporting other popular schedulers such as the rate-monotonic fixed-priority scheduler with a different schedulability test and a utilization bound.

REFERENCES

[1] Tarek F. Abdelzaher, Vivek Sharma, and Chenyang Lu. 2004. A utilization bound for aperiodic tasks and priority driven scheduling. *IEEE Transactions on Computers* 53, 3 (2004), 334–350.

[2] Ashik Ahmed Bhuiyan, Kecheng Yang, Samsil Arefin, Abusayeed Saifullah, Nan Guan, and Zhishan Guo. 2019. Mixed-criticality multicore scheduling of real-time gang task systems. In *Proceedings of the 2019 IEEE Real-Time Systems Symposium (RTSS'19)*. 469–480. https://doi.org/10.1109/RTSS46320.2019.00048

[3] Waqar Ali, Rodolfo Pellizzoni, and Heechul Yun. 2020. Virtual gang based scheduling of real-time tasks on multicore platforms. *arXiv:1912.10959 [cs]* (Feb. 2020).

[4] Waqar Ali, Rodolfo Pellizzoni, and Heechul Yun. 2021. Virtual gang scheduling of parallel real-time tasks. In *Proceedings of the 2021 Design, Automation, and Test in Europe Conference and Exhibition (DATE'21)*. 270–275. https://doi.org/10.23919/DATE51398.2021.9474015

[5] W. Ali and H. Yun. 2019. RT-G: Real-time gang scheduling framework for safety-critical systems. In *Proceedings of the 25th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'19)*. 143–155.

[6] AUTOSAR. (n.d). *Specification of Timing Extensions. AUTOSAR Classic Platform 4.3.1.* AUTOSAR.

[7] M. A. Awan and S. M. Petters. 2011. Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems. In *Proceedings of the 23rd Euromicro Conference on Real-Time Systems (ECRTS'11)*. 92–101. https://doi.org/10.1109/ECRTS.2011.17

[8] Hakan Aydin, Vinay Devadas, and Dakai Zhu. 2006. System-level energy management for periodic real-time tasks. In *Proceedings of the 2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*. 313–322. https://doi.org/10.1109/RTSS.2006.48

[9] H. Aydin and Qi Yang. 2003. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings of the International Parallel and Distributed Processing Symposium*. 9. https://doi.org/10.1109/IPDPS.2003.1213225

[10] M. Bambagini, M. Bertogna, and G. Buttazzo. 2014. On the effectiveness of energy-aware real-time scheduling algorithms on single-core platforms. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA'14)*. 1–8. https://doi.org/10.1109/ETFA.2014.7005108

[11] Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. 2016. Energy-aware scheduling for real-time systems: A survey. *ACM Transactions on Embedded Computing Systems* 15, 1 (Jan. 2016), Article 7, 34 pages.

[12] Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. 2016. Energy-aware scheduling for real-time systems: A survey. *ACM Transactions on Embedded Computing Systems* 15, 1 (Jan. 2016), Article 7, 34 pages. https://doi.org/10.1145/2808231

[13] M. Bambagini, F. Prosperi, M. Marinoni, and G. Buttazzo. 2011. Energy management for tiny real-time kernels. In *Proceedings of the International Conference on Energy Aware Computing*. 1–6. https://doi.org/10.1109/ICEAC.2011.6136687

[14] Robert Basmadjian and Hermann de Meer. 2012. Evaluating and modeling power consumption of multi-core processors. In *Proceedings of the 2012 3rd International Conference on Future Systems: Where Energy, Computing, and Communication Meet (e-Energy'12)*. 1–10. https://doi.org/10.1145/2208828.2208840

[15] Ashikahmed Bhuiyan, Di Liu, Aamir Khan, Abusayeed Saifullah, Nan Guan, and Zhishan Guo. 2020. Energy-efficient parallel real-time scheduling on clustered multi-core. *IEEE Transactions on Parallel and Distributed Systems* 31, 9 (2020), 2097–2111.

[16] Ashikahmed Bhuiyan, Sai Sruti, Zhishan Guo, and Kecheng Yang. 2019. Precise scheduling of mixed-criticality tasks by varying processor speed. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems (RTNS'19)*. ACM, New York, NY, 123–132. https://doi.org/10.1145/3356401.3356410

[17] Enrico Bini, Giorgio Buttazzo, and Giuseppe Lipari. 2005. Speed modulation in energy-aware real-time systems. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS'05)*. 3–10. https://doi.org/10.1109/ECRTS.2005.29

[18] Stephen Boyd, Seung-Jean Kim, Lieven Vandenberghe, and Arash Hassibi. 2007. A tutorial on geometric programming. *Optimization and Engineering* 8, 1 (2007), 67.

[19] Gang Chen, Kai Huang, and Alois Knoll. 2014. Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination. *ACM Transactions on Embedded Computing Systems* 13, 3s (March 2014), 1–21. https://doi.org/10.1145/2567935

[20] Jian-Jia Chen, Heng-Ruey Hsu, and Tei-Wei Kuo. 2006. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*. 408–417. https://doi.org/10.1109/RTAS.2006.25

[21] Tianyang Chen and Linh Thi Xuan Phan. 2018. SafeMC: A system for the design and evaluation of mode-change protocols. In *Proceedings of the 2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'18)*. IEEE, Los Alamitos, CA, 105–116.

[22] Hoon Sung Chwa, Jinkyu Lee, Kieu-My Phan, Arvind Easwaran, and Insik Shin. 2013. Global EDF schedulability analysis for synchronous parallel tasks on multicore platforms. In *Proceedings of the 2013 25th Euromicro Conference on Real-Time Systems*. 25–34. https://doi.org/10.1109/ECRTS.2013.14

[23] Comma.ai. (n.d). Comma.ai Driving Dataset. Retrieved December 20, 2022 from https://archive.org/details/comma-dataset.

[24] Daniel Cordeiro, Grégory Mounié, Swann Perarnau, Denis Trystram, Jean-Marc Vincent, and Frédéric Wagner. 2010. Random graph generation for scheduling simulations. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools'10)*. Article 60, 10 pages.

[25] Chen Da-Ren, Chen Young-Long, and Chen You-Shyang. 2014. Time and energy efficient DVS scheduling for real-time pinwheel tasks. *Journal of Applied Research and Technology* 12, 6 (Dec. 2014), 1025–1039. https://doi.org/10.1016/S1665-6423(14)71663-3

[26] Dakai Zhu, R. Melhem, and D. Mosse. 2004. The effects of energy management on reliability in real-time embedded systems. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD'04)*. 35–40. https://doi.org/10.1109/ICCAD.2004.1382539

[27] Electric Vehicle Database. (n.d). Energy Consumption of Full Electric Vehicles. Retrieved December 20, 2022 from https://ev-database.org/cheatsheet/energy-consumption-electric-car.

[28] Abhijit Davare, Qi Zhu, Marco Di Natale, Claudio Pinello, Sri Kanajan, and Alberto Sangiovanni-Vincentelli. 2007. Period optimization for hard real-time distributed automotive systems. In *Proceedings of the 44th Annual Design Automation Conference*. 278–283.

[29] Arturo Dávila and Mario Nombela. 2011. *Sartre-Safe Road Trains for the Environment Reducing Fuel Consumption Through Lower Aerodynamic Drag Coefficient.* Technical Report. SAE.

[30] L. C. Davis. 2012. Stability of adaptive cruise control systems taking account of vehicle response time and delay. *Physics Letters A* 376, 40-41 (2012), 2658–2662.

[31] Vinay Devadas and Hakan Aydin. 2010. Coordinated power management of periodic real-time tasks on chip multiprocessors. In *Proceedings of the International Conference on Green Computing*. 61–72. https://doi.org/10.1109/GREENCOMP.2010.5598261

[32] Zheng Dong and Cong Liu. 2019. Analysis techniques for supporting hard real-time sporadic gang task systems. *Real-Time Systems* 55, 3 (July 2019), 641–666. https://doi.org/10.1007/s11241-018-9318-7

[33] Nico Feiertag, Kai Richter, Johan Nordlander, and Jan Jonsson. 2009. A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In *Proceedings of the IEEE Real-Time Systems Symposium*. IEEE, Los Alamitos, CA.

[34] Dror G. Feitelson and Larry Rudolph. 1992. Gang scheduling performance benefits for fine-grain synchronization. *Journal of Parallel and Distributed Computing* 16 (1992), 306–318.

[35] James H. Gawron, Gregory A. Keoleian, Robert D. De Kleine, Timothy J. Wallington, and Hyung Chul Kim. 2018. Life cycle assessment of connected and automated vehicles: Sensing and computing subsystem and vehicle level effects. *Environmental Science & Technology* 52, 5 (March 2018), 3249–3256. https://doi.org/10.1021/acs.est.7b04576

[36] Joël Goossens and Pascal Richard. 2016. Optimal scheduling of periodic gang tasks. *Leibniz Transactions on Embedded Systems* 3, 1 (June 2016), Article 4, 18 pages. https://doi.org/10.4230/LITES-v003-i001-a004

[37] Michael Grant and Stephen Boyd. 2014. CVX: Matlab Software for Disciplined Convex Programming, version 2.1. Retrieved December 20, 2022 from http://cvxr.com/cvx.

[38] Zhishan Guo, Ashikahmed Bhuiyan, Di Liu, Aamir Khan, Abusayeed Saifullah, and Nan Guan. 2019. Energy-efficient real-time scheduling of DAGs on clustered multi-core platforms. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'19)*. IEEE, Los Alamitos, CA, 156–168. https://doi.org/10.1109/RTAS.2019.00021

[39] Arne Hamann, Dakshina Dasari, and Falk Wurst. 2019. WATERS Industrial Challenge. Retrieved December 20, 2022 from https://www.ecrts.org/forum/download/WATERS_Industrial_Challenge_2019_final.pdf.

[40] Seonyeong Heo, Sungjun Cho, Youngsok Kim, and Hanjun Kim. 2020. Real-time object detection system with multi-path neural networks. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'20)*. IEEE, Los Alamitos, CA, 174–187.

[41] Sebastian Herbert and Diana Marculescu. 2007. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of the 2007 International Symposium on Low Power Electronics and Design (ISLPED'07)*. 38–43. https://doi.org/10.1145/1283780.1283790

[42] Biao Hu, Zhengcai Cao, and Mengchu Zhou. 2022. Energy-minimized scheduling of real-time parallel workflows on heterogeneous distributed computing systems. *IEEE Transactions on Services Computing* 15, 5 (2022), 2766–2779. https://doi.org/10.1109/TSC.2021.3054754

[43] K. Huang, L. Santinelli, J. Chen, L. Thiele, and G. C. Buttazzo. 2009. Periodic power management schemes for real-time event streams. In *Proceedings of the 48th IEEE Conference on Decision and Control (CDC'09) Held Jointly with the 2009 28th Chinese Control Conference*. 6224–6231. https://doi.org/10.1109/CDC.2009.5400034

[44] Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and Giorgio C. Buttazzo. 2011. Applying real-time interface and calculus for dynamic power management in hard real-time systems. *Real-Time Systems* 47, 2 (March 2011), 163–193. https://doi.org/10.1007/s11241-011-9115-z

[45] T. Ishihara and H. Yasuura. 1998. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design.* 197–202. https://doi.org/10.1145/280756.280894

[46] Ravindra Jejurikar and Rajesh Gupta. 2005. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *Proceedings of the 42nd Design Automation Conference*. 111–116.

[47] Sebastian Kehr, Eduardo Quiñones, Dominik Langen, Bert Böddeker, and Günter Schäfer. 2017. Parcus: Energy-aware and robust parallelization of AUTOSAR legacy applications. In *Proceedings of the 2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'17)*. IEEE, Los Alamitos, CA, 343–352.

[48] Lukas Krawczyk, Mahmoud Bazzal, Ram Prasath Govindarajan, and Carsten Wolff. 2019. An analytical approach for calculating end-to-end response times in autonomous driving applications. In *Proceedings of the 10th International Workshop on Analysis Tools and Methodologies for Embedded and Realtime Systems (WATERS'19)*.

[49] Seulki Lee and Shahriar Nirjon. 2020. SubFlow: A dynamic induced-subgraph strategy toward real-time DNN inference and training. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'20)*. IEEE, Los Alamitos, CA, 15–29.

[50] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md. E. Haque, Lingjia Tang, and Jason Mars. 2018. The architectural implications of autonomous driving: Constraints and acceleration. In *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems*. 751–766.

[51] Chung Laung Liu and James W. Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM* 20, 1 (1973), 46–61.

[52] Sidi Lu and Weisong Shi. 2021. The emergence of vehicle computing. *IEEE Internet Computing* 25, 3 (May 2021), 18–22. https://doi.org/10.1109/MIC.2021.3066076

[53] Biswadip Maity, Saehanseul Yi, Dongjoo Seo, Leming Cheng, Sung-Soo Lim, Jong-Chan Kim, Bryan Donyanavard, and Nikil Dutt. 2021. Chauffeur: Benchmark suite for design and end-to-end analysis of self-driving vehicles on embedded systems. *ACM Transactions on Embedded Computing Systems* 20, 5s (Sept. 2021), Article 74, 22 pages. https://doi.org/10.1145/3477005

[54] Santiago Pagani and Jian-Jia Chen. 2014. Energy efficiency analysis for the single frequency approximation (SFA) scheme. *ACM Transactions on Embedded Computing Systems* 13, 5s (Dec. 2014), 1–25. https://doi.org/10.1145/2660490

[55] Federico Reghenzani, Ashikahmed Bhuiyan, William Fornaciari, and Zhishan Guo. 2021. A multi-level DPM approach for real-time DAG tasks in heterogeneous processors. In *Proceedings of the 2021 IEEE Real-Time Systems Symposium (RTSS'21)*. 14–26. https://doi.org/10.1109/RTSS52674.2021.00014

[56] Tyler G. R. Reid, Sarah E. Houts, Robert Cammarata, Graham Mills, Siddharth Agarwal, Ankit Vora, and Gaurav Pandey. 2019. Localization requirements for autonomous vehicles. *SAE International Journal of Connected and Automated Vehicles* 2, 3 (Sept. 2019), 173–190. https://doi.org/10.4271/12-02-03-0012

[57] Abusayeed Saifullah, Sezana Fahmida, Venkata P. Modekurthy, Nathan Fisher, and Zhishan Guo. 2020. CPU energy-aware parallel real-time scheduling. In *Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS'20)*.

[58] Saad Zia Sheikh and Muhammad Adeel Pasha. 2018. Energy-efficient multicore scheduling for hard real-time systems: A survey. *ACM Transactions on Embedded Computing Systems* 17, 6 (Dec. 2018), Article 94, 26 pages. https://doi.org/10.1145/3291387

[59] Joohyung Sun, Hyeonjoong Cho, Arvind Easwaran, Ju-Derk Park, and Byeong-Cheol Choi. 2019. Flow network-based real-time scheduling for reducing static energy consumption on multiprocessors. *IEEE Access* 7 (2019), 1330–1344. https://doi.org/10.1109/ACCESS.2018.2886562

[60] Niklas Ueter and Mario Günzel. 2021. Hard real-time stationary GANG-scheduling. In *Proceedings of the 33rd Euromicro Conference on Real-Time Systems (ECRTS'21)*. 1–19.

[61] Woonseok Kim, Jihong Kim, and Sang Lyul Min. 2004. Preemption-aware dynamic voltage scaling in hard real-time systems. In *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*. 393–398. https://doi.org/10.1109/LPE.2004.241299

[62] Changjiu Xian, Yung-Hsiang Lu, and Zhiyuan Li. 2007. Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time. In *Proceedings of the 2007 44th ACM/IEEE Design Automation Conference*. 664–669.

[63] Guoqi Xie, Hao Peng, Zhetao Li, Jinlin Song, Yong Xie, Renfa Li, and Keqin Li. 2018. Reliability enhancement toward functional safety goal assurance in energy-aware automotive cyber-physical systems. *IEEE Transactions on Industrial Informatics* 14, 12 (2018), 5447–5462.

[64] Guoqi Xie, Gang Zeng, Xiongren Xiao, Renfa Li, and Keqin Li. 2017. Energy-efficient scheduling algorithms for real-time parallel applications on heterogeneous distributed embedded systems. *IEEE Transactions on Parallel and Distributed Systems* 28, 12 (Dec. 2017), 3426–3442. https://doi.org/10.1109/TPDS.2017.2730876

[65] Yann-Hang Lee, K. P. Reddy, and C. M. Krishna. 2003. Scheduling techniques for reducing leakage power in hard real-time systems. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)*. 105–112. https://doi.org/10.1109/EMRTS.2003.1212733

[66] Saehanseul Yi, Tae-Wook Kim, Jong-Chan Kim, and Nikil Dutt. 2021. Energy-efficient adaptive system reconfiguration for dynamic deadlines in autonomous driving. In *Proceedings of the 2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC'21)*. 96–104. https://doi.org/10.1109/ISORC52013.2021.00023

[67] Ying Zhang and K. Chakrabarty. 2003. Energy-aware adaptive checkpointing in embedded real-time systems. In *Proceedings of the 2003 Design, Automation, and Test in Europe Conference and Exhibition (DATE'03)*. 918–923. https://doi.org/10.1109/DATE.2003.1253723

[68] Gang Zeng, Tetsuo Yokoyama, Hiroyuki Tomiyama, and Hiroaki Takada. 2009. Practical energy-aware scheduling for real-time multiprocessor systems. In *Proceedings of the 2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. 383–392. https://doi.org/10.1109/RTCSA.2009.47

Received 6 April 2022; revised 11 October 2022; accepted 12 October 2022