Model-based Design of Resource-Efficient Automotive Control Software

Invited Paper

Wanli Chang
TUM CREATE, Singapore
Singapore Institute of Technology
wanli.chang@tum-create.edu.sg

Debayan Roy, Licong Zhang, Samarjit Chakraborty
Institute for Real-Time Computer Systems
TU Munich, Germany
samarjit@tum.de

ABSTRACT

Automotive platforms today run hundreds of millions of lines of software code implementing a large number of different control applications spanning across safety-critical functionality to driver assistance and comfort-related functions. While such control software today is largely designed following model-based approaches, the underlying models do not take into account the details of the implementation platforms, on which the software would eventually run. Following the state-of-the-art in control theory, the focus in such design is restricted to ensuring the stability of the designed controllers and meeting control performance objectives, such as settling time or peak overshoot. However, automotive platforms are highly cost-sensitive and the issue of designing "resource-efficient" controllers has largely been ignored so far and is addressed using very ad hoc techniques. In this paper, we will illustrate how, following traditional embedded systems design oriented thinking, computation, communication and memory issues can be incorporated in the controller design stage, thereby resulting in control software not only satisfying the usual control performance metrics but also making efficient utilization of the resources on distributed automotive architectures.

1. INTRODUCTION

Modern automotive Electrical/Electronic (E/E) systems are becoming increasingly larger and more complex. A premium car today can contain up to 100 Electronic Control Units (ECUs) and hundreds of millions of lines of software code running on them. These software codes implement functions spanning across vehicle dynamics control, body components control to advanced driver assistance systems. While such control software today is largely designed following model-based approaches, the underlying models do not take into account the details of the implementation platforms. To guarantee the safety and performance of the control software, usually conservative assumptions are made when the controllers are designed. This would often lead to inefficient utilization of the embedded platform resources, for example, the communication, computation and memory resources.

The limitation of the resources is always one of the main constraints on an embedded platform. This issue is even more important for the automotive domain. First of all, the automotive industry is highly cost-sensitive and providing more resources is often coupled with increasing cost. Furthermore, the design and implementation process of such a system usually follows an iterative scheme and the issue of dealing with legacy components and configurations needs to be taken into consideration. Hence, sufficient resources are not only required for the current design iteration, but also need to be provisioned for possible future components. Therefore, the topic of resource-efficient design is both helpful and necessary for automotive embedded systems.

Resource-efficient design, with notions of efficiency being those related to computation, communication and memory, has been one of the main research areas in the embedded system and real-time system community for some time. However, when it comes to control algorithms, none of these notions have been used. Instead, the focus has been more on the stability and performance of the control software. As more control applications are getting incorporated into the automotive E/E system, there is an increasing need for a systematic approach of resource-efficient implementation of controllers. Since the control algorithms are designed in a model-based fashion, the goal here is to connect model-based design of controllers with techniques for resource-efficient implementations.

The resources on an embedded platform can be divided into different categories. Examples of important resources include the communication resource, the computation resource and the memory resource. The communication resource can generally be represented as the bandwidth of a communication bus or a network link, which denotes the number of bits that can be transmitted per second. Therefore, there is only a limited amount of data that can be transmitted within a specific time frame. More precise characterization of the communication resource, however, is protocol-specific. In a Time Division Multiple Access (TDMA) bus, for example, the communication resource can be translated into the number of utilized slots. The computation resource usually means the available execution time of a processor. Considering multiple applications sharing one single-core processor, each application is allocated a certain period of execution time. This execution time allocation can be static (e.g., time-triggered scheduling) or dynamic (e.g., priority-based scheduling). The memory resource mainly refers to the size of memory available on, e.g., an ECU. There are typically two levels of memory — on-chip cache and main memory. The main memory has a large size and can thus store all the application programs and data, yet experiences high read/write latencies (hundreds of processor cycles). The on-chip cache is faster (several processor cycles), yet usually limited in size due to its high cost.

In this paper, we illustrate how resource-efficient design following the traditional embedded system design oriented thinking can be incorporated into the control software design, thereby resulting in control software not only satisfying the usual control perfor-

This work is supported by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) program.

ICCAD '16, November 07-10, 2016, Austin, TX, USA

 ISBN 978-1-4503-4466-1/16/11…$15.00

DOI: http://dx.doi.org/10.1145/2966986.2980075

mance metrics but also making efficient utilization of the resources. Here, we first provide a brief review of the basics of the embedded control software, including the feedback control systems and the embedded architecture. Then, we illustrate the resource-efficient design paradigm using three state-of-the-art approaches in communication resource (Section 3), computation resource (Section 4) and memory resource (Section 5), respectively. Finally, in Section 6, we discuss some possibilities and directions for future work.

2. BACKGROUND

2.1 Feedback control systems

System dynamics: The continuous-time dynamic behavior of a feedback linear single-input single-output (SISO) control system can be modeled by the following differential equations,

$$\dot{x}(t) = Ax(t) + Bu(t),\ y(t) = Cx(t), \tag{1}$$

where $x(t) \in \mathbb{R}^n$, $y(t)$ and $u(t)$ represent system states, system output and control input, respectively and A, B, C are respectively the system matrix, input matrix and output matrix of the corresponding dimensions. In an embedded implementation, the continuous-time model can be discretized according to a sampling period h,

$$x[k+1] = A_d x[k] + B_d u[k],\ y[k] = C_d x[k], \tag{2}$$

where $x[k]$, $y[k]$ and $u[k]$ denote the discrete-time system states, the system output and the control input, at the kth instance ($k \in \mathbb{Z}^*$), respectively. A_d, B_d, C_d can be computed as,

$$A_d = e^{Ah},\ B_d = \int_0^h (e^{At}dt)B,\ C_d = C. \tag{3}$$

Distributed implementation: A feedback control loop is implemented with software codes running on the ECUs, which can be partitioned into three different types of tasks: (i) *sensor task* measures the system states (using sensors) of the physical plant under control if measurable. (ii) *controller task* computes the control input based on the measured system states. (iii) *actuator task* applies the control input (using actuators) to the physical plant. Depending on the actual design requirements (e.g., the placement of the sensors and actuators), these tasks can either be mapped onto the same ECU or on different ECUs. In the case of a distributed control application, the data between the tasks are transmitted over the bus. The time between two consecutive instances of sensor task is defined as the *sampling period* (denoted as h) of the control application. Usually the control applications are implemented with a constant sampling period. However, in some cases, non-uniform sampling can also be applied. The time between the start of the sensor task and the end of the actuator task is defined as the *sensor-to-actuator delay* or *closed-loop delay* (denoted as d).

Control performance: There are different metrics to measure the performance of a control system. Here we consider two common ones. (i) The steady-state performance of a control application can be commonly measured by a *cost function*, which in the discrete case can be represented as

$$J = \sum_{k=0}^{n} [\lambda u[k]^2 + (1-\lambda)\sigma[k]^2]h, \tag{4}$$

where λ is a weight taking the value between 0 and 1, $u[k]$ is the control input and $\sigma[k] = |r - y[k]|$ is the tracking error. We further consider (ii) the *settling time* ξ as another control performance metric, where ξ denotes the time necessary for the system to reach and remain within 1% of the reference value,

$$J = \xi. \tag{5}$$

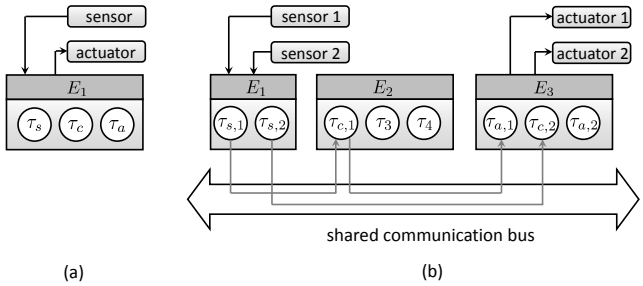

Figure 1: (a) Single ECU case example. τ_s, τ_c and τ_a represent respectively the sensor, controller and actuator task of a feedback control loop. (b) Distributed case example. $\tau_{s,i}$, $\tau_{c,i}$ and $\tau_{a,i}$ are of the ith control loop and τ_3 and τ_4 represent other non-control tasks that can also be mapped on E_2.

Depending on the specific requirements of the control application, different performance metrics can be used.

Controller design: The control input $u[k]$ of a control application based on state-feedback control can be designed as

$$u[k] = Kx[k - \left\lfloor \frac{d}{h} \right\rfloor] + Fr, \tag{6}$$

where K is the feedback gain to stabilize the system and F is the static feedforward gain to track the reference r for $y[k]$ to reach. The feedback and feedforward gains can be designed with controller design techniques like pole-placement [1] and there are sufficient research works on the controller design taking the sensor-to-actuator delay into account [2].

Pole-placement using PSO For a control application, in order to design the controller that optimizes the control performance for a given sampling period and sensor-to-actuator delay, an optimization problem for the pole-placement can be formulated. In this problem, the decision variables are poles of the closed-loop system and the objective is to optimize the value of the selected control performance metric. Constraints like the stability of the system and control input saturation need to be satisfied. It is challenging to solve such a constrained non-convex optimization problem with significant non-linearity. Here a highly efficient and scalable Particle Swarm Optimization (PSO) technique can be used [3].

2.2 Embedded systems architecture

An automotive E/E architecture typically consists of a number of ECUs connected by different bus protocols. The ECUs are categorized in different domains (e.g., chassis, powertrain, body and infotainment) according to the nature of the functions mapped on them. In each domain, a cluster of ECUs are connected by one or more bus systems and the domains are connected to a central gateway, which can provide inter-domain communication. An embedded controller mapped on such an architecture is usually implemented with one or multiple tasks (e.g., sensor, controller and actuator task), where each task is a piece of software code running on the processor. The tasks can either be mapped on a single ECU or distributed on multiple ECUs. Figure 1 shows an example of both cases. In the case of a single ECU, these tasks are executed on the same processor, while in a distributed architecture, the sensor, controller and actuator tasks can also be mapped on different processor and the data between the tasks are transferred on the bus as messages. It is also common, that tasks of different controllers are mapped in common ECUs, where resources like the communication, computation and memory are shared between these control applications. Therefore, how to allocate the resources for the soft-

ware implementation of the controllers forms the problem of architecture design. This can be translated into design parameters like the task partitioning and mapping, the scheduling on the processors and the bus, the use of cache, etc. More specific definitions of these parameters depend on characteristics of the embedded architecture, for example, the scheduling schemes used on the processors and the bus protocols used for the communication.

3. COMMUNICATION-EFFICIENT DESIGN

In this section, we illustrate the resource-efficient model-based design of automotive control software using a state-of-the-art co-optimization approach [4] that synthesizes simultaneously controllers, task and communication schedules for a FlexRay-based ECU network, and in the process optimizes both the overall system control performance and the communication resource usage. Here, we will first explain the specific problem setting and then the approach and the results.

3.1 Problem setting

We consider a distributed architecture, where a set of ECUs are connected via FlexRay bus, running several control applications, $C_i \in C$. We assume that each control application C_i is partitioned into three software tasks, i.e., *sensor task* ($\tau_{s,i}$), *control task* ($\tau_{c,i}$), and *actuator task* $\tau_{a,i}$, mapped on different ECUs. Then the sensor values measured by $\tau_{s,i}$ are sent on the bus via the message $m_{s,i}$ and the control input calculated by $\tau_{c,i}$ is sent via the message $m_{c,i}$. Furthermore, we consider that time-triggered non-preemptive scheduling scheme is exhibited by the Real-Time Operating Systems (RTOS) on the ECUs. Each task of an application is considered to be periodic and its schedule is defined by the tuple consisting of the *offset*, the *period* and the Worst-Case Execution Time (WCET) of the task.

Here, we consider transmission of messages only on the static segment of FlexRay which exhibits time-triggered communication. A FlexRay schedule corresponding to the message $m_{x,i}$ can be represented by the tuple of the *slot id*, the *base cycle* and the *repetition rate*. The slot id identifies the slot within a bus cycle, in which the message is sent. The base cycle denotes the communication cycle when the message is first sent. The repetition rate $r_{x,i}$ is the number of communication cycles that elapse between two consecutive transmissions of the same frame and takes the value $r_{x,i} \in \{2^n | n \in \{0, 1, ..., 6\}\}$. We consider the FlexRay Version 3.0.1 [5], where *slot multiplexing* among different ECUs is allowed. It means that a particular slot can be assigned to different ECUs in different cycles.

For FlexRay time-triggered communication, the bus usage U can be defined as the percentage of the slots in the static segment that is allocated to the control applications. In this case, the smaller the value of U, the better is the resource efficiency as more slots can be left vacant for use by other non-control applications.

Depending on the specific requirements of the control application, one of the two performance metrics discussed in Section 2.1 can be used. In both the metrics, smaller value of J implies better control performance. In a system consisting of multiple control applications with different characteristics and performance metrics, it is required to normalize the control performance in order to compare and combine them. Each closed-loop C_i with control performance J_i must satisfy some performance threshold J_i^r defined by the user. Thus, the control performance can be normalized as a fraction of the required value J_i^r and, therefore, the overall control quality J_0 of a set of control loops C can be formulated as a weighted sum of the normalized values.

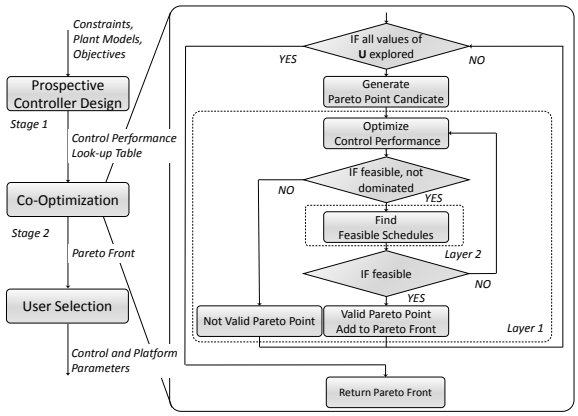

Figure 2: Design flow of the co-optimization approach

Co-optimization problem: The co-optimization problem can be formulated to find a set of parameters for each $C_i \in C$, which can be denoted as $par_i = \{\tau_{s,i}, \tau_{c,i}, \tau_{a,i}, m_{s,i}, m_{c,i}, h_i, K_i, F_i\}$, while optimizing the bus usage U and the overall control quality J_0.

3.2 The co-optimization approach

The approach to solve the co-optimization problem is divided into two stages, i.e., the prospective controller design and the co-optimization, as shown in Figure 2. Eventually, several optimal design configurations are synthesized, which can be represented by a Pareto front depicting the design trade-off between the overall control performance and the bus usage. From the set of Pareto points, the designer can select the one that is the most suitable for the overall design requirements and the corresponding design configuration is implemented. The two stages of the approach are explained in the following.

Prospective controller design: Besides the control plant model, the control performance J_i of the control application C_i depends mainly on three factors: (i) the sampling period h_i, (ii) the sensor-to-actuator delay d_i and (iii) the controller gains K_i and F_i. In this stage, for each valid combination of the sampling period and delay, a set of optimal controller gains need to be designed. However, we consider schedules for the tasks and the messages leading to the case where the delay equals to the sampling period, i.e., $d_i = h_i$. This is because it would reduce the dimensions of the design space from (i) - (iii) to only (i) and (iii), thus reducing the complexity and enhancing the scalability. With $d_i = h_i$, the closed-loop system experiences one sampling period delay and the pole-placement method explained in Section 2.1 is employed. Moreover, we make use of the fact that the sampling period can only take discrete values and further prune the design space. Since each control application C_i is implemented by the tasks $\tau_{s,i}$, $\tau_{c,i}$, $\tau_{a,i}$ and messages $m_{s,i}$, $m_{c,i}$, there is a dependency between the sampling period h_i and the repetition rates of the messages $r_{s,i}$, $r_{c,i}$, which can be represented as

$$h_i = r_{s,i} T_{bus} = r_{c,i} T_{bus}, \tag{7}$$

where T_{bus} is the bus clock cycle. Due to the fact that $r_{s,i}$, $r_{c,i}$ can only take discrete values in $\{2^k | k \in \{0, 1, ..., 6\}\}$, the choice of h_i is also constrained to the corresponding discrete values. Considering this, in this stage, we determine for each control application the controller gains at each possible value of the sampling period that optimize the control performance. After this stage, a look-up table for each control application C_i can be formulated, where for each possible sampling period h_i^k, an optimal control performance J_i^{k*} corresponding to the controller gains K_i^{k*} and F_i^{k*} can be assigned.

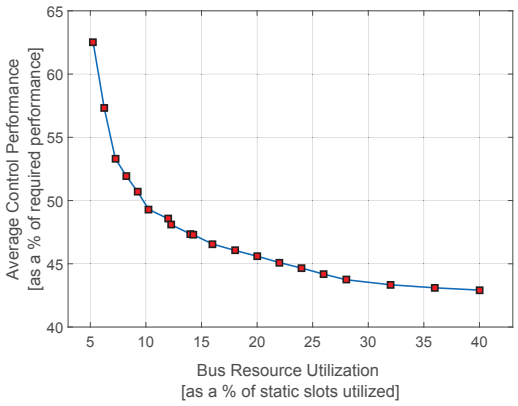

Figure 3: Pareto front.

In the co-optimization stage, these tables will be used to formulate the objective of overall control performance.

Co-optimization: The communication-efficient model-based control software design for the setting considered can be formulated as a constrained optimization problem with two objectives, i.e., the overall system control performance and the bus usage. In this case, the two design objectives are noticed to be often conflicting, and therefore a much more informative and designer-friendly co-optimization approach is to first generate a Pareto front and let the designer explore the trade-off between the two objectives according to his customized preference.

A customized optimization approach as shown in Figure 2 is employed to obtain the desired Pareto front. The objective on bus usage U is discrete and can only take a limited number of values. Therefore, for each possible value of U in ascending order, given the equality constraint on U, we solve the optimization problem with J_0 as the single objective and obtain a solution. The additional constraint is that J_0 of this solution has to be better than J_0 of the last solution (Pareto criterion), in order to ensure that all solutions are non-dominated. Therefore, the co-optimization problem with two objectives is turned into a series of single-objective optimization problems, where each may generate a Pareto point.

Widely applied approaches like Mixed Integer Linear Programming (MILP) or meta-heuristic methods cannot be applied directly to solve each of the single-objective optimization problems due to reasons like non-linear constraints, small solution space, etc. However, considering that both the objectives only depend on the sampling periods, a nested two-layered technique, as shown in Figure 2, is employed to solve each of the problems. On Layer 1, the outer layer, for a given value of U, a set of sampling periods $\{h_i\}$ corresponding to the set of applications $\{C_i\}$ are determined, which satisfy the value of U and optimize J_0 such that every application control performance requirement is fulfilled. On Layer 2, the inner layer, task and message schedules are synthesized according to the given values of sampling periods such that all the system constraints are satisfied. This process is iterative in the way that if the synthesis fails in Layer 2, the algorithm goes back to Layer 1 for the next best solution until Pareto criterion is satisfied. This optimization technique ensures optimality and also efficiency.

3.3 Case study and results

As a case study, we have considered 5 control applications running on 3 ECUs connected by FlexRay bus. The Pareto front of the system is obtained as shown in Figure 3. The bus usage values range from 5.25% to 40% of the bus bandwidth in the static segment. The overall control performance varies from 42.92% to 62.54% of the required value. It is obvious that there is a large freedom among these viable design points. For larger system size we can expect more trade-off opportunities.

Table 1: An example OSEK/VDX OS time table

Time	Release
$0ms$	Applications with periods of $2ms$/$5ms$/$10ms$
$2ms$	Applications with the period of $2ms$
$4ms$	Applications with the period of $2ms$
$5ms$	Applications with the period of $5ms$
$6ms$	Applications with the period of $2ms$
$8ms$	Applications with the period of $2ms$
$10ms$	**Repeat actions at** $0ms$

4. COMPUTATION-EFFICIENT DESIGN

OSEK/VDX-compliant Operating Systems (OS), with preemptive fixed-priority scheduling, are widely used in the automotive domain [6, 7]. With such an OS, once each application gets released, it is allowed to access the processor periodically. There are various predefined periods of release times and each application is assigned one. Different applications may have different periods. Every time an application is released, its program gets the chance to be executed, depending on its priority.

Here, a time table containing all the periodic release times within the alleged hyperperiod (i.e., the minimum common multiple of all periods) of the applications needs to be configured. An example with a set of three periods $2ms$, $5ms$ and $10ms$ is illustrated in Table 1. The hyperperiod is equal to $10ms$ and the time table repeats itself every $10ms$ by resetting the timer.

Generally for a feedback control application, a shorter sampling period allows the controller to respond to its plant more frequently, and is thus potentially able to achieve better control performance. The obvious downside is a higher processor load, since the control program is executed more frequently. Let us assume that the set of available periods restricted by an OSEK/VDX OS be ϕ. Considering a single-core processor p in an ECU, denoting E_i^{wc} to be the WCET of a control application C_i, if a uniform sampling period of h is used, the processor load for C_i is

$$L_i = \frac{E_i^{wc}}{h}. \tag{8}$$

The upper bound on the processor load is 1 and we have

$$\sum_{\{i|C_i \text{ runs on } p\}} L_i \leq 1. \tag{9}$$

Clearly, increasing the sampling period of a control application decreases its processor load, and thus potentially enables more applications to be integrated on the ECU, thereby resulting in a more cost-efficient system.

Since an OSEK/VDX OS only offers a limited set of predefined sampling periods to the control applications at hand and often the optimal sampling period for a given control application is not directly realizable, the conventional way is to use the largest sampling period offered by the OS that is smaller than the optimal one. This clearly wastes computation resources.

A *computation-efficient controller*, in contrast, switches between multiple available sampling periods offered by the OSEK/VDX OS, thereby achieving an average sampling period closer to the optimal one and reducing processor load. Possible switchings with sampling periods of $2ms$, $5ms$ and $10ms$ on OSEK/VDX OS are illustrated in Figure 4. For one application, switching between two sampling periods can only occur at the common multiple of them. For

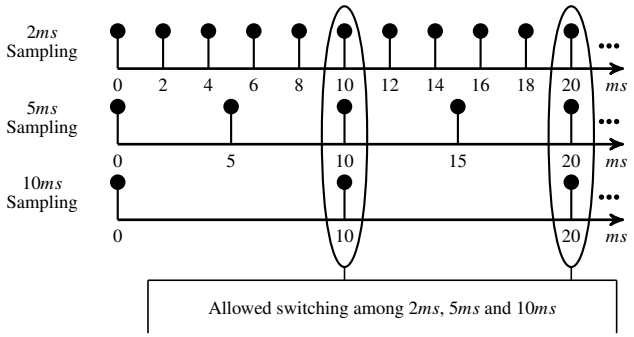

Figure 4: Allowed switching instants among multiple sampling periods

instance, switching between $2ms$ and $5ms$ is possible at the time instants of $10ms$, $20ms$, and so on. Therefore, possible sequences of sampling periods are $\{2ms, 2ms, 2ms, 2ms, 2ms, 5ms, 5ms, repeat\}$, $\{5ms, 5ms, 10ms, repeat\}$, and so on. The challenge is to design a performance-oriented controller that exploits this non-uniform sampling to satisfy the requirements on control performance.

We assume that the cyclic sequence of sampling periods for a control application defines a schedule S,

$$S = \{T_1, T_2, T_3, \ldots, T_N\}, \tag{10}$$

where $\forall j \in \{1, 2, \ldots, N\}$, $T_j \in \phi$. Dictated by the schedule S, this boils down to N systems (combination of plant and controller) switching cyclically in a deterministic fashion. The dynamics of these N systems within one cycle of S is given by

$$\begin{aligned} x[k+1] &= A_d(T_1)x[k] + B_d(T_1)u[k-1], \\ x[k+2] &= A_d(T_2)x[k+1] + B_d(T_2)u[k], \\ &\vdots \\ x[k+N] &= A_d(T_N)x[k+N-1] + B_d(T_N)u[k+N-1]. \end{aligned} \tag{11}$$

In order to avoid varying sensor-to-actuator delays, the actuation occurs at the end of a sampling period and the sensor-to-actuator delay is equal to one sampling period. Therefore, $x[k+1]$ depends on its previous state $x[k]$ and the control input $u[k-1]$, which is computed according to $x[k-1]$. We now introduce a new state $z[k] = \begin{bmatrix} x[k]^T & u[k-1] \end{bmatrix}^T$. Then, $\forall j \in \{1, 2, \ldots, N\}$,

$$\begin{aligned} z[k+j] = &\begin{bmatrix} A_d(T_j) & B_d(T_j) \\ \mathbf{0} & 0 \end{bmatrix} z[k+j-1] \\ &+ \begin{bmatrix} \mathbf{0} & 1 \end{bmatrix}^T u[k+j-1], \end{aligned} \tag{12}$$

where $\mathbf{0}$ is a zero vector of appropriate dimension. The system output is

$$y[k+j-1] = C_{aug} z[k+j-1], \tag{13}$$

where

$$C_{aug} = \begin{bmatrix} C & 0 \end{bmatrix}. \tag{14}$$

The control input is designed as

$$u[k+j-1] = K_j z[k+j-1] + F_j r. \tag{15}$$

Denoting A_{aug} and B_{aug} as

$$\begin{aligned} A_{aug}(T_j) &= \begin{bmatrix} A_d(T_j) & B_d(T_j) \\ \mathbf{0} & 0 \end{bmatrix} \\ B_{aug}(T_j) &= \begin{bmatrix} \mathbf{0} & 1 \end{bmatrix}^T, \end{aligned} \tag{16}$$

Table 2: Settling time and processor load of three schedules. Bold numbers indicate satisfied settling time requirement.

Schedule	Settling Time	Processor Load
$S1 = \{5ms\}$	$256.40ms$	14%
$S2 = \{2ms\}$	$\mathbf{113.27ms}$	35%
$S0$	$\mathbf{132.14ms}$	24.5%

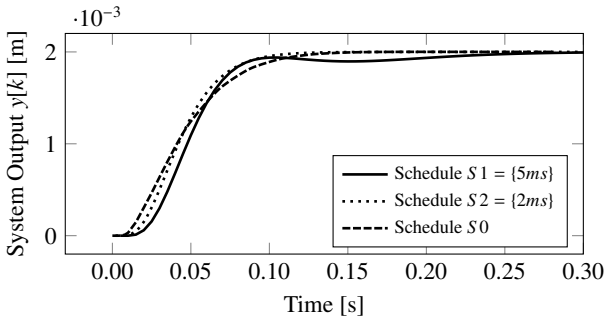

Figure 5: System output of three different schedules

The closed-loop dynamics is

$$\begin{aligned} z[k+j] &= A_{aug}(T_j)z[k+j-1] + B_{aug}(T_j)u[k] \\ &= (A_{aug}(T_j) + B_{aug}(T_j)K_j)z[k+j-1] \\ &\quad + B_{aug}(T_j)F_j r. \end{aligned} \tag{17}$$

We denote the closed-loop system matrix as

$$A_{cl,j} = A_{aug}(T_j) + B_{aug}(T_j)K_j. \tag{18}$$

The poles to be placed are the eigenvalues of $A_{cl,j}$. Note that (13), (15), (16), (17), and (18) are applied for every j in $\{1, 2, \ldots, N\}$. We now formulate an optimization problem for the pole-placement as

$$\begin{aligned} &\min_{\mathbb{D}} \xi, \quad \text{subject to} \\ &|u[k]| \le U_{max}, \quad \xi \le \xi^r, \end{aligned} \tag{19}$$

where the poles are decision variables and the settling time ξ is to be minimized as the objective. There are three constraints. First, the input saturation has to be respected, where U_{max} is the physical limit of the actuator. Second, the settling time requirement ξ^r has to be satisfied. Third, $\mathbb{D}$ is a domain of poles that ensure the system stability. As discussed in Section 2, such a constrained non-convex optimization problem with significant non-linearity can be solved by heuristics like PSO.

We now show some evaluation results of such a computation-efficient controller using an Electro-Mechanical Braking (EMB) system. The settling time requirement ξ^r is $150ms$. The set of available sampling periods offered by the OSEK/VDX OS in this case is

$$\begin{aligned} \phi = \{&1ms, 2ms, 5ms, 10ms, 20ms, 50ms, \\ &100ms, 200ms, 500ms\}. \end{aligned} \tag{20}$$

As shown in Table 2 and Figure 5, the schedule $S1 = \{5ms\}$ cannot meet the settling time requirement. The largest sampling period smaller than $5ms$ in ϕ is $2ms$. The schedule $S2 = \{2ms\}$ is able to fulfill all the requirements. According to (8), assuming that the WCET is $0.7ms$, the processor load of $S2$ is 35%. Clearly, this number can be unnecessarily large, preventing more applications from being packed into the same ECU. We then evaluate the schedule $S0 = \{2ms, 2ms, 2ms, 2ms, 2ms, 5ms, 5ms\}$ switching between $2ms$ and $5ms$. $S0$ has a slightly longer settling time than $S2$, yet

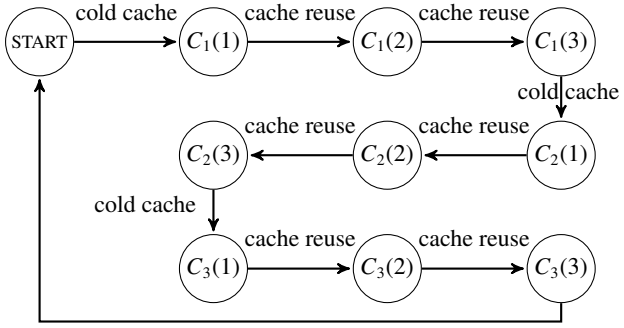

Figure 6: Memory analysis of an example with three control applications. Each application is consecutively executed three times. After the first execution $C_i(1)$, some instructions in the cache can be reused and thus the WCETs of the following two executions are shortened.

still fulfills the settling time requirement of $150ms$. Extending (8), the processor load is 24.5%, achieving a 30% reduction compared to $S2$, and now allowing more applications to be packed. While this is one among several possible techniques to achieve computationally efficient controllers, it illustrates the need to design controllers by taking into account the characteristics and constraints of the implementation platform, which also holds for other resource types like memory and communication.

5. MEMORY-EFFICIENT DESIGN

Memory and especially on-chip memory on ECUs substantially contributes to the ECU cost. In many automotive setups, the code for different control applications is stored in a bigger inexpensive flash memory. When a particular instruction is executed, it is fetched from the flash to the on-chip memory, if it is not in the on-chip memory yet. Further access to this instruction will be much faster, before it is replaced by other instructions. The smaller the on-chip memory is, the more cost-efficient is the ECU. However, the additional latency involved in fetching the code from the flash memory deteriorates the control performance. The question is, can the control algorithms be designed to mitigate such delays and exploit this memory hierarchy?

Given a collection of control applications (e.g., C_1, C_2, C_3), it is conventional to run the control loops of them in a round-robin fashion ($C_1, C_2, C_3, C_1, C_2, C_3, \cdots$). This frequently refreshes the ECU on-chip memory and the time it takes to fetch a code from the flash increases its WCET. In order to address this issue, again a non-uniform sampling scheme is helpful. Here, the control loop of each application is consecutively run multiple times — in order to increase the cache or on-chip memory reuse, before moving on to the next application. For example, ($C_1, C_1, C_1, C_2, C_2, C_2, C_3, C_3, C_3, \cdots$), can be used.

As shown in Figure 6, $C_i(j)$ denotes the jth execution of the control application C_i. Before the first execution $C_i(1)$, the cache is either empty (i.e., cold cache) or filled with instructions from other applications, that are not used by C_i (equivalent to cold cache). The WCET of $C_i(1)$ can be computed by a number of existing standard techniques [8, 9, 10]. Before the second execution $C_i(2)$, the instructions in the cache are from the same application C_i and thus can be reused. This results in more cache hits and hence shorter WCET. Depending on which path the program takes, the amount of WCET reduction varies.

After WCET results are computed, the next task is to derive the control timing parameters (e.g., sampling periods and sensor-to-actuator delays). In particular, we explore the relationship between WCET results and control timing parameters of two example sampling schemes. As illustrated in Figure 7, S1 is the conventional memory-oblivious scheme and summarized as follows,

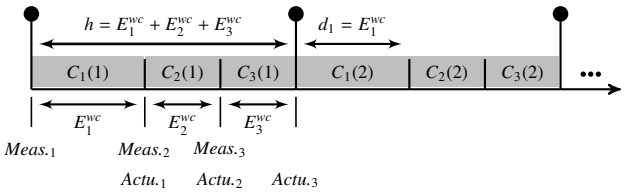

Figure 7: In S1, there is no cache reuse. The WCET of all executions for the same application E_i^{wc} remains constant. The sampling period of every control application h is *uniform* under this scheme. The sensor-to-actuator delay d_i is equal to E_i^{wc}.

$$S1 : C_1(1) \rightarrow C_2(1) \rightarrow C_3(1) \rightarrow C_1(2) \rightarrow C_2(2) \rightarrow C_3(2) \rightarrow C_1(3) \rightarrow C_2(3) \rightarrow C_3(3) \rightarrow \cdots . \tag{21}$$

There is no *cache reuse* in S1 as discussed above, considering that different control applications typically have different instructions to execute. In other words, when $C_i(j)$ starts execution, all instructions of C_i need to be brought into the cache from the flash memory. Therefore,

$$E_i^{wc}(1) = E_i^{wc}(2) = \cdots = E_i^{wc}, \tag{22}$$

where $E_i^{wc}(j)$ is the WCET of the jth execution for C_i. The WCET of the application C_i is denoted by E_i^{wc}, since all executions of the same application have equal WCET. Clearly, all control applications run with a uniform sampling period of

$$h = \sum_{i=1,2,3} E_i^{wc}. \tag{23}$$

Moreover, for the sensor-to-actuator delay,

$$d_i = E_i^{wc}. \tag{24}$$

As has been shown in Figure 6, S2 is an example memory-aware sampling order and summarized as,

$$S2 : C_1(1) \rightarrow C_1(2) \rightarrow C_1(3) \rightarrow C_2(1) \rightarrow C_2(2) \rightarrow C_2(3) \rightarrow C_3(1) \rightarrow C_3(2) \rightarrow C_3(3) \rightarrow \cdots . \tag{25}$$

As illustrated in Figure 8, we denote the effective WCET taking into account the cache reuse with $\bar{E}_i^{wc}(j)$. From the above discussion,

$$\forall i \in \{1, 2, 3\}, \quad \bar{E}_i^{wc}(1) = E_i^{wc}, \tag{26}$$

since there is no cache reuse for the first execution of every application $C_i(1)$. $\bar{E}_i^{wc}(2)$ and $\bar{E}_i^{wc}(3)$ are shorter than $\bar{E}_i^{wc}(1)$ due to cache reuse. The amounts of cache reuse are the same for $C_i(2)$ and $C_i(3)$ in the worst case. Denoting the guaranteed WCET reduction irrespective of the execution path as $\bar{E}_i^g$, we have
$\forall i \in \{1, 2, 3\}$,

$$\bar{E}_i^{wc}(2) = \bar{E}_i^{wc}(3) = \bar{E}_i^{wc}(1) - \bar{E}_i^g. \tag{27}$$

From these varying WCETs, the sampling periods of all three applications can be calculated. Taking C_1 as an example, there are three sampling periods $h_1(1)$, $h_1(2)$ and $h_1(3)$, which repeat themselves periodically,

$$h_1(1) = \bar{E}_1^{wc}(1), \quad h_1(2) = \bar{E}_1^{wc}(2), \quad h_1(3) = \bar{E}_1^{wc}(3) + \Delta, \tag{28}$$

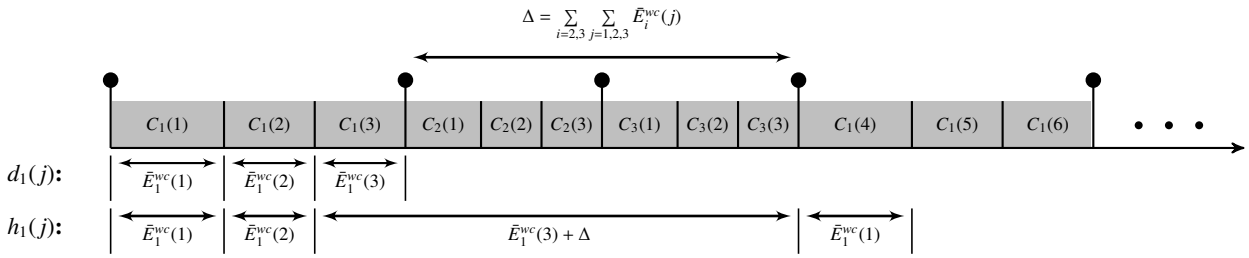

Figure 8: In S2, the WCETs of the same control application vary, due to cache reuse. The sampling period for a control application is *non-uniform*.

where Δ is computed as

$$\Delta = \sum_{i=2,3} \sum_{j=1,2,3} \bar{E}_i^{wc}(j). \tag{29}$$

Similar derivation can be done for C_2 and C_3. The average sampling period of an application h_{avg} is

$$h_{\text{avg}} = \frac{\sum_{i=1,2,3} \sum_{j=1,2,3} \bar{E}_i^{wc}(j)}{3} < h. \tag{30}$$

According to (26) and (27),

$$h_{\text{avg}} < \frac{\sum_{i=1,2,3} 3 \times E_i^{wc}}{3}. \tag{31}$$

From (23), we get

$$h_{\text{avg}} < h. \tag{32}$$

Moreover, the corresponding sensor-to-actuator delay $d_i(j)$ also varies with cache reuse as
$\forall i \in \{1, 2, 3\}$,

$$\begin{aligned} d_i(1) &= h_i(1) = \bar{E}_i^{wc}(1) \\ d_i(2) &= h_i(2) = \bar{E}_i^{wc}(2) \\ d_i(3) &= \bar{E}_i^{wc}(3). \end{aligned} \tag{33}$$

As all control parameters have been derived, we can see that the sampling period $h_i(j)$ of a control application is *non-uniform* for the memory-aware scheme. The *average* sampling period of S2 is shorter than the uniform sampling period of S1 as shown in (30), due to WCET reduction resulting from cache reuse. The sensor-to-actuator delay $d_i(j)$ varies as shown in (33). Now, a controller design technique similar to the one presented in Section 4 may be used to exploit the shortened non-uniform sampling periods and achieve better control performance.

6. CONCLUDING REMARKS AND FUTURE DIRECTIONS

The model-based design of resource-efficient automotive control software is, of course, a relatively new and open research field and the state-of-the-art approaches are certainly not limited to the ones shown in this paper. Moreover, it is possible to explore other research directions in this context as discussed in the following.

Task partitioning and mapping, frame packing: Most of the current resource-efficient control software design methods [4, 11] consider the task partition and mapping as given by the specification. However, in such a case the co-optimization is not comprehensive, i.e., there may exist some other task partitions or mappings that result in better system performance or resource utilization. Therefore, when synthesizing systems, it is desirable to integrate task partition and mapping with computation of schedules and controller parameters in a holistic framework. Task partition and mapping are well studied [12] and can be formulated as an optimization problem. However, when they are integrated into the resource-efficient control software design, additional dimensions are introduced into the mathematical model and the combined model can easily become intractable for large system size.

In the same vein, we have assumed in the example of Section 3 that each message is packed into one frame. However, this assumption results in wastage of communication bandwidth. This is because the slot length is decided based on the size of the longest message, and therefore shorter messages do not use the complete slot allocated to them and a fraction of such a slot remains unused. It is possible to pack a number of messages into one frame that uses only one slot. This frame packing problem is studied in [13] and can be incorporated in the resource-efficient control software design phase for better results.

Other scheduling schemes and bus systems: In Section 3, we consider Time-Triggered Architectures (TTAs) because of its inherent determinism. However, TTAs often lead to inefficient resource utilization that may not be sustainable in cost-sensitive automotive systems. Therefore, there is a need to consider Event-Triggered Architectures (ETAs) for improving resource utilization without compromising a lot on control performance. However, it must be noted that in case of ETAs, it is required to compute parameters like task and message priorities, arbitration policy, etc. instead of task and message schedules. Moreover, due to non-determinism in ETAs, there is a huge variation in the closed-loop delay, and therefore, we may need to incorporate in the control software design problem some timing analysis [14, 15, 16] to obtain the worst-case closed-loop delay. Furthermore, we need to design controller that can handle non-determinism like flexible delays, timing violations, packet drops, etc.

Similarly, we can also exploit the fact that a control law need not be executed at high frequency if the controlled plant is in equilibrium [17]. Therefore, based on the system states we can allocate resource to the respective controller. We can also map the controller in event-triggered domain when the corresponding plant is stable and switch to time-triggered when disturbed [18]. However, for such considerations we need to design separate controllers for different states, e.g., the stable and the disturbed states, and also need to ensure that the switching between controllers is stable. Moreover, we may also decide on a strategy to allocate time-triggered resources to disturbed control loops in case of multiple control applications sharing the same resource.

Heterogeneous networks and gateways: Modern embedded systems like automotive E/E architectures typically consist of different functional domains. Depending on the requirement in each domain, different bus systems are used, like FlexRay for chassis, high-speed CAN or FlexRay for powertrain, low-speed CAN and LIN for body, MOST and Ethernet for infotainment. The different bus clusters in

an automotive system are connected via gateways for inter-domain communication. With the newly developed applications for driver-assistance systems and autonomous driving, increasingly more applications require inter-domain interaction. However, considering this is not straightforward since requirements of different applications of different criticality (e.g., hard real-time, soft real-time, control, non-real-time, etc.) have different characteristics and it is difficult to integrate them as constraints into a single framework. For example, hard real-time applications have stringent timing requirements while control applications must satisfy stability and performance constraints. Moreover, different bus protocols like CAN, FlexRay, etc. require different timing models and analysis techniques. Furthermore, for inter-domain communication the gateways need to be characterized using correct timing models. There are very few works [19] on embedded system design which consider heterogeneous networks. [19] has proposed a hybrid analysis technique which allows results from different analysis techniques to be composed together. As an extension, resource-efficient control software design based on such hybrid analysis techniques can be considered.

Computation-efficient design in multi-core systems: While Section 4 considers only single-core processor architecture, it can be extended to multi-core architectures. There are mainly two challenges to address in this case. First, due to load balancing requirements, it might be necessary to distribute different parts of complex control applications to different cores. This introduces additional delays for sensor-to-actuator cause-effect chains that need to be taken into account during controller design to ensure stability. Second, memory partitioning and code placement need to be considered, since they have a major influence on the execution times of control programs.

Optimal memory-efficient scheduling: In Section 5, we show an example memory-efficient schedule, which could have considerable influence on control system performance. The next problem to solve is finding the optimal schedule that maximizes the overall control performance. It consists of two stages. First, given a schedule, we need to find the optimal poles that maximize the overall control performance while the input saturation constraint is respected. This is a challenging problem to solve, due to the non-convexity, non-linearity and many-dimensional decision space. Second, based on the results from the first stage, the optimal schedule needs to be found. As the number of applications grows, the number of possible periodic schedules increases exponentially. Considering that the control performance evaluation of one schedule is computationally heavy, brute force is practically infeasible. Heuristic methods need to be developed that are able to achieve a flexible balance between optimality and scalability.

Conclusion: Automotive systems are highly cost-sensitive and thus, resource-efficient design of automotive control software is an important research direction. Towards this direction, in this paper, we have discussed three concrete examples of state-of-the-art design approaches, targeting respectively at communication-efficient design, memory-efficient design and computation-efficient design, to illustrate the basic idea. In Section 3, a co-design approach is described where both the control law and the controller implementation are determined in a holistic framework, thereby, co-optimizing the control performance and the communication bandwidth usage. Section 4 shows how a multirate controller can be used to reduce the processor load of a control application, while still satisfying the control performance requirement. Section 5 suggests a non-uniform sampling schedule to exploit cache reuse, thereby improving the control performance for a given memory resource.

7. REFERENCES

[1] K. Astrom and B. Wittenmark. *Computer-Controlled Systems: Theory and Design, 3rd Edition*. Prentice Hall, 1997.

[2] D. Goswami, R. Schneider and S. Chakraborty. Relaxing signal delay constraints in distributed embedded controllers. *IEEE Transaction on Control Systems Technology*, 22(6):2337-2345, 2014.

[3] D. Sedighizadeh and E. Masehian. Particle swarm optimization methods, taxonomy and applications. *International Journal of Computer Theory and Engineering*, 1(4):486-502, 2009.

[4] D. Roy, L. Zhang, W. Chang, D. Goswami and S. Chakraborty. Multi-objective co-optimization of FlexRay-based distributed control systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2016.

[5] The FlexRay communications system protocol specification, Version 3.0.1 [Online].

[6] OSEK/VDX Consortium, OSEK/VDX operating system specification, Version 2.2.3, 2005.

[7] P. Feiler. Real-time application development with OSEK: A review of the OSEK standards. Carnegie Mellon University, 2003.

[8] R. Wilhelm et al. The worst-case execution-time problem — overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3), pp. 36:1-53, 2008.

[9] S. Andalam, A. Girault, R. Sinha, P. Roop and J. Reineke. Precise timing analysis for direct-mapped caches. In *ACM/IEEE Design Automation Conference*, 2013.

[10] R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister and C. Ferdinand. Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(7):966-978, 2009.

[11] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty. Time-triggered implementations of mixed-criticality automtoive software. In *Design, Automation, and Test in Europe Conference and Exhibition*, 2012.

[12] V. M. Lo. Heuristic algorithms for task assignment in distributed systems. *IEEE Transactions on Computers*, 37(11):1384-1397, 1988.

[13] M. Kang, K. Park and M. Jeong. Frame packing for minimizing the bandwidth consumption of the FlexRay static segment. *IEEE Transactions on Industrial Electronics*, 60(9):4001-4008, 2013.

[14] R. Davis, A. Burns, R. Bril and J. Lukkien. Controller Area Network (CAN) schedulability analysis: refuted, revisited and revised. *Real-Time Systems*, 35(3):239-272, 2007.

[15] R. Schneider, L. Zhang, D. Goswami, A. Masrur and S. Chakraborty. Compositional analysis of switched ethernet topologies. In *Design, Automation and Test in Europe Conference and Exhibition*, 2013.

[16] F. Reimann, S. Graf, F. Streit, M. Glaß and J. Teich. Timing analysis of Etherent AVB-based automotive E/E architectures. In *IEEE Conference on Emerging Technologies and Factory Automation*, 2013.

[17] P. Marti, J. M. Fuertes, G. Fohler and K. Ramamritham. Improving quality-of-control using flexible timing constraints: metric and scheduling. In *IEEE Real-Time Systems Symposium*, 2002.

[18] D. Goswami, R. Schneider and S. Chakraborty. Reengineering cyber-physical control applications for hybrid communication protocols. In *Design, Automation and Test in Europe Conference and Exhibition*, 2011.

[19] M. Glaß, M. Lukasiewycz, J. Teich, U. Bordoloi and S. Chakraborty. Designing heterogeneous ECU networks via compact architecture encoding and hybrid timing analysis. In *ACM/IEEE Design Automation Conference*, 2009.