Cache-Aware Task Scheduling for Maximizing Control Performance

Wanli Chang*, Debayan Roy†, Xiaobo Sharon Hu‡ and Samarjit Chakraborty†
*Infocomm Technology Cluster, Singapore Institute of Technology
†Chair of Real-Time Computer Systems, TU Munich
‡Department of Computer Science and Engineering, University of Notre Dame

***Abstract*—Embedded control applications are widely implemented on small, low-cost and resource-constrained microcontrollers, e.g., in the automotive domain. Conventionally, control algorithms are designed using model-based approaches, without considering the details of the implementation platform. This leads to inefficient utilization of the resources. With the emergence of the cyber-physical system (CPS)-oriented thinking, there has lately been a strong interest in co-design of control algorithms and their implementation platforms. Some recent efforts have shown that a schedule on multiple applications with more on-chip cache reuse is able to improve the control performance. However, it has not been studied how the control performance can be maximized for a given schedule and how an optimal schedule can be computed. In this work, we propose a two-stage framework to compute the schedule maximizing the overall control performance of all the applications. First, a holistic controller design taking all the sampling periods and sensing-to-actuation delays in a schedule into account is presented, aiming to maximize the overall control performance. Second, a hybrid search algorithm for discrete decision space is reported to efficiently compute an optimal schedule. Experimental results on a case study with multiple automotive applications show that a significant improvement of 10-20% in control performance can be achieved by the proposed cache-aware scheduling approach.**

I. Introduction

Embedded control applications are mainly implemented on microcontrollers with limited computation, communication, and memory resources. Traditionally, control system design and implementation were strictly separated, the former being pursued by control theorists and the latter by embedded system engineers. Though this design paradigm is able to achieve the required control performance, it often leads to inefficient utilization of the resources. With the emergence of the cyber-physical system (CPS)-oriented thinking, there has lately been a strong interest in co-design of control algorithms and their implementation platforms, motivating works on communication-aware design of networked control systems [1], [2], [3] and computation-aware embedded control system design [4], [5], [6]. Many of the papers consider the characteristics of the communication and computation resources while tackling the scheduling problem of embedded control systems.

An embedded implementation platform is often shared by multiple control applications. Each application is realized by a sequence of (repeated) tasks. For feedback control applications considered in this work, each task completes a control loop within one sampling period, which is counted from the starting time instant of one task to the starting time instant of the next task belonging to the same application. Task scheduling to improve cache reuse [7], [8], with the aim of minimizing worst-case execution time (WCET), has been widely studied in the literature. Some recent efforts have shown that a schedule with more on-chip cache reuse is able to improve the control performance [9]. However, it has not been investigated how the control performance can be maximized for a given schedule and how an optimal schedule can be computed.

In this work, we perform task scheduling and control performance optimization by judiciously reuse cache. In particular, we aim to compute a task schedule that maximizes the overall control performance of all given control applications. There are two main challenges. First, for a given schedule, the overall control performance depends on the controller design. We consider non-uniform sampling (i.e., tasks of one control application may have varying sampling periods) and propose a holistic method taking all the sampling periods and sensing-to-actuation delays in a schedule into account. The control performance is measured by settling time, which is the key metric for many real-time control applications and more difficult to optimize than quadratic cost. System constraints (e.g., input saturation) need to be respected. Second, the number of periodic schedules under consideration grows exponentially with the number of applications. The control performance evaluation of each schedule is computationally intensive. We introduce a hybrid search algorithm for discrete decision space to compute an optimal schedule efficiently. It is based on gradient descent and equipped with features of simulated annealing.

The rest of the paper is organized as follows. Fundamentals of the cache-aware embedded control system design are described in Section II. The controller design method to maximize the control performance of a given schedule is presented in Section III. In Section IV, the hybrid search algorithm to find an optimal schedule is reported. Experimental results are shown in Section V and Section VI makes concluding remarks.

II. Fundamentals of Cache-Aware Embedded Control System Design

In this work, we consider periodic schedules of n feedback control applications $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n\}$ running on the microcontroller with a single processor, an on-chip cache, and a flash memory. The cache size is assumed to be smaller than the size of a control program, since our focus is on embedded control systems with limited hardware resources.

This work was partially supported by Deutsche Forschungsgemeinschaft (DFG) through the TUM International Graduate School of Science and Engineering (IGSSE).

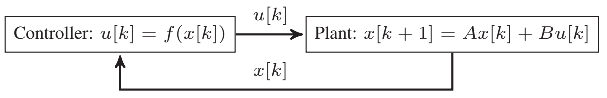


Fig. 1. A discrete-time feedback control system.

For any application $\mathcal{C}_i$, where $i \in \{1, 2, \ldots, n\}$, the number of consecutively executed tasks in one schedule period is denoted by m_i. Then, we have the periodically repeating schedule denoted by $(m_1, m_2, \ldots, m_n)$. Interleaved schedules will be briefly discussed at the end of this paper and left for future work. Compared with the conventional round-robin schedule $(1, 1, \ldots, 1)$, consecutively executing tasks of one application increases the cache reuse and reduces the WCET, yet resulting in non-uniform sampling periods. This will be exploited by the controller design presented in the next section to achieve better control performance.

This paper focuses on instruction cache, since majority of control algorithms utilize the freshest sensor reading for computing the control inputs and do not consider obsolete sensor values. This further implies that such algorithms require little data memory and their control performances are mostly driven by the instruction cache. In this section, we describe fundamentals of cache-aware embedded control system design, including basics of the discrete-time feedback control system under consideration, cache analysis, and control timing parameter derivation.

A. Basics of discrete-time feedback control systems

Discrete-time control systems: In this work, we consider discrete-time linear time-invariant (LTI) single-input single-output (SISO) feedback control applications. Majority of the applications in practice are modeled as LTI systems and many nonlinear applications are linearized. The approach proposed in this paper can be easily adapted for multiple-input multiple-output (MIMO) applications. The system dynamics of an application is described as follows,

$$x[k+1] = Ax[k] + Bu[k], \quad y[k] = Cx[k], \tag{1}$$

where $x[k] \in \mathbb{R}^l$ and $y[k]$ are the *system state* and the *system output* respectively at the time instant t_k. The *control input* computed based on $x[k]$ (state-feedback control) is denoted as $u[k]$. The number of system states is l. We assume that the system state $x[k]$ is measurable. The system output $y[k]$ is expected to track a reference r. Sampling instants are $t = t_k$ ($k = 1, 2, 3, \cdots$) and the sampling period h is $t_{k+1} - t_k$. It should be noted that h might not be a constant. A, B and C are constant matrices of appropriate dimensions depending on the application characteristics and the sampling period. The relationship between the controller and the plant is illustrated in Figure 1.

Overall control performance: The control performance can be quantified by various metrics. In this work, we consider *settling time* as the performance index, which is the key metric for many real-time control applications, such as the electric motor control, steering control and braking control in automobiles [10]. The control goal is to make $y[k] \to r$ as soon as possible. The time it takes for $y[k]$ to reach and stay

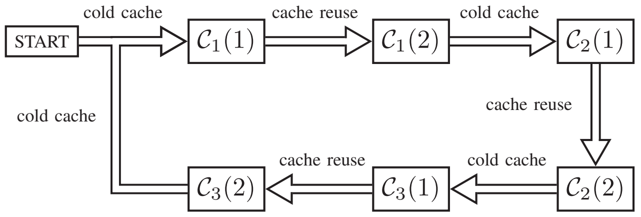


Fig. 2. Cache analysis of an example schedule $(2, 2, 2)$. After the first task $\mathcal{C}_i(1)$ is executed, some instructions in the cache can be reused and thus the WCET of the following task is shortened.

in a closed region around r (e.g., $0.98r$ to $1.02r$) is the settling time. Shorter settling time implies better control performance. For an application, we consider the worst-case settling time by assuming that the reference tracking starts after its last consecutive task in a schedule.

The overall control performance is defined as a weighted sum of application control performances, which are normalized to be comparable. Assuming that the settling time of an application $\mathcal{C}_i$ is s_i and the normalization reference is s_i^0, then the control performance is defined to be $1 - \frac{s_i}{s_i^0}$. Since both s_i and s_i^0 are positive numbers, the control performance is less than 1. The overall control performance can be calculated as

$$P_{\text{all}} = \sum_{i=1}^{n} w_i P_i = \sum_{i=1}^{n} w_i \left(1 - \frac{s_i}{s_i^0}\right), \tag{2}$$

where w_i is the weight of the application $\mathcal{C}_i$ and the sum of weights is 1.

Constraints: We consider four constraints in this work. First, all control systems must be stable. Second, in almost every real-world system, there is a maximum available control input. The controller needs to be designed such that the maximum value of $u[k]$ does not exceed this limit U_{max}, i.e., $u[k] \le U_{\text{max}}$. For example, in electric motor control, the magnitude of the input current is always limited. Third, when an application $\mathcal{C}_i$ is safety-critical, there is usually a maximum settling time (i.e., settling deadline) s_i^{max} that cannot be violated [11]. This is the reason why the worst-case settling time is considered as discussed above. We use this deadline as the normalization reference, i.e., $s_i^0 = s_i^{\text{max}}$. The constraint is then, $\forall i \in \{1, 2, \ldots, n\}$,

$$P_i \ge 0. \tag{3}$$

Fourth, for an application $\mathcal{C}_i$, there is a maximum allowed idle time t_i^{idle} to prevent the application from being driven to an unsafe state by perturbations. The idle time is defined as the interval between two consecutive sampling instants and thus equal to the sampling period. Denoting h_i^{max} to be the longest sampling period of $\mathcal{C}_i$ in a schedule, we must have $\forall i \in \{1, 2, \ldots, n\}$,

$$h_i^{\text{max}} \le t_i^{\text{idle}}. \tag{4}$$

B. Cache analysis

An example schedule $(2, 2, 2)$ with three applications for cache analysis is illustrated in Figure 2, where each application $\mathcal{C}_i$ ($i \in \{1, 2, 3\}$) consecutively executes two tasks in one schedule period and $\mathcal{C}_i(j)$ ($j \in \{1, 2\}$) denotes the jth task.

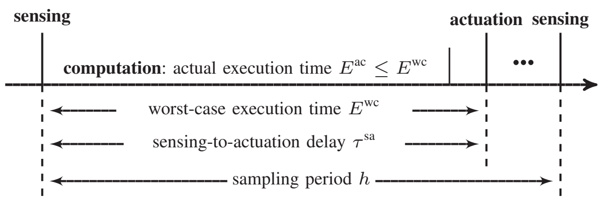

sensing
actuation
sensing
computation: actual execution time $E^{ac} \leq E^{wc}$
•••
worst-case execution time E^{wc}
sensing-to-actuation delay τ^{sa}
sampling period h

Fig. 3. The general timing model of a control loop.

Before the first task $\mathcal{C}_i(1)$ is executed, the cache is either empty (i.e., cold cache) or filled with instructions from other applications, that are not used by $\mathcal{C}_i$ (equivalent to cold cache). The WCET of $\mathcal{C}_i(1)$ can be computed by existing standard techniques [12]. Before the second task $\mathcal{C}_i(2)$ is executed, the instructions in the cache are from the same application $\mathcal{C}_i$ and thus can be reused. This results in more cache hits and hence shorter WCET. The reduction in WCET depends on the execution path. The guaranteed WCET reduction of $\mathcal{C}_i(2)$ can be computed using program analysis techniques, such as those in [13]. The effective WCET of $\mathcal{C}_i(2)$ can then be calculated by subtracting this guaranteed reduction due to cache reuse from the WCET considering cold cache. It is noted that the branches within the application have been taken into account by the above WCET analysis. We consider no branches between applications, which is the usual case in practice.

C. Control timing parameters

The state-feedback control loop completed by a control task performs three operations: sensing (measuring the system states $x[k]$ with sensors), computation (computing the control input $u[k]$ based on $x[k]$), and actuation (applying $u[k]$ to the plant). The general timing model is illustrated in Figure 3, assuming that sensing and actuation operations are done instantaneously. The computation operation executes the control program, which takes E^{ac} time units. The sampling period h is the time duration between two consecutive sensing operations. The time interval between the sensing and the corresponding actuation operations in the same sampling period is the sensing-to-actuation delay τ^{sa}, which is equal to the control program WCET E^{wc}.

The relationship between WCETs and control timing parameters (sampling periods and sensing-to-actuation delays) in the schedule $(2, 2, 2)$ is illustrated in Figure 4. Denoting $E_i^{\text{wc}}(j)$ to be the WCET of the jth task for $\mathcal{C}_i$ in a schedule period and E_i^{gu} to be the guaranteed WCET reduction,
$\forall i \in \{1, 2, 3\}$,

$$E_i^{\text{wc}}(2) = E_i^{\text{wc}}(1) - E_i^{\text{gu}}. \tag{5}$$

From these varying WCETs, the sampling periods of all the three applications can be calculated. Taking $\mathcal{C}_1$ as an example, there are two sampling periods $h_1(1)$ and $h_1(2)$, which repeat themselves periodically,

$$h_1(1) = E_1^{\text{wc}}(1), \quad h_1(2) = E_1^{\text{wc}}(2) + \Delta, \tag{6}$$

where Δ is computed as

$$\Delta = \sum_{i=2,3} \sum_{j=1,2} E_i^{\text{wc}}(j). \tag{7}$$

Similar derivation can be done for $\mathcal{C}_2$ and $\mathcal{C}_3$. It can be seen that the sampling periods are constrained by WCETs of the control programs. Moreover, the corresponding sensing-to-actuation delay $\tau_i^{\text{sa}}(j)$ is
$\forall i \in \{1, 2, 3\}$,

$$\tau_i^{\text{sa}}(1) = h_i(1) = E_i^{\text{wc}}(1), \quad \tau_i^{\text{sa}}(2) = E_i^{\text{wc}}(2). \tag{8}$$

With the fundamentals of cache-aware embedded control system design explained, we can now proceed to discuss task scheduling for maximizing control performance involving two stages. First, the overall control performance for a given schedule is maximized with a holistic controller design taking all the sampling periods and sensing-to-actuation delays in the schedule into account. Second, an optimal schedule is found with the hybrid search algorithm.

III. Controller Design for Control Performance Maximization of a Given Schedule

This section presents the controller design to maximize the control performance of a given schedule. The example schedule $(2, 2, 2)$ is used for illustration. Generalization to any periodic schedule as defined at the beginning of Section II is straightforward. Given a schedule, it is assumed that the controller designs of different applications are independent. Therefore, we first maximize the control performance of each application running on the microcontroller and then obtain the maximum overall control performance with the weighted sum as per (2).

In a state-feedback controller, we need to design $u[k]$ utilizing the system state $x[k]$. The general structure is,

$$u[k] = K \cdot x[k] + F \cdot r, \tag{9}$$

where K is the feedback gain and F is the static feedforward gain. With this feedback controller, the closed-loop system dynamics is derived using (1) as

$$x[k+1] = (A + BK)x[k] + BFr = A_{cl}x[k] + BFr, \tag{10}$$

where A_{cl} is the closed-loop system matrix. Different locations of closed-loop poles, i.e., eigenvalues of A_{cl}, result in different system behaviors and corresponding control performances. In *pole-placement*, we place poles in desired locations (set eigenvalues) to optimize the control performance while respecting the constraint on the control input. It is noted that we assume the system in (1) is controllable, which is often the case. All the poles must have absolute values of less than unity in order to ensure stability. In this work, we use the particle swarm optimization (PSO) technique for pole-placement [14]. Details are omitted due to the page limit. The feedback gain K can be calculated according to the pole locations based on Ackermann's formula [15]. The static feedforward gain F is designed to achieve $y[k] \to r$ and computed by

$$F = \frac{1}{C(\mathbf{I} - A - BK)^{-1}B}, \tag{11}$$

where $\mathbf{I}$ is the identity matrix of appropriate dimension.

In order to maximize the control performance for an application $\mathcal{C}_i$, we propose a holistic method that designs controllers for all the control inputs in a schedule period together while

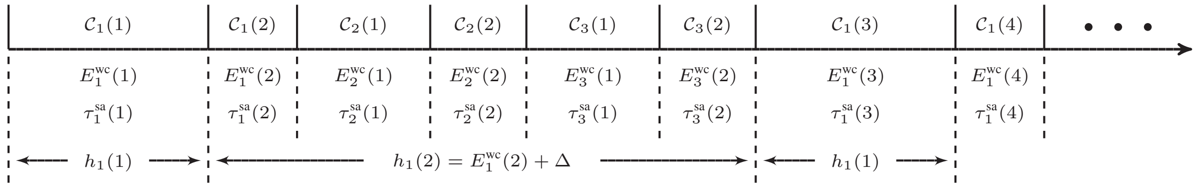
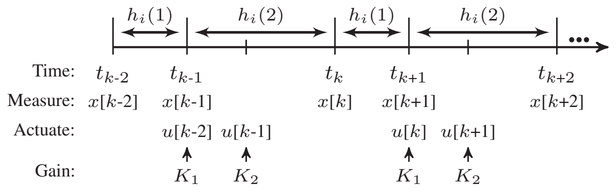


Fig. 4. In the example schedule $(2, 2, 2)$, the times of two consecutive executions for the same control application vary, due to cache reuse. The sampling period for a control application is *non-uniform*. The sensing-to-actuation delay τ_i^{sa} is equal to E_i^{wc}.



$h_i(1)$
$h_i(2)$
$h_i(1)$
$h_i(2)$
Time: $t_{k\text{-}2}$ $t_{k\text{-}1}$ t_k t_{k+1} t_{k+2}
Measure: $x[k\text{-}2]$ $x[k\text{-}1]$ $x[k]$ $x[k\text{+}1]$ $x[k\text{+}2]$
Actuate: $u[k\text{-}2]$ $u[k\text{-}1]$ $u[k]$ $u[k\text{+}1]$
Gain: K_1 K_2 K_1 K_2

Fig. 5. Periodically switched sampling periods for $\mathcal{C}_i$ in the schedule $(2, 2, 2)$.

considering all the sampling periods and sensing-to-actuation delays. The schedule $(2, 2, 2)$ is used for illustration. As shown in Figure 5, there are two sampling periods $h_i(1)$ and $h_i(2)$, which are repeated periodically. The system dynamics switches between the followings,

$$\begin{aligned} x[k] &= A_2 x[k-1] + B_2^1 u[k-2] + B_2^2 u[k-1], \\ x[k+1] &= A_1 x[k] + B_1 u[k-1], \end{aligned} \tag{12}$$

where A_1 and B_1 depend on the first sampling period $h_i(1)$. A_2, B_2^1 and B_2^2 depend on the second sampling period $h_i(2)$ and the sensing-to-actuation delay of the second task $\tau_i^{\text{sa}}(2)$. The system output is $y[k] = Cx[k]$. It should be noted that $x[k]$ is influenced by not only $u[k-2]$ but also $u[k-1]$, since $\tau_i^{\text{sa}}(2)$ is smaller than $h_i(2)$, i.e., $u[k-1]$ is applied before the sensing of $x[k]$. Two control inputs are designed as

$$\begin{aligned} u[k-2] &= K_1 x[k-2] + F_1 r, \\ u[k-1] &= K_2 x[k-1] + F_2 r. \end{aligned} \tag{13}$$

The feedback signals used by $u[k-2]$ and $u[k-1]$ are $x[k-2]$ and $x[k-1]$, respectively. The closed-loop system dynamics are then

$$\begin{aligned} x[k] = {} & (A_2 + B_2^2 K_2) x[k-1] + B_2^1 K_1 x[k-2] \\ & + (B_2^1 F_1 + B_2^2 F_2) r, \end{aligned} \tag{14}$$

and

$$\begin{aligned} x[k+1] &= A_1 x[k] + B_1 K_2 x[k-1] + B_1 F_2 r \\ &= (A_1 A_2 + A_1 B_2^2 K_2) x[k-1] \\ &\quad + A_1 B_2^1 K_1 x[k-2] \\ &\quad + (A_1 B_2^1 F_1 + A_1 B_2^2 F_2 + B_1 F_2) r. \end{aligned} \tag{15}$$

Introducing a new state $z[k] = \begin{bmatrix} x[k] & x[k+1] \end{bmatrix}^T$, we can obtain

$$\begin{aligned} z[k] = {} & A^{\text{hol}} z[k-2] \\ & + \begin{bmatrix} B_2^1 & B_2^2 \\ A_1 B_2^1 & A_1 B_2^2 + B_1 \end{bmatrix} \begin{bmatrix} F_1 & F_2 \end{bmatrix}^T r, \end{aligned} \tag{16}$$

where

$$\begin{aligned} A^{\text{hol}} &= \begin{bmatrix} B_2^1 K_1 & A_2 + B_2^2 K_2 \\ A_1 B_2^1 K_1 & A_1 A_2 + A_1 B_2^2 K_2 \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{0} & A_2 \\ \mathbf{0} & A_1 A_2 \end{bmatrix} + \begin{bmatrix} B_2^1 & B_2^2 \\ A_2 B_2^1 & A_2 B_2^2 \end{bmatrix} \begin{bmatrix} K_1 & \mathbf{0} \\ \mathbf{0} & K_2 \end{bmatrix}. \end{aligned}$$

The bold letter **0** is the zero matrix of appropriate dimension. A^{hol} is the overall closed-loop system matrix. It is noted that (16) is in a similar form to (10), where A^{hol} is the counterpart of A_{cl}. Therefore, the method explained at the beginning of this section can be used to place the poles and compute the gains. Ackermann's formula needs to be trivially extended to compute the feedback gains. The static feedforward gains are computed with (11) referring to their respective sampling periods,
$\forall j \in \{1, 2\}$,

$$F_j = \frac{1}{C(\mathbf{I} - A_j - B_j K_j)^{-1} B_j}, \tag{17}$$

where B_2 has not been mentioned yet and depends on the second sampling period $h_i(2)$.

With this controller design method, both control inputs in the schedule period are designed together taking all the information (all the sampling periods and sensing-to-actuation delays) into account. This is helpful for the control performance maximization. On the other hand, A^{hol} is a $2l \times 2l$ square matrix. The number of poles to place in A^{hol} is $2l$, or $m_i l$ in general. Therefore, evaluating the control performance of one schedule can be computationally intensive, especially when the number of consecutively executed tasks m_i is large. This makes an efficient search for an optimal schedule desirable.

IV. Computing an Optimal Schedule

After presenting the method to evaluate the overall control performance of one schedule, the next stage is to find an optimal one among all the schedules. That is, we need to determine $m_1, m_2, \ldots, m_n$ for n applications. The formulation is

$$\begin{aligned} & \max_{\{m_1, m_2, \ldots, m_n\}} P_{\text{all}} \\ & \text{subject to} \quad \{m_i \in \mathbb{N}^+ | i \in \{1, 2, \ldots, n\}\}. \end{aligned} \tag{18}$$

The objective to optimize is the overall control performance. The constraints on schedule feasibility are shown in (3) and (4). The number of dimensions in the decision space is equal to n. This is a nonlinear discrete optimization problem and the simplest method to solve it is exhaustive search. Denoting the number of values that m_i can take as $|m_i|$, the total number of schedules to evaluate is $\prod_{i=1}^{n} |m_i|$. Considering that the overall control performance evaluation can be computationally intensive, we need a more efficient method than brute force.

Gradient-based search algorithms, such as sequential quadratic programming (SQP), require a small number of objective function (the overall control performance in this work) evaluations. However, they are easily trapped by local optima. Simulated annealing is able to find the solution close to the global optimum, yet often needs to evaluate a large number of objective functions. In this work, we propose a hybrid search algorithm, which is based on SQP and takes the features from simulated annealing, in order to efficiently find a schedule close to the global optimum.

We first randomly initialize a point in the decision space. For SQP with a continuous decision space, an n-dimensional quadratic model on the point is built to derive the search direction with the steepest descent (for a minimization problem). However, when the decision space is discrete, it is unlikely that the computed direction is available. Besides, building the n-dimensional quadratic model requires evaluating the overall control performance $2n + \binom{n}{2}$ times, which is non-polynomial on the number of applications n. Therefore, we build a quadratic model for every dimension of the decision space and compute the gradient. The direction with the largest positive gradient is selected. The 1-dimensional quadratic model requires evaluating the overall control performance of two points on both sides of the current point. Since there are n quadratic models, the search direction determination takes $2n$ evaluations of overall control performance, at the maximum. If some overall control performance values have already been computed, this number can be smaller than $2n$. The step size is fixed to be 1. That is, the next point is always the closest neighbor to the current point, along the selected search direction. This process is iterated to locate one point after another, until no improvement on the objective value can be achieved. It is noted that feasibility must always be ensured. That is, if the next point along the direction with the best gradient violates the schedule feasibility constraints in (3) and (4), we will go for the second best direction and so on.

We implement two techniques to prevent this gradient-based search algorithm from being trapped by local optima. First, parallel searches can be conducted. As the number of initialized points is increased, the chance that the global optimum can be found rises. Second, we do not insist improvement on the objective value during the search process, which is similar to the simulated annealing. An appropriate tolerance threshold that can be empirically decided is likely to get rid of local optima and help the search algorithm reach the global optimum.

V. Experimental Results

In the experiment, we investigate an automotive control system case study with three applications $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$ running on a microcontroller with one processor and shared cache (Infineon XC23xxB Series). A schedule is denoted as (m_1, m_2, m_3). $\mathcal{C}_1$ is position control of a servo motor that can be used, e.g., in a steer-by-wire system [16]. $\mathcal{C}_2$ is speed control of a DC motor that can be used in electric vehicle

TABLE I
WCET results with and without cache reuse

Application	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$
WCET w/o Cache Reuse	$907.55\ \mu$s	$645.25\ \mu$s	$749.15\ \mu$s
Guaranteed WCET Reduction	$455.40\ \mu$s	$470.25\ \mu$s	$514.80\ \mu$s
WCET w/ Cache Reuse	$452.15\ \mu$s	$175.00\ \mu$s	$234.35\ \mu$s

TABLE II
Application parameters

Application	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$
Weight (w_i)	0.4	0.4	0.2
Settling deadline ($s_i^{\max}$)	45 ms	20 ms	17.5 ms
Maximum allowed idle time (t_i^{idle})	3.4 ms	3.9 ms	3.5 ms

TABLE III
Control performance comparison

Application	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$
Settling time for $(1, 1, 1)$	43.2 ms	17.7 ms	17.3 ms
Settling time for $(3, 2, 3)$	37.7 ms	15.3 ms	14.4 ms
Control performance improvement	13%	14%	17%

cruise control [17]. $\mathcal{C}_3$ is control of the electronic wedge brake system developed by Siemens as a brake-by-wire solution [18].

In the experimental configuration for the cache analysis, the processor clock frequency is 20 MHz. The cache is set to have 128 cache lines and each cache line is 16 bytes. When there is a cache hit, it takes 1 clock cycle to fetch the instruction and when there is a cache miss, it takes 100 clock cycles. The WCETs are calculated with the method discussed in Section II-B and reported in Table I. Control timing parameters of a given schedule can then be derived as explained in Section II-C. As described in Section II-A, the weights, settling deadlines and maximum allowed idle times of all the three applications are presented in Table II. The controller design presented in Section III is used to evaluate the overall control performance of one schedule.

The hybrid search algorithm presented in Section IV is deployed to find the optimal schedule. Two searches are run in parallel, starting from two randomly initialized schedules $(4, 2, 2)$ and $(1, 2, 1)$. Both reach the schedule $(3, 2, 3)$. The maximum overall control performance is 0.195. The optimal schedule $(3, 2, 3)$ is verified by the exhaustive search, which evaluates 76 schedules, including 74 feasible schedules. Two infeasible schedules violate the settling deadline constraint (3), which is known only after the control performance evaluation. Using the computer with an Intel i5 processor operating at 2.6 GHz with 4 GB RAM, evaluating the application control performance takes from seconds (when $m_i = 1$) to hours (when $m_i > 5$). Completing the exhaustive search of all the 76 schedules costs days. With our proposed hybrid search algorithm, the search starting from $(4, 2, 2)$ evaluates 9 schedules, which is 11.8% of the 76 schedules using brute force. The search starting from $(1, 2, 1)$ evaluates 18 schedules.

Comparison of the system output responses between the conventional cache-oblivious round-robin schedule $(1, 1, 1)$ and the optimal cache-aware schedule $(3, 2, 3)$ for all the three control applications is presented in Figure 6. Control performance comparison quantified by the settling times is reported in Table III. It can be seen that with the cache-aware task scheduling in the embedded control system design, a significant improvement of 10-20% in the control performance

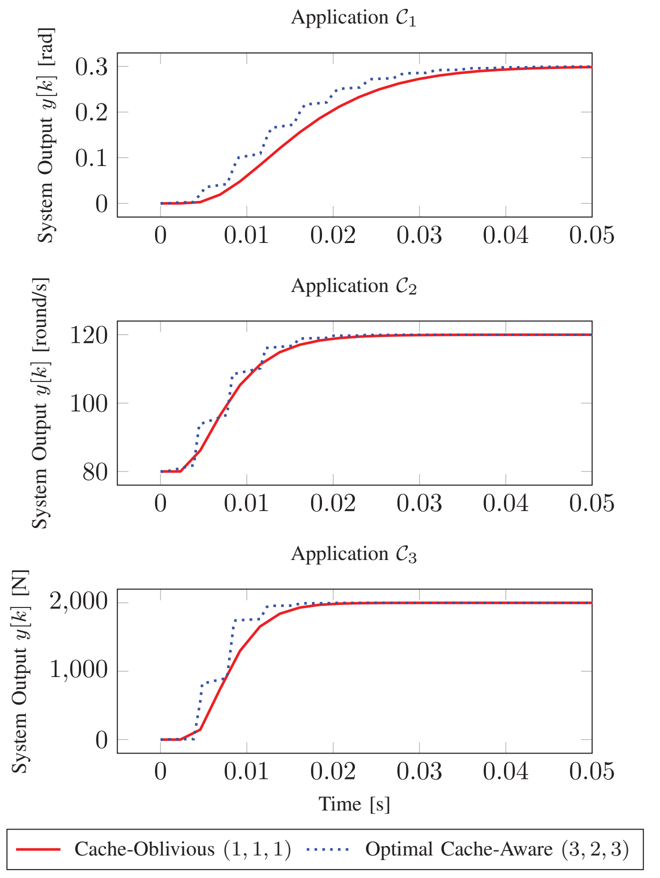


Fig. 6. Control system outputs of the cache-oblivious and optimal cache-aware schedules.

is achieved. The settling time is derived from simulation and as discussed in Section II-A, measured in the most conservative manner. That is, the reference tracking for an application starts after its last consecutive task in a schedule. In this case, the cache-aware schedule with longer idle time before the controller starts to make an effect is at a disadvantage. Therefore, the control performance improvement can be even more in practice.

VI. Concluding Remarks

This work deals with multiple embedded control applications running on a single processor with shared cache. It can be naturally extended to a multi-core architecture, where each core has its own cache. We maximize the overall control performance by an optimal choice of schedule taking into account the effects of cache reuse, in an integrated framework of schedule computation and controller design. The proposed method supported by the experimental results clearly shows its benefit in terms of design optimality. It further establishes potential impact of the memory hierarchy in the design of embedded control systems.

In this work, periodic schedules $(m_1, m_2, \ldots, m_n)$ as defined in Section II are considered. As part of the future research, it should be studied whether more general interleaved schedules, such as $(m_1(1), m_2, m_1(2), m_3)$ ($\mathcal{C}_1$ is consecutively executed $m_1(1)$ times, followed by $\mathcal{C}_2$ m_2 times, $\mathcal{C}_1$ $m_1(2)$ times and $\mathcal{C}_3$ m_3 times), result in better overall control performance, and if they do, what is the optimal schedule. This is a challenging problem to address, since the schedule format is not fixed anymore and the number of schedules to consider increases. In addition, we only consider static schedules resulting in fixed timing that can be exploited by the controller design to maximize the control performance. With scheduling policies resulting in dynamic schedules, it is very challenging to optimize the control performance and instead some basic properties (such as stablity) are often resorted to. This could be another interesting research direction.

References

[1] Q. Leng, Y. Wei, S. Han, A. K. Mok, W. Zhang, and M. Tomizuka, "Improving control performance by minimizing jitter in RT-WiFi networks," in *Proceedings of the 35th. IEEE Real-Time Systems Symposium (RTSS)*, 2014.

[2] S. Al-Areqi, D. Görges, and S. Liu, "Event-based control and scheduling codesign: stochastic and robust approaches," *IEEE Transactions on Automatic Control*, vol. 60, no. 5, pp. 1291–1303, 2015.

[3] D. Roy, L. Zhang, W. Chang, D. Goswami, and S. Chakraborty, "Multi-objective co-optimization of flexray-based distributed control systems," in *Proceedings of the 22nd. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016.

[4] R. Castane, P. Marti, M. Velasco, A. Cervin, and D. Henriksson, "Resource management for control tasks based on the transient dynamics of closed-loop systems," in *Proceedings of the 18th. Euromicro Conference on Real-Time Systems (ECRTS)*, 2006.

[5] L. Greco, D. Fontanelli, and A. Bicchi, "Design and stability analysis for anytime control via stochastic scheduling," *IEEE Transactions on Automatic Control*, vol. 56, no. 3, pp. 571–585, 2011.

[6] S. Chakraborty, M. A. A. Faruque, W. Chang, D. Goswami, M. Wolf, and Q. Zhu, "Automotive cyber-physical systems: A tutorial introduction," *IEEE Design & Test*, vol. 33, no. 4, pp. 92–108, 2016.

[7] K. W. Batcher and R. A. Walker, "Dynamic round-robin task scheduling to reduce cache misses for embedded systems," in *Proceedings of the 11th. Conference on Design, Automation and Test in Europe (DATE)*, 2008.

[8] A. Moonen, M. Bekooij, R. van den Berg, and J. van Meerbergen, "Cache aware mapping of streaming applications on a multiprocessor system-on-chip," in *Proceedings of the 11th. Conference on Design, Automation and Test in Europe (DATE)*, 2008.

[9] W. Chang, D. Goswami, S. Chakraborty, L. Ju, C. J. Xue, and S. Andalam, "Memory-aware embedded control systems design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 4, pp. 586–599, 2017.

[10] I. Bate, P. Nightingale, and A. Cervin, "Establishing timing requirements and control attributes for control loops in real-time systems," in *Proceedings of the 15th. Euromicro Conference on Real-Time Systems (ECRTS)*, 2003.

[11] O. Ljungkrantz, H. Lönn, H. Blom, C. Ekelin, and D. Karlsson, "Modelling of safety-related timing constraints for automotive embedded systems," in *Proceedings of the 2012 International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, 2012.

[12] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, and D. W. et al., "The worst-case execution-time problem – overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, 2008.

[13] S. Chakraborty, T. Mitra, A. Roychoudhury, and L. Thiele, "Cache-aware timing analysis of streaming applications," *Real-Time Systems*, vol. 41, no. 1, pp. 52–85, 2009.

[14] D. Sedighizadeh and E. Masehian, "Particle swarm optimization methods, taxonomy and applications," *International Journal of Computer Theory and Engineering*, vol. 1, no. 5, pp. 486–502, 2009.

[15] J. Ackermann and V. Utkin, "Sliding mode control design based on Ackermann's formula," *IEEE Transactions on Automatic Control*, vol. 43, no. 2, pp. 234–237, 1998.

[16] P. Yih, "Steer-by-wire: Implications for vehicle handling and safety," Ph.D. dissertation, Stanford University, 2005.

[17] W. Chang, A. Pröbstl, D. Goswami, M. Zamani, and S. Chakraborty, "Battery- and aging-aware embedded control systems for electric vehicles," in *Proceedings of the 35th. IEEE Real-Time Systems Symposium (RTSS)*, 2014.

[18] J. Fox, R. Roberts, C. Baier, L. Ho, L. Lacraru, and B. Gombert, "Modeling and control of a single motor electronic wedge brake," SAE Technical Paper, Tech. Rep., 2007.