Received August 27, 2019, accepted September 13, 2019, date of publication September 18, 2019, date of current version September 30, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2941958

Sensitivity Analysis of Strictly Periodic Tasks in Multi-Core Real-Time Systems

JINCHAO CHEN[1], CHENGLIE DU[1], PENGCHENG HAN[1], AND YONG ZHANG[2]
[1]School of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China
[2]Department of Software System Development, North Automatic Control Technology Institute, Taiyuan 030000, China

Corresponding author: Jinchao Chen (cjc@nwpu.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB1001900, and in part by the Defense Industrial Technology Development Program of China under Grant JCKY2016607B006 and Grant JCKY2017208B017.

ABSTRACT In the design phase of real-time systems, it cannot be expected that the timing attributes of all tasks are completely specified and never changed. The increased computation times or shortened periods in a schedulable system often cause deadlines to be missed. In such situations, sensitivity analysis is an effective approach to provide quantitative indications for the design modification, by identifying the borderlines on parameter variations while keeping the systems schedulable. In this paper, we propose a new approach to analyze the sensitivity of the timing parameters of tasks with strict periods in multi-core real-time systems. We first analyze a schedulability condition to determine whether a task is schedulable on a given processor without changing the start times of the existing tasks. Then, following a game theory analogy, we design recursive algorithms to compute the permissible changes in the task timing parameters, by allowing each task to optimize its own start time and processor allocation. Finally, we conduct experiments with randomly generated tasks to show that our approach is more efficient than the existing solutions to solve the sensitivity problem. The proposed approach has a wide range of applications, only guiding the design of multi-core systems, but also improving the robustness of a design subject to future changes.

INDEX TERMS Sensitivity analysis, scheduling, strictly periodic task, multi-core real-time system, schedulability analysis, scheduling algorithm.

I. INTRODUCTION

Scheduling and schedulability analysis for real-time systems have been a fundamental issue in providing guarantees for temporal feasibility of task execution in anticipated situations. Schedulability theory checks the time constraints of tasks, and significantly improves the efficiency of designs and implementations of real-time systems. However, it cannot be expected that all task parameters required by the schedulability analysis are fully determined and available up front. In many situations, real-time systems are composed of a high degree of uncertainties on task activations and execution behaviors [1], making the application of schedulability analysis much less productive.

Changes are usually applied to a specific task or an entire system in practice. The computation times which are more accurately estimated and measured with further research, often exceed the initial estimation time budgets and make a system unschedulable. Meanwhile, in order to perfect or extend system functionalities, computation times and periods of certain tasks may be modified. In all aforementioned cases, designers need to keep track of the flexibility of a system [2], and quickly determine how much available slack there is before the system becomes unschedulable. This is the domain of sensitivity analysis.

Sensitivity analysis is an effective approach to deal with uncertainties that result from inaccurate specifications and to provide an accurate prediction for future modifications. It determines the bounds on parameter variations under which schedulability constraints of systems are not violated. Sensitivity analysis only offers the exact amount of change affordable in task timing parameters before a schedulable system becomes unschedulable, but also provides quantitative indications of the actions (e.g., the decrease of task computation times or the increase of handling speed of processors) required to bring an unschedulable system back into a schedulable state. Sensitivity analysis is a trustworthy method to improve the flexibility and robustness of a design

The associate editor coordinating the review of this manuscript and approving it for publication was Shih-Wei Lin.

subject to future changes [3], by allowing the designers to reallocate the use of total resources as needed instead of being confined by initial boundaries.

Tasks with strict periods are usually adopted in practical real-time applications such as avionics [4], [5] and process control [6], [7] where continual sampling and processing of data are required accurately. The range of application domains where one error or miss may lead to a disastrous consequence makes the change of these systems challenging and time consuming. Faced with this difficult situation, the system designers inevitably tend to rely on decision-making tools to determine whether changes are affordable into the original system.

In this paper, we study the scheduling and sensitivity problem of strictly periodic tasks on a multi-core platform, and provide an informative result to show the effect that the modification of each task parameter may have on the schedulability of a system. We mainly aim at providing the indications on how much change that the task parameters could afford without violating schedulability constraints, which not only helps in dealing with uncertainties that result from inaccurate specifications, but also guides the design and modification of multi-core real-time systems. We address three aspects during the system design process:

1. How to determine whether a strictly periodic task is schedulable in a multi-core real-time system?

2. What is the exact amount of change affordable in the computation time and the period of a single task to keep the system schedulable?

3. What is the maximum value of the scaling factor [8] (i.e., the possible change for the computation times of all tasks) before a system becomes unschedulable?

Real-time scheduling and sensitivity problem is widely studied in large-scale systems such as Internet of Things [9] and Cyber-Physical Systems [10]. Significant efforts have been made to provide efficient methods to solve the scheduling and sensitivity problem. Based on fuzzy theory and a genetic algorithm, Shojafar *et al.* [11] presented a hybrid job scheduling approach to assign jobs with reducing total execution time and execution cost in cloud computing. Racu *et al.* [12] presented a sensitivity analysis framework for both one-dimensional and multi-dimensional sensitivity of large real-time systems with complex timing dependencies and requirements. However, in the aforementioned works, the periods of tasks were not strict and some slack time was allowed between successive instances of a periodic task. Therefore, the schedulability conditions and sensitivity approaches proposed in those works are not suitable for strictly periodic tasks considered in this work.

The scheduling and sensitivity problem of strictly periodic tasks is classified as a non-preemptive and strictly periodic multiprocessor scheduling problem [5]. This problem is very challenging because of the strict periodicity constraint, which not only adds computation complexity [13] on the problem, but also compounds the difficulty in obtaining the boundary scheduling conditions [14]. Even though some efforts have been made to schedule the strictly periodic tasks and schedulability conditions have been proposed, only a few approaches have been proposed to analyze the sensitivity of strictly periodic tasks. Sheikh *et al.* [5] calculated the critical scaling factor by a best-response algorithm, and used the factor to determine whether all tasks were schedulable on a limited number of processors. Pira and Artigues [15] did a similar work and gave a new heuristic to solve the problem with a propagation mechanism for non-overlapping constraints. However, all of the aforementioned approaches only calculated the maximum scaling factors of tasks, sensitivity in the domains of computation times and periods was not taken into account.

In this paper, we first analyze a schedulability condition to determine whether strictly periodic tasks are schedulable in a multi-core real-time systems, then propose approaches to seek affordable changes in the task timing parameters before the system becomes unschedulable. We not only calculate the maximum scaling factors for all tasks, but also analyze the sensitivity of the computation time and the period for a single task. The proposed approach has a wide range of applications and can be adapted to tasks with both harmonic and non-harmonic periods. It not only guides the development of multi-core systems, but also improves the robustness of a design subject to future changes.

A. MOTIVATION EXAMPLE

We illustrate the main problems studied in this paper with a simple example. Consider a set that contains three tasks τ_1, τ_2 and τ_3 with strict periods. As indicated in Fig. 1(a), the computation times of the three tasks are 2; while the periods are 6, 12 and 12 respectively. From Fig. 1(a) we can see that the three tasks have no overlapping time unit and are schedulable on a uniprocessor platform. We are interested in analyzing the sensitivity of task timing parameters from the following three aspects.

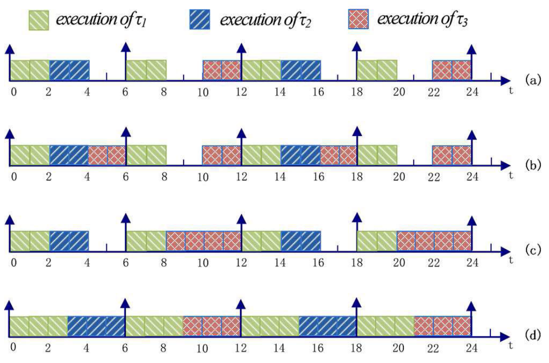

FIGURE 1. A motivation example constituted of three tasks.

(A) Suppose τ_3 is a real-time task used to collect data samples at each period. The decrease of the period of τ_3 would reduce the time interval between two successive collections and make the produced data more accurate. As shown in Fig. 1(b), when the period of τ_3 is 6, the three tasks are still schedulable, which means that the period of τ_3 could be shortened to 6 at least. However, what is the minimum value of the period that τ_3 can have while keeping the task set schedulable? We need to keep the computation times constant and find the bound on the period of τ_3.

(B) If we want to extend the functionality of τ_3, the computation time of τ_3 would be increased. What is the maximum permissible increase in the computation time that τ_i can afford without sacrificing schedulability? At this time, the period of τ_3 should be considered fixed and the computation time of τ_3 should be extended as much as possible before the system becomes unschedulable. Figure 1(c) shows the non-overlapping execution of the three tasks when the computation time of τ_3 is increased to 4.

(C) Since slowing down processor clock speed by adjusting the frequency is an effective method of reducing power consumption [16], we try to reduce the handling speed of processors furthest. Then a new problem arises: how much reduction in the speed of the processor is affordable to keep the task set still schedulable? As can be seen from Fig. 1(d), the instances of the three tasks do not collide in time even when the computation times of all tasks are multiplied by 1.5 proportionally, which means the speed of the processor can be slowed down by one third at least.

B. CONTRIBUTIONS

In this paper, we study the sensitivity problem for tasks with strict periods and propose a new approach to compute the admissible changes in the timing parameters upon a multi-core real-time platform. The contributions of this work are summarized as follows.

First, we analyze a schedulability condition to determine whether a task is schedulable on a given processor without changing the start times of the existing tasks. This condition works no matter how many tasks have already been assigned to a specific processor, and does not require any form of search. It is efficient and provides a schedulability constraint for the computation time and period of a single task when all existing tasks are fixed.

Second, using the schedulability condition proposed, we build exact formulations to seek the permissible changes in the task computation times, periods and scaling factors, while keeping the task set schedulable in a multi-core system. The obtained values are the borderlines on the task timing parameters, and help guide the design and modification of real-time systems.

Third, following a game theory analogy, we design recursive algorithms to calculate affordable changes in the task timing parameters, by allowing each task to optimize its own offset and processor allocation. Unfortunately, the values obtained from these algorithms may be not exact because the algorithms stop when equilibrium states are reached and do not search the solution space completely. However, the speeds of these algorithms are fast, and the relative error ratios are in the allowable range, which can be accepted when taking into account the time cost.

C. ORGANIZATION

The rest of the paper is organized as follows. The related work is introduced in Sect. II. The notations and task model used in this paper is presented in Sect. III. A schedulability condition for determining whether a task is schedulable is analyzed in Sect. IV. Sect. V analyzes the maximum computation time that a single task can have while keeping the task set schedulable. Sect. VI gives a method to find the minimum value of the period of a single task before the system becomes unschedulable. Sect. VII calculates the maximum scaling factor of the computation times of all tasks. Sect. VIII provides the simulation experiments and evaluates the performance of our approaches with respect to the corresponding exact formulations. Finally, Sect. IX presents the conclusions of this paper and the directions for future work.

II. RELATED WORK

In the research area of sensitivity analysis, great efforts have been made concerning preemptive scheduling systems, especially uniprocessor systems with fixed priority scheduling algorithms. The first work to the sensitivity problem was done by Lehoczky *et al.* [8]. The authors considered the fixed priority preemptive scheduling with rate monotonic priority assignment, and defined the critical scaling factor as the largest possible change for the computation times of all tasks, while still guaranteeing the schedulability of the task set.

Lots of improvements to Lehoczky's analysis were made based on fixed priority or other scheduling algorithms. Vestal [17] introduced slack variables into the exact schedulability conditions proposed in [8]. Punnekkat *et al.* [18] used an efficient approach, which combined a binary search with modified version of response time schedulability, to obtain the sensitivity bounds. Racu *et al.* [12] presented a sensitivity analysis framework for both one-dimensional and multi-dimensional sensitivity of large real-time systems with complex timing dependencies and requirements. Zhang *et al.* [3] addressed the changes in task timing parameters under a uniprocessor platform for arbitrary deadline real-time systems with the Earliest Deadline First algorithm. However, all those works study the parameter variations based on non-strictly periodic tasks. The time duration between two successive instances of a task may vary whereas it is a constant in our case.

Strictly periodic tasks are usually adopted in the time-triggered systems [19], [20], where tasks are activated by the progression of time only. Some research has been done to analyze the changes affordable and to calculate the maximum scaling factor of the computation times of all tasks. Sheikh *et al.* [5] first derived a method to calculate the largest evolution coefficient (i.e., the maximum rate of the increase of each task computation time) for one task. Then they extended this method to all tasks and proposed a best-response algorithm to find the critical scaling factor for the task set. Finally, they used the critical scaling factor to determine whether all tasks were schedulable upon a limited number of processors. Afterword, Pira and Artigues [15] improved the best-response algorithm presented by Sheikh *et al.* [5] with local optimization and a propagation mechanism.

To the best of our knowledge, this is the first work studying the sensitivity analysis of the computation time and the period for a single task with strict period under non-preemptive scheduling in a multi-core real-time system. Fortunately, significant efforts have been made to schedule the strictly periodic tasks and some efficient schedulability conditions have been proposed.

Korst *et al.* [21] presented a sufficient and necessary schedulability condition for two strictly periodic tasks, which had been proved to be a sufficient condition by Kermia and Sorel [6] in multi-task situations. Eisenbrand *et al.* [4] used bin trees to analyze the schedulability of multiple tasks under the constraint that the periods of all tasks were harmonic, i.e., for each two tasks, the period of one task is a multiple of that of the other one. Marouf and Sorel [22] did a similar work and gave a schedulability condition for a new task when its period was a multiple of one of the existing tasks. Zhang *et al.* [23] presented an efficient approach to select the start times and provided a sufficient schedulability condition for strictly periodic tasks on a uniprocessor. Even though the schedulability conditions proposed in the aforementioned works have special constraints that sharply restrict the range of applications, they help the researchers in analyzing the parameter variations of the strictly periodic tasks.

In Chen el al. [24], the authors represented a strictly periodic task by its *eigentask* (i.e., setting its worst case execution time to 1), calculated the valid scheduling slots for a new task and presented sufficient schedulability conditions to determine whether the new task is schedulable on a limited number of processors. In this research, based on the schedulability conditions proposed in [24], we provide approaches to analyze the indications on how much change that the task parameters could afford without violating schedulability constraints. We not only calculate the exact amount of change affordable in the computation time and the period of a single task, but also seek the maximum value of the scaling factor of all tasks. Experiments with randomly generated tasks show that our approach is more efficient than the existing solutions to solve the sensitivity problem of strictly periodic tasks.

III. NOTATIONS AND SYSTEM MODEL

In this paper, we consider a real-time system constituted of m identical processors on which a set of n tasks with strict periods are scheduled. Each task is independent and consists of an infinite stream of instances that should be executed non-preemptively. A task τ_i $(1 \leq i \leq n)$ is characterized by a tuple $\tau_i = \langle c_i, p_i \rangle$ where c_i is its computation time and p_i is its period with $0 < c_i \leq p_i$. We use s_i and a_i to denote the offset (i.e., the start time of the first instance) and the assignment (i.e., the processor to which the task is assigned) of τ_i, then $0 \leq s_i \leq p_i - c_i$ and $1 \leq a_i \leq m$. We assume that the period and computation time of each task are multiples of the basic dealing cycles of the processor clocks, i.e., $\forall i \in [1, n]$, $p_i, c_i \in \mathbb{N}^*$.

For any task τ_i, one instance is generated at every time unit $s_i + kp_i$ $(k \in \mathbb{N})$ and should be executed immediately after its generation. Let $B_i^k(s_i)$ denote the time units used by the kth instance of τ_i, then using the strict periodicity constraint, there is: $\forall \tau_i \in T, \forall k \in \mathbb{N}, B_i^k(s_i) = [s_i + kp_i, s_i + kp_i + c_i)$. We use E_i to represent the time units occupied by all instances of τ_i, i.e., $E_i = \bigcup_{k\in\mathbb{N}} B_i^k(s_i)$. Figure 2 shows the timing characters of a strictly periodic task.

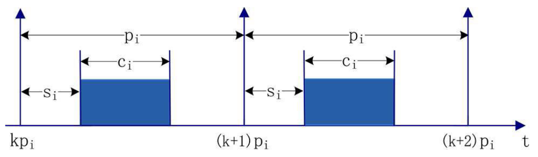

FIGURE 2. Strictly periodic task model used in this paper.

In a multi-core real-time system, tasks are unevenly distributed and the number of tasks allocated to each core may be different. Let T_p denote the set of tasks assigned to the processor p $(1 \leq p \leq m)$. Meanwhile, we use $g_{i,j}$ to represent the greatest common divisor (GCD) of the periods of any two tasks τ_i and τ_j, i.e., $g_{i,j} = GCD(p_i, p_j)$. Table 1 summarizes the basic notations used in this paper.

TABLE 1. Basic notations used in this paper.

Symbol	Description
T	the periodic task set
n	the number of tasks in T
m	the number of identical processors in the system
τ_i	the ith task in T
c_i	the computation time of τ_i
p_i	the period of τ_i
s_i	the offset of τ_i
a_i	the assignment of τ_i
T_p	the task set allocated to the processor p
E_i	the time units used by all instances of τ_i
$g_{i,j}$	the greatest common divisor of the periods of τ_i and τ_j
$B_i^k(s_i)$	the time units used by the kth instance of τ_i

IV. PREVIOUS RESULTS ON SCHEDULABILITY ANALYSIS FOR STRICTLY PERIODIC TASKS

Schedulability analysis which determines whether a set of tasks with temporal parameters meets their constraints according to a given scheduling algorithm, can significantly increase the efficiency of design of real-time systems, and is a critical foundation for sensitivity analysis.

In this section, we analyze the previous research results on schedulability analysis for tasks with strict periods. We first introduce a schedulability condition for two tasks allocated to the same processor. Then, we represent a task by its eigentask (i.e., setting its computation time to 1), and calculate all free scheduling slots for the eigentask. Finally, we obtain a new schedulability condition to determine whether the original task is schedulable using the free scheduling slots calculated. The schedulability condition proposed provides a constraint on the computation time and period of a single task when it is schedulable with other tasks, and will be adopted to analyze the parameter variations of strictly periodic tasks in later sections.

A. SCHEDULABILITY ANALYSIS FOR TWO TASKS WITH STRICT PERIODS

In 1991, Korst *et al.* [21] noted that two strictly periodic tasks are schedulable if and only if the time units occupied by their instances do not collide, and proposed a necessary and sufficient condition to determine whether two tasks with strict periods were schedulable. The schedulability condition is given by the following theorem.

Theorem 1: Two strictly periodic tasks $\tau_i = \langle c_i, p_i \rangle$ and $\tau_j = \langle c_j, p_j \rangle$ are schedulable on the same processor if and only if

$$c_i \le (s_j - s_i) mod(g_{i,j}) \le g_{i,j} - c_j \tag{1}$$

We can observe that Condition (1) works for two tasks at a time and cannot be applied into multi-task situations. It is difficult to directly propose a more general schedulability condition to determine whether all tasks are schedulable on a given processor. We solve this multi-task scheduling problem by adopting the concepts of *eigentask* and *eigenoffset*, which were inspired from *eigenvector* [25] in the matrix theory and first introduced by Chen *et al.* [26].

B. EIGENTASK AND EIGENOFFSET

If a new task is schedulable on a given processor, there exists at least one valid offset such that the instances of the new task have no overlapping time unit with those of other tasks. Therefore, it is a very reliable method to determine the schedulability of a new task by analyzing whether a valid offset exists.

Assume that $T = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is a set constituted of all schedulable tasks and $\tau_r = \langle c_r, p_r \rangle$ is a new task to be scheduled. It is difficult to obtain all valid offsets for τ_r in a multi-task situation. However, if we can get the valid offsets for a task $\tau_{r'} = \langle 1, p_r \rangle$, it is quite easy to determine whether enough many (i.e., as many as the computation time of τ_r) consecutive free time units are available. If there are, τ_r can be executed in the corresponding time intervals and is schedulable with all tasks in T.

We refer to $\tau_{r'} = \langle 1, p_r \rangle$ as the *eigentask* of $\tau_r = \langle c_r, p_r \rangle$. That is to say, eigentask is the task whose computation time is 1 but period is the same as that of the original task. Meanwhile, we refer to the set constituted of all valid offsets of $\tau_{r'}$ as the *eigenoffset* of τ_r, and use $E_T(r)$ to denote the eigenoffset of τ_r. Then, $\tau_{r'}$ is schedulable with all tasks in T if its offset s is one of the integers in $E_T(r)$, which can be expressed as:

$$\forall k, l \in \mathbb{N}, \quad \forall s \in E_T(r),\ B_i^k(s_i) \cap B_{r'}^l(s) = \emptyset$$

The following theorem which was proposed in [24], promoted a method for calculating the eigenoffset.

Theorem 2: An eigentask $\tau_{r'} = \langle 1, p_r \rangle$ is schedulable with all tasks in a set T on a uniprocessor platform if and only if its offset s

$$s \in Z(r) \setminus \bigcup_{\tau_i \in T} BTU(i, r) \tag{2}$$

In Condition 2, $Z(r)$ denotes the time interval from 0 to $p_r - 1$, i.e., $Z(r) = [0, p_r - 1]$, and $BTU(i, r)$ can be calculated by:

$$BTU(i, r) = \{e' \mid e' = (e) mod(p_r), \quad \forall e \in B_i^k(s_i), \\ \forall k \in [0, \frac{v}{p_i}), v = LCM(p_i, p_r)\}$$

From Theorem 2 we know, $Z(r) \setminus \bigcup_{\tau_i \in T} BTU(i, r)$ contains all valid offsets for the eigentask $\tau_{r'}$. Then the eigenoffset $E_T(r)$ of τ_r can be calculated via,

$$\begin{aligned} E_T(r) &= Z(r) \setminus \bigcup_{\tau_i \in T} BTU(i, r) \\ &= \{x \in \mathbb{N} \mid x \in [0, p_r - 1], x \notin \bigcup_{\tau_i \in T} BTU(i, r)\} \end{aligned} \tag{3}$$

C. SCHEDULABILITY DETERMINATION

The eigenoffset calculated in Sect. IV-B contains all free time units that can be used for the eigentask of a given task. In this section, we use the eigenoffset to determine whether a new task is schedulable with all tasks running on the same processor. We need to introduce a notation $LLC(S)$, which was defined in Chen *et al.* [27] and represents the longest length of consecutive integers in a finite set S.

Theorem 3: A task $\tau_r = \langle c_r, p_r \rangle$ is schedulable with all tasks in a set T on a uniprocessor platform if and only if

$$c_r \le LLC(E_T(r)) \tag{4}$$

where $E_T(r)$ is given in Eq. (3)

Proof: We assume $\tau_{r'} = \langle 1, p_r \rangle$ is the eigentask of τ_r. τ_r is schedulable with all tasks in T if and only if the instances of τ_r and the existing tasks in T do not collide, i.e., $\forall k, l \in \mathbb{N}$, $\forall \tau_i \in T$, $B_r^k(s_r) \cap B_i^l(s_i) = \emptyset$.

$$\begin{aligned} & B_r^k(s_r) \cap B_i^l(s_i) = \emptyset \\ \iff & [s_r + kp_r, s_r + kp_r + c_r - 1] \cap B_i^l(s_i) = \emptyset \\ \iff & (\bigcup_{w=0}^{c_r - 1} [s_r + kp_r + w, s_r + kp_r + w]) \cap B_i^l(s_i) = \emptyset \\ \iff & (\bigcup_{w=0}^{c_r - 1} B_{r'}^k(s_r + w)) \cap B_i^l(s_i) = \emptyset \\ \iff & B_{r'}^k(s_r + w) \cap B_i^l(s_i) = \emptyset, \quad \forall w \in [0, c_r - 1] \end{aligned}$$

Therefore, τ_r is schedulable with all tasks in T if and only if there are at least c_r consecutive integers from s_r to $s_r + c_r - 1$ in the valid offsets of $\tau_{r'}$, i.e., $c_r \le LLC(E_T(r))$. □

Condition (4) determines the schedulability of a task quickly and can be exploited to deliver the optimal timing parameters of the existing tasks. Based on this condition, we analyze the maximum computation time and the minimum period of a single task, and calculate the maximum scaling factor of the computation times of all tasks in later sections.

V. MAXIMUM COMPUTATION TIME CALCULATION

Since computation time is one of the most important timing factors in a real-time system, the modification of the computation time of a single task is a common form of sensitivity analysis and has received lots of attention in the research community. In this section, we analyze the maximum possible increase in the computation time that a single task can afford without sacrificing the schedulability of the system. We assume that the design variable is only the computation time of the target task, whereas the period of the target task and the timing parameters of other tasks remain the same.

We assume the task that we want to maximize its computation time is $\tau_r = \langle c_r, p_r \rangle$. We need to calculate the largest value of c_r before the task set T becomes unschedulable. This means that the computation time of τ_r can be set to c_r while keeping T schedulable, and any small increase in c_r would make the set T unschedulable. We first investigate an exact formulation of this maximization problem based on linear programming, then propose a more efficient heuristic to generate optimized allocations for the tasks and get the maximum computation time for the target task.

A. EXACT FORMULATION

The calculation problem of the maximum computation time can be defined as maximizing an object (computation time c_r) subject to linear constraints (schedulability constraints of all tasks). Because the timing parameters of tasks in this paper are integers, we propose an exact formulation based on Mixed Integer Linear Programming (MILP) [28], which can completely search the solution space if no time limit is set.

We use Condition (1) to describe the non-overlapping constraint for two tasks; however the modulo operation (*mod*) in it is not linear. In MILP, the modulo operation should be transformed to

$$(s_j - s_i) mod(g_{i,j}) = (s_j - s_i) - g_{i,j} \times e_{i,j} \tag{5}$$

where $e_{i,j} = \lfloor \frac{s_j - s_i}{g_{i,j}} \rfloor$. $e_{i,j}$ is a new integer variable representing the quotient from the modulo operation. Since $0 \le s_i \le p_i - c_i$ and $0 \le s_j \le p_j - c_j$, the value of $e_{i,j}$ ranges from $\frac{c_i - p_i}{g_{i,j}}$ to $\frac{p_j - c_j}{g_{i,j}}$. Therefore, Condition (1) becomes

$$c_i \le (s_j - s_i) - g_{i,j} \times e_{i,j} \le g_{i,j} - c_j$$
$$\frac{c_i - p_i}{g_{i,j}} \le e_{i,j} \le \frac{p_j - c_j}{g_{i,j}}$$

The MILP formulation for the maximum computation time problem can be written as the following program:

$$\begin{aligned}
&\text{maximize } c_r \\
&\text{subject to } c_i \le (s_j - s_i) - g_{i,j} \times e_{i,j} \le g_{i,j} - c_j \\
&\qquad \frac{c_i - p_i}{g_{i,j}} \le e_{i,j} \le \frac{p_j - c_j}{g_{i,j}} \\
&\qquad \forall i,\ j \in [1, n],\ a_i = a_j, j \ne i \qquad (6) \\
&\qquad c_r \in (0, p_r] \qquad (7) \\
&\qquad a_i \in [1, m], \quad s_i \in [0, p_i - c_i], \quad \forall i \in [1, n] \qquad (8)
\end{aligned}$$

Condition (6) describes the non-overlapping constraints of tasks allocated to the same processor. Condition (7) requires that the value of c_r should be no more than p_r, and Condition (8) shows the range restrictions of the offsets and assignments of all tasks.

Even though this MILP formulation can find an exact solution, it has a serious limitation. It considers all possible allocations of tasks and has n^2+n integer variables. The exact MILP formulation is inefficient on large scale or even fairly complex problem instances because of the required execution times. This is due to the fact that the non-preemptive allocation and scheduling problem for tasks with strict periods is NP-Hard [21]. A relatively efficient heuristic is proposed in the following sections.

We try to optimize the offset s_i and assignment a_i of one task τ_i at a time, while keeping the other tasks fixed. All tasks take turns to select their best offsets and assignments such that the target task can have the largest permissible computation time according to the current allocations. This method has an analogy with Game Theory, where each player makes a best response according to the mostly known strategies of the other players. This method is known as the Best Response solution, which was first introduced by Sheikh *et al.* in [5], [29], and also studied by Pira and Artigues in [15], [30].

In the following, we first give a best offset procedure to find the best start time for a task on a given processor while keeping the other task fixed. Then, we extend this procedure to all processors and design a best response procedure to obtain the best assignment besides the best offset. Finally, we present a heuristic to calculate the maximum computation time for the target task by moving the tasks round after round until an equilibrium is reached.

B. BEST OFFSET PROCEDURE ON A PROCESSOR

For each task τ_i ($1 \le i \le n$), we design a best offset procedure $BO_i^p(r)$ to get an optimal offset for τ_i such that τ_r can have the largest permissible computation time on a given processor p, while keeping the offsets and assignments of other tasks fixed. As we pointed out in Sect. III, T_p represents all tasks allocated to the processor p. We use T_p^{-i} to represent all tasks in T_p but τ_i, i.e., $T_p^{-i} = T_p \setminus \{\tau_i\}$. Then, the offsets of all tasks in T_p^{-i} remain the same.

From Condition (4) in Theorem 3 we know, for the target task τ_r, the largest length of consecutive integers in its eigenoffset is its *computation time bound*, i.e., the maximum permissible computation time that τ_r can have when it is schedulable on the same processor. Then, for each allocation of tasks in T_p, there is a computation time bound on the processor p. We need to consider all valid allocations, calculate the corresponding computation time bounds and record the largest one. Since the offsets of tasks in T_p^{-i} remain the same, each offset s_i of τ_i satisfying the non-overlapping constraints leads to a valid allocation. Then, the $BO_i^p(r)$ procedure should consider all valid values of s_i, and find the best offset for τ_i such that τ_i has the largest permissible computation time. Algorithm 1 shows the pseudo-code for this procedure.

Algorithm 1 Best Offset Procedure $BO_i^p(r)$

```
Input: $\tau_i$, $\tau_r$ and a processor $p$
Output: the largest permissible computation time $c_r^p$ for
        $\tau_r$, the best offset $bs_i$ for $\tau_i$
1  $c_r^p \leftarrow -1$; $bs_i \leftarrow -1$;
2  $T_p^{-i} \leftarrow T_p \setminus \{\tau_i\}$;
3  for $k = 0$ to $p_i - c_i$ do
4      $s_i \leftarrow k$;
5      if Condition (6) is satisfied then
6          $E_{T_p}(r) \leftarrow Z(r) \setminus \bigcup_{\tau_j \in T_p^{-i}} BTU(j, r) \setminus BTU(i, r)$;
7          $tc_r \leftarrow LLC(E_{T_p}(r))$;
8          if $tc_r > c_r^p$ then
9              $c_r^p \leftarrow tc_r$; $bs_i \leftarrow k$;
10         end
11     end
12 end
13 return $(c_r^p, bs_i)$;
```

We can find that the main computation part of Algorithm 1 is from line 3 to 12, which is a loop and repeats at most p_i times. In each iteration, the eigenoffset of τ_i is calculated, and the complexity is $O(n_p)$, where n_p is the number of tasks allocated to the processor p. Therefore the total running time of Algorithm 1 is $O(n_p p_i)$. If we use P_{max} to denote the maximum period of all tasks, the running time complexity of Algorithm 1 is $O(n_p P_{max})$.

C. BEST RESPONSE PROCEDURE ON A MULTI-CORE PLATFORM

Based on the best offset procedure $BO_i^p(r)$, we design a best response procedure $BR_i(r)$, which returns the best strategy (i.e., the best offset and the best assignment) of each task τ_i according to the current allocations of other tasks. The best response procedure ensures that the target task τ_r can have the largest permissible computation time when all tasks except τ_i keep fixed.

From Sect. V-B we know, when τ_i is allocated to the processor p, the computation time bound for τ_r on this processor is $BO_i^p(r)$. Meanwhile, according to Theorem 3, the computation time bound for τ_r on any other core c ($1 \le c \le m$ and $c \neq p$) is $LLC(E_{T_c}(r))$. Note that in a multi-core system, τ_r may be executed on any of the processors. We use mc_r^p to denote the maximum permissible computation time of τ_r when τ_i is allocated to the processor p, then

$$mc_r^p = \max\left(BO_i^p(r), \max_{1\le c\le m, c\neq p} LLC(E_{T_c}(r))\right) \quad (9)$$

The best response procedure $BR_i(r)$ needs to assign τ_i to each processor and records the largest value. We use mc_r to denote the maximum permissible computation time for τ_r when only the offset and assignment of τ_i can be changed, then

$$\begin{aligned} mc_r &= \max_{1\le p\le m} mc_r^p \\ &= \max_{1\le p\le m} \max\left(BO_i^p(r), \max_{1\le c\le m, c\neq p} LLC(E_{T_c}(r))\right) \end{aligned} \quad (10)$$

The best response procedure $BR_i(r)$ stops when all processors to which τ_i can be allocated have been considered. The pseudo-code for $BR_i(r)$ is given in Algorithm 2. Since the complexity of the best offset procedure $BO_i^p(r)$ is $O(n_p P_{max})$, the running cost of the best response procedure $BR_i(r)$ is $O(nP_{max})$.

Algorithm 2 Best Response Procedure $BR_i(r)$

```
Input: $\tau_i$ and the target task $\tau_r$
Output: the maximum permissible computation time
        $mc_r$ for $\tau_r$, the best offset $bs_i$ and assignment
        $ba_i$ for $\tau_i$
1  $mc_r \leftarrow -1$; $bs_i \leftarrow -1$; $ba_i \leftarrow -1$;
2  for $p = 1$ to $m$ do
3      $(c_r^p, tbs_i) \leftarrow BO_i^p(r)$;
4      $tLen \leftarrow -1$;
5      for $c = 1$ to $m$ do
6          if $c \neq p$ then
7              $tc \leftarrow LLC(E_{T_c}(r))$
8              if $tc > tLen$ then
9                  $tLen \leftarrow tc$; $ta_r \leftarrow c$;
10             end
11         end
12     end
13     if $c_r^p < tLen$ then
14         $c_r^p \leftarrow tLen$;
15     end
16     if $mc_r < c_r^p$ then
17         $(mc_r, bs_i, ba_i) \leftarrow (c_r^p, tbs_i, p)$;
18     end
19 end
20 return $(mc_r, bs_i, ba_i)$;
```

D. EQUILIBRIUM-BASED HEURISTIC

In this section, we extend τ_i to all tasks in the system and present an equilibrium-based heuristic to calculate the maximum permissible computation time for τ_r when all tasks can change their offsets and assignments freely. Each task optimizes its offset and assignment according to the best response procedure $BR_i(r)$ proposed in Sect. V-C at its turn. The tasks are moved round after round until an equilibrium is reached, i.e., no task can improve its offset or assignment using the best response procedure.

Let T^k ($k \ge 1$) represent the tasks after the kth iteration, then the heuristic stops when $T^k = T^{k-1}$. We use $c_{r\cdot i}^k$, s_i^k and a_i^k to denote the permissible computation time calculated for τ_r, the corresponding offset and assignment for τ_i after τ_i

updates its allocation in the kth iteration. Algorithm 3 shows the pseudo-code for this equilibrium-based heuristic.

Algorithm 3 Maximum Computation Time Algorithm (MCTA)

```
Input: $\tau_r$ and a task set $T$
Output: the maximum permissible computation time
        $mc_r$ for $\tau_r$
$k \leftarrow 0$;
repeat
    for $i = 1$ to $n$ do
        if $i \neq r$ then
            $(c^k_{r\cdot i}, s^k_i, a^k_i) \leftarrow BR_i(r)$;
            $(s_i, a_i) \leftarrow (s^k_i, a^k_i)$;
            $mc_r \leftarrow c^k_{r\cdot i}$;
        end
    end
    $k \leftarrow k + 1$;
until $T^k = T^{k-1}$;
return $mc_r$
```

E. CONVERGENCE AND COMPLEXITY ANALYSIS

In this section, we analyze the convergence and computation complexity of the equilibrium-based heuristic proposed in Sect. V-D. We first prove that this heuristic will converge in a finite number of iterations. Then using the converge property, we analyze the computation complexity of this heuristic and show that this heuristic runs in pseudo-polynomial time.

As with efficiency and dependability, convergence is a desired property for recursive algorithms. We first prove that the permissible computation time calculated by the best response procedure increases as the iterative process continues. Then we show that the calculated values are bounded in all iterations and the maximum computation time algorithm converges in a finite number of iterations.

Lemma 1: In the iterative processes, there are: $\forall k \geq 1$, $\forall i \in [1, n-1], i \neq r$

$$c^k_{r\cdot i} \leq c^k_{r\cdot(i+1)} \quad \textit{and} \quad c^k_{r\cdot n} \leq c^{k+1}_{r\cdot 1}$$

Proof: When τ_i uses the best response procedure $BR_i(r)$ to improve its offset and assignment in the kth iteration, only the tasks whose turns are in front of τ_i have optimized their allocations in this kth iteration. Then, $c^k_{r\cdot i}$ is obtained when the offset sequence and assignment sequence for all tasks are $[s^k_1, s^k_2, \ldots, s^k_{i-1}, s^k_i, s^{k-1}_{i+1}, \ldots, s^{k-1}_n]$ and $[a^k_1, a^k_2, \ldots, a^k_{i-1}, a^k_i, a^{k-1}_{i+1}, \ldots, a^{k-1}_n]$. Now τ_{i+1} begins to update its offset and assignment. The offset sequence and assignment sequence for all tasks except τ_{i+1} are $[s^k_1, s^k_2, \ldots, s^k_{i-1}, s^k_i, s^{k-1}_{i+2}, \ldots, s^{k-1}_n]$ and $[a^k_1, a^k_2, \ldots, a^k_{i-1}, a^k_i, a^{k-1}_{i+2}, \ldots, a^{k-1}_n]$. When the offset and assignment of τ_{i+1} are equal to s^{k-1}_{i+1} and a^{k-1}_{i+1}, the value of the permissible computation time calculated is $c^k_{r\cdot i}$. Therefore, $c^k_{r\cdot(i+1)}$ is equal to $c^k_{r\cdot i}$ even the other offsets and assignments of τ_{i+1} cannot improve the current value. Thus $c^k_{r\cdot i} \leq c^k_{r\cdot(i+1)}$. Similarly, $c^k_{r\cdot n} \leq c^{k+1}_{r\cdot 1}$ can be proved. □

Theorem 4: The equilibrium-based heuristic converges in at most $\binom{n+p_r}{p_r}n$ *iterations.*

Proof: From Sect. V-C we know, the permissible computation time bound when τ_i is assigned to the processor p is not larger than p_r, i.e., $mc^p_r \leq p_r$. According to Eq. (10), $mc_r = \max_{1\leq p\leq m} mc^p_r \leq p_r$. Thus, the value calculated in each iteration is bounded. Meanwhile, Lemma 1 shows that the permissible computation time calculated has an integer value and increases as the iterative process continues. Therefore, the minimum value that the permissible computation time increases in each iteration is 1. According to the Proposition 4 proposed by Sheikh *et al.* in [29], the heuristic converges and the maximum number of iterations in this heuristic is $\binom{n+p_r}{p_r}n$. □

Now we analyze the complexity of this equilibrium-based heuristic. According to Theorem 4, the heuristic converges in at most $\binom{n+p_r}{p_r}n$ iterations where n is the number of tasks and p_r is the period of the target task. In each iteration, the best response procedure $BR_i(r)$ is adopted to select the best offset and assignment for the considered task. Since $BR_i(r)$ runs in $O(nP_{max})$, the computation complexity of the heuristic is $O\big(n^2 P_{max}\binom{n+p_r}{p_r}\big)$. Since the running time cost of this heuristic depends on not only the number of tasks n, the period of the target task p_r, but also the largest period P_{max}, this heuristic runs in pseudo-polynomial time and its complexity explodes as soon as n becomes large.

VI. MINIMUM PERIOD CALCULATION

Period, as well as the computation time, is one of the most important timing characters in multi-core real-time systems. The decrease of a task period would reduce the time duration between two successive instances and improve the immediateness and accuracy of the data generated. In this section, we study the sensitivity problem in the domain of task periods, and compute the minimum value of the period that a single task can have while keeping the multi-core system still schedulable.

Let $\tau_r = \langle c_r, p_r\rangle$ denote the target task that we want to reduce its period. The value of c_r and the timing parameters of other tasks are fixed. We need to find the minimum value of p_r that ensures the task set is schedulable. The period of τ_r can be optimally decreased to a certain value while the task set is still schedulable. However, any small further decrease in p_r would results in an unschedulable task set. We first give an exact formulation for this minimization problem based on Mixed Integer Quadratically Constrained Programming (MIQCP) [31], then propose a more convenient heuristic based on Game Theory to seek the minimum value of the period of the target task.

A. EXACT FORMULATION

When we calculated the maximum computation time in Sect. V, the periods of all tasks are fixed; however, in this section, the period of the target task is an object we want to minimize and does not remain the same anymore. The greatest common divisor (GCD) of the periods of τ_r and any other task τ_i would be changed. Therefore, when Condition (1) is used to determine whether two tasks are schedulable on the same processor, $g_{i,r}$ is a new variable and $g_{i,r} = GCD(p_i, p_r)$ should be transformed to

$$g_{i,r} \times w_{i,r} = p_r$$
$$g_{i,r} \times v_{i,r} = p_i$$

where $w_{i,r}$ and $v_{i,r}$ are two integer variables representing the quotients from the division operations, i.e., $w_{i,r} = p_r/g_{i,r}$ and $v_{i,r} = p_i/g_{i,r}$.

The optimization process is seeking optimal offset and assignment allocations for all tasks such that the target task τ_r can decrease its period as much as possible while keeping the task set schedulable. The exact formulation can be written as the followings:

$$\begin{aligned}
&\text{minimize } p_r \\
&\text{subject to } \; c_i \leq (s_j - s_i) - g_{i,j} \times e_{i,j} \leq g_{i,j} - c_j \\
&\qquad \frac{c_i - p_i}{g_{i,j}} \leq e_{i,j} \leq \frac{p_j - c_j}{g_{i,j}} \\
&\qquad g_{i,j} \times w_{i,j} = p_j, \; g_{i,j} \times v_{i,j} = p_i \\
&\qquad \forall i, \;\; j \in [1, n], \;\; a_i = a_j, j \neq i \qquad (11) \\
&\qquad p_r \geq c_r \qquad (12) \\
&\qquad a_i \in [1, m], \;\; s_i \in [0, p_i - c_i], \;\; \forall i \in [1, n] \qquad (13)
\end{aligned}$$

Constraint (11) describes the non-overlapping requirements of each two tasks assigned to the same processor. Condition (12) shows that the value of p_r should be larger than or equal to c_r, which is required by the strictly periodic task model proposed in Sect. III. Condition (13) provides the range restrictions of the offsets and assignments of all tasks. Since there are quadratic terms (such as $g_{i,r} \times w_{i,r}$ and $g_{i,r} \times v_{i,r}$) in Constraints (11), this formulation is not a linear program but a mixed integer quadratically constrained one (i.e., MIQCP formulation), which can be solved using Cplex Optimizer or LocalSolver.

In this exact formulation, there are n^2+4n integer variables and requires a significant amount of time to seek optimal offset and assignment allocations for all tasks. Its application to large scale or even fairly complex problem instances may be inefficient because of the required execution times. We propose a faster heuristic based on Game Theory to solve the optimization problem. As did in Sect. V, we first design a procedure to get the best offset s_i for a task τ_i such that τ_r can has the least integral period on a given processor. Then, we try to find the best assignment a_i besides the best offset s_i on a multi-core platform while keeping all other tasks fixed. Finally, we present a heuristic to calculate the minimum value of p_r by moving the tasks round and round until an equilibrium state is reached.

B. BEST OFFSET PROCEDURE ON A PROCESSOR

In this section, we first propose a minimum period procedure $MP(p, r)$ to calculate the least permissible period of τ_r on a given processor p when all tasks are fixed. Then we modify the offset of any task τ_i on the processor p (i.e., $\tau_i \in T_p$), and design a best offset procedure $BP_i^p(r)$ to find an optimal start time for τ_i such that the value of p_r is the minimum according to current allocations. The pseudo-code for $MP(p, r)$ is given in Algorithm 4. $MP(p, r)$ increases p_r from c_r to $LCM(\forall j, \tau_j \in T_p)$ and stops when the permissible computation time calculated is not less than the computation time of τ_r.

Algorithm 4 Minimum Period Procedure $MP(p, r)$

```
   Input: the target task $\tau_r$ and a given processor $p$
   Output: the minimum period $mp_r$ for $\tau_r$
1  $mp_r \leftarrow \infty$;
2  for $p_r = c_r$ to $LCM(\forall j, \tau_j \in T_p)$ do
3      $tc_r \leftarrow LLC(E_{T_p}(r))$;
4      if $tc_r \geq c_r$ then
5          $mp_r \leftarrow p_r$;
6          break;
7      end
8  end
9  return $mp_r$;
```

$MP(p, r)$ seeks the minimum value of p_r on the processor p when the offsets of all tasks keep fixed. Now we consider the situation that only the task τ_i can change its offset freely. Since all tasks in T_p except τ_i (i.e., tasks in T_p^{-i}) remain the same, each valid offset s_i results in a valid task allocation and a possible period bound for the target task τ_r. Therefore, we need to consider all valid values of s_i, compute the associated possible period bound with $MP(p, r)$ and record the minimum value. The best offset procedure $BP_i^p(r)$ performs these operations, and the pseudo-code for this procedure is given in Algorithm 5.

The main computation part of Algorithm 5 is from line 3 to 12, which is a loop and repeats at most p_i times. In each iteration, the minimum period procedure $MP(p, r)$ is used to obtain the least permissible period of τ_r according to the current allocation. Since the complexity of $MP(p, r)$ is $O(n_p p_r l_p)$, where $l_p = LCM(\forall j, \tau_j \in T_p)$, the total computation time of $BP_i^p(r)$ is $O(n_p l_p p_r p_i)$. We use P_{max} to denote the maximum period of all tasks, then the running time complexity of $BP_i^p(r)$ is $O(n_p l_p p_r P_{max})$.

C. BEST STRATEGY PROCEDURE ON A MULTI-CORE PLATFORM

In this section, we design a best strategy procedure $BS_i(r)$ to find the best strategy (i.e., the best offset and the best assignment) for τ_i, such that the target task τ_r can have the minimum period according to the current allocations.

Algorithm 5 Best Offset Procedure $BP_i^p(r)$

```
Input: $\tau_i$, $\tau_r$ and a processor $p$
Output: the minimum value of $p_r$, and the best offset for
        $\tau_i$
1  $minP \leftarrow \infty$; $bs_i \leftarrow -1$;
2  $T_p^{-i} \leftarrow T_p \setminus \{\tau_i\}$;
3  for $k = 0$ to $p_i - c_i$ do
4      $s_i \leftarrow k$;
5      if Condition (6) is satisfied then
6          $tp_r \leftarrow MP(p, r)$;
7          if $tp_r < minP$ then
8              $minP \leftarrow tp_r$;$bs_i \leftarrow k$;
9          end
10     end
11 end
12 return $(minP, bs_i)$;
```

We first remove τ_i from the set T, then the allocations of all tasks left are fixed. We use b_p to denote the permissible period bound of τ_r on the processor p. From Sect. VI-B we know, $b_p = MP(p, r)$. We sort the processors in descending order according to the values of permissible period bounds and use $Q = \{q_1, q_2, \ldots, q_m\}$ to represent the ordered processor set, i.e., $\forall k, l \in [1, m], k \leq l \iff b_{q_k} \geq b_{q_l}$.

Now we begin to assign τ_i to the best processor. Note that the permissible value of p_r on a given processor p tends to increase or remain constant when τ_i is assigned to p. Therefore, we allocate τ_i to the processors in descending order by their permissible period bounds, i.e., from q_1 to q_m in the set Q. When τ_i can be allocated to the processor q_j $(1 \leq j \leq m)$, there are two cases:

(A) If $j < m$, the minimum period mp_r of τ_r is the permissible period bound on the processor q_m, then

$$mp_r = b_{q_m} = MP(q_m, r)$$

(B) If $j = m$, the minimum permissible period of τ_r on the processor q_m should be calculated by the best offset procedure. Meanwhile, the minimum permissible period of τ_r on any other processor is the permissible period bound on the processor q_{m-1}. Since τ_r can be assigned to any processor in a multi-core system, the minimum value of p_r is the smaller one of the two permissible periods. Then

$$mp_r = \min(BP_i^{q_m}(r), b_{q_{m-1}})$$

Putting cases (A) and (B) together, we get the minimum value of p_r when only the offset and assignment of τ_i change. Algorithm 6 shows the pseudo-code for this best strategy procedure. Since the running time complexity of $BP_i^p(r)$ is $O(n_p l_p p_r P_{max})$, the computation complexity of Algorithm 6 is $O(nlp_r P_{max})$, where l is the least common multiple of the periods of all tasks, i.e., $l = LCM(\forall j, \tau_j \in T)$.

Algorithm 6 Best Strategy Procedure $BS_i(r)$

```
Input: $\tau_i$, $\tau_r$ and ordered processor set
       $Q = \{q_1, q_2 \ldots, q_m\}$
Output: the minimum value of $p_r$, and the best offset
        and assignment for $\tau_i$
1  for $p = q_1$ to $q_m$ do
2      $(minP, bs_i) \leftarrow BP_i^p(r)$;
3      if $bs_i \neq -1$ then
4          $ba_i \leftarrow p$;
5          if $p \neq q_m$ then
6              $minP \leftarrow b_{q_m}$;
7          end
8          else
9              if $minP \geq b_{q_{m-1}}$ then
10                 $minP \leftarrow b_{q_{m-1}}$;
11             end
12         end
13         break;
14     end
15 end
16 return $(minP, bs_i, ba_i)$;
```

D. EQUILIBRIUM-BASED HEURISTIC

An equilibrium-based heuristic is proposed to calculate the minimum value of the period of τ_r, by optimizing the offset and assignment of one task at a time while keeping those of other tasks fixed. In this heuristic, tasks take turns to make their best strategy according to the current known allocations. Each task selects the best start time and processor such that τ_r can decrease its period as much as possible.

When no task's offset or assignment can be modified according to the best strategy procedure $BS_i(r)$, an equilibrium state is reached and the heuristic stops. Similar to the notations used in Sect. V-D, $p_{r\cdot i}^k$, s_i^k and a_i^k represent the permissible period calculated for τ_r, the corresponding offset and assignment for τ_i after τ_i uses the best strategy procedure to optimize its allocation in the kth iteration. Algorithm 7 shows the pseudo-code for this equilibrium-based heuristic.

E. CONVERGENCE AND COMPLEXITY ANALYSIS

In this section, we first prove that this equilibrium-based heuristic converges and one or more fixed points would be reached, then we calculate the running time cost of the minimum period algorithm proposed in Sect. VI-D and analyze its influence factors.

Lemma 2: In the iterative processes, there are: $\forall k \geq 1$, $\forall i \in [1, n-1], i \neq r$

$$p_{r\cdot i}^k \geq p_{r\cdot(i+1)}^k \quad and \quad p_{r\cdot n}^k \geq p_{r\cdot 1}^{k+1}$$

Proof: The proof of this lemma is similar to that of the Lemma 1 presented in Sect. V-D. □

Theorem 5: This equilibrium-based heuristic converges in at most $\binom{n+l}{l}n$ iterations, where $l = LCM(\forall j, \tau_j \in T)$.

Algorithm 7 Minimum Period Algorithm (MPA)

```
Input: $\tau_r$ and a task set $T$
Output: the minimum period $minP$ for $\tau_r$
1  $k \leftarrow 0$;
2  repeat
3      for $i = 1$ to $n$ do
4          $p \leftarrow a_i^{k-1}$;
5          $T_p^{-i} \leftarrow T_p \setminus \{\tau_i\}$;
6          $b_p \leftarrow MP(p, r)$;
7          sort $Q$ in descending order according to the
           permissible period bounds;
8          $(p_{r \cdot i}^k, s_i^k, a_i^k) \leftarrow BS_i(r)$;
9          $(s_i, a_i) \leftarrow (s_i^k, a_i^k)$;
10         $minP \leftarrow p_{r \cdot i}^k$;
11     end
12     $k \leftarrow k + 1$;
13 until $T^k = T^{k-1}$;
14 return $minP$;
```

Proof: According to Sect. VI-B, the permissible period calculated cannot be larger than l or smaller than the bound on the processor q_m, i.e., $l \geq p_{r \cdot i}^k \geq b_{q_m} = MP(q_m, r) \geq c_r$. Then, the value calculated after each task τ_i updates its offset and assignment in any iteration is bounded. Meanwhile, from Lemma 2 we know, the period calculated by each step has an integer value and decreases along with the step. Therefore, the minimum value that the permissible period decreases in each step is 1. According to the Proposition 4 proposed by Sheikh *et al.* in [29], the heuristic converges and the maximum number of iterations is $\binom{n+l}{l}n$. □

From Theorem 5 we can find that, the minimum period algorithm converges in at most $\binom{n+l}{l}n$ iterations. In each iteration, the best strategy procedure $BS_i(r)$ is used to find the best strategy for any task τ_i. Since the running time cost of $BS_i(r)$ is $O(nlp_rP_{max})$, the computation complexity of the minimum period algorithm is $O\big(n^2lp_rP_{max}\binom{n+l}{l}\big)$. Therefore, the running time cost of the minimum period algorithm depends on four factors: the number of tasks n, the largest period P_{max}, the least common multiple of the periods of tasks l, and the period of the target task p_r. The complexity of this heuristic explodes as soon as n and l become large.

VII. MAXIMUM SCALING FACTOR CALCULATION

We study the sensitivity problem in the domain of computation times of all tasks in this section. We compute the maximum value of the scaling factor, by which the computation times of all tasks can be multiplied while the system remains or becomes schedulable.

The scaling factor λ is an easily recognized sign of the schedulable state of a task set. For example, in [5], the authors calculated the scaling factor based on game theoretic approach, and used it to provide a determination of whether all tasks in a set were schedulable. If $\lambda \geq 1$, the task set was considered to be schedulable upon a limited number of processors; otherwise, more processors were required.

The scaling factor λ also helps in optimizing the speed of the processors. Assume that S is the original speed of the processors. Scaling the computation times of all tasks by λ is equivalent to changing the speed of the processors to S/λ. In a schedulable system, λ is the maximum slow down factor that the processors can have, and the system remains schedulable as long as the speed of the processors are not less than S/λ. Reciprocally, λ is the minimum speed up factor in an unschedulable system. Only when the speed of processors are increased to S/λ (in this case $\lambda < 1$), the system can be brought back to a schedulable state.

The calculation of the maximum scaling factor is similar with that of the maximum computation time presented in Sect. V. We first give an exact formulation of this constrained maximization problem based on MILP. Then, we propose a more convenient and time saving heuristic by allowing each task to optimize its offset and assignment according to the mostly known allocations.

A. EXACT FORMULATION

In this section, we propose an MILP formulation for calculating the maximum scaling factor. We first analyze the extension process of task computation times when they are scaled. Figure 3 illustrates the impact of λ on the computation times of two tasks assigned to the same processor. Hashed rectangles represent the initial time units occupied by the first instances of the two tasks, whereas the larger filled ones represent the scaled time budgets. Figure 3 (a) extend the computation times of two tasks according to the method proposed by Sheikh *et al.* [5], in which the start times of the instances remain the same but the end times change in accordance with the scaling factor. However, as shown in Fig. 3 (b), the extension process discussed in this paper is different. The computation time of each instance is equally extended from the center to both the left and right side. This means that the centers of the computation time units remain the same; but the start times and end times of the instances are changed when the scaling factor λ is not equal to 1.

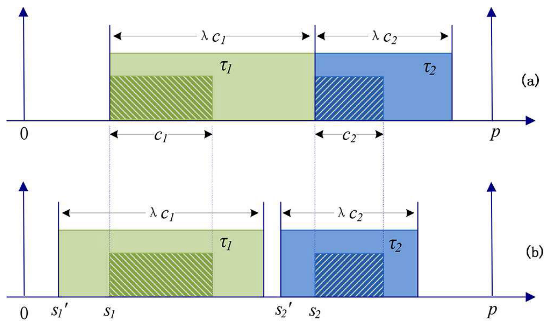

FIGURE 3. Impact of the scaling factor λ on the computation times of two strictly periodic tasks.

Since all computation times are scaled by λ proportionally, for any task τ_i $(1 \leq i \leq n)$, the value of its computation time is changed to λc_i. We use s'_i to denote the start time of τ_i after its computation time has been scaled. From Fig. 3 (b) we know, $2(s_i - s'_i) + c_i = \lambda c_i$, which yields $s'_i = s_i - (\lambda - 1)c_i/2$. Condition (1), which is used to determine whether two original tasks are schedulable on the same processor, should be updated to

$$\lambda c_i \leq (s'_j - s'_i) mod(g_{i,j}) \leq g_{i,j} - \lambda c_j \tag{14}$$

As we pointed out in Sect. V-A, the modulo operation in Condition (14) is not linear and should be replaced by Eq. (5) in the MILP formulation. Then, Condition (14) becomes

$$\begin{gathered} \lambda c_i \leq (s'_j - s'_i) - g_{i,j} \times e_{i,j} \leq g_{i,j} - \lambda c_j \\ \frac{\lambda c_i - p_i}{g_{i,j}} \leq e_{i,j} \leq \frac{p_j - \lambda c_j}{g_{i,j}} \end{gathered} \tag{15}$$

The calculation of the maximum scaling factor is seeking optimal offset and assignment allocations for all tasks, such that the largest possible change in the task computation times can be affordable to satisfy the non-overlapping constraints. The exact formulation can be written as follows:

$$\begin{aligned} &\text{maximize } \lambda \\ &\text{subject to } \lambda c_i \leq (s'_j - s'_i) - g_{i,j} \times e_{i,j} \leq g_{i,j} - \lambda c_j \\ &\qquad \frac{\lambda c_i - p_i}{g_{i,j}} \leq e_{i,j} \leq \frac{p_j - \lambda c_j}{g_{i,j}} \\ &\qquad \forall i,\ j \in [1,n],\ a_i = a_j, j \neq i && (16) \\ &\qquad s'_i = s_i - (\lambda - 1)c_i/2, \quad \forall i \in [1,n] && (17) \\ &\qquad a_i \in [1,m],\ s_i \in [0, p_i - \lambda c_i], \quad \forall i \in [1,n] && (18) \end{aligned}$$

Condition (16) is the non-overlapping constraint of each two tasks distributed to the same processor. Equation (17) provides the start time of each task after its computation time is scaled by λ. Constraint (18) shows the range restrictions of the offsets and assignments of the tasks.

The exact MILP formulation discussed above seeks the maximum scaling factor by searching all possible offset and assignment allocations for the tasks, which is exceedingly laborious and time-consuming. Sometimes the exact formulation fails in supplying any solution within 24 hours, which is not acceptable in practice. Inspired from the Best Response solution presented in Sheikh *et al.* [5] and in Chen *et al.* [14], a highly efficient heuristic is proposed in the following sections.

We first optimize the center (i.e., the middle time point of the first instance) of one task τ_i on a given processor p. Then on a multi-core platform, we try to find the best assignment besides the best center for τ_i to ensure that τ_i has the largest scaling factor λ_i. Finally, tasks take turns to optimize their centers and assignments according to mostly known allocations until an equilibrium is reached. When this calculation stops, the maximum scaling factor is the minimum value of the factors of all tasks, i.e., $\lambda = \min_{1 \leq i \leq n} \lambda_i$.

B. BEST CENTER PROCEDURE ON A PROCESSOR

For each task τ_i ($1 \leq i \leq n$), we design a best center procedure $BC(i, p)$ to find an optimal center o_i such that the computation times of τ_i and all other tasks can be scaled by the largest factor. The offsets, assignments and centers of other tasks remain the same.

Since o_i is the center of τ_i, $o_i = s_i + \frac{c_i}{2} = s'_i + \frac{\lambda c_i}{2}$. τ_i is schedulable with any task τ_j on the same processor if and only if Condition (15) is satisfied. Then, putting the variable o_i into Condition (15), we get

$$\begin{aligned} &\lambda c_i \leq (s'_j - s'_i) - g_{i,j} \times e_{i,j} \leq g_{i,j} - \lambda c_j \\ &\Longrightarrow \lambda c_i \leq (o_j - \frac{\lambda c_j}{2} - o_i + \frac{\lambda c_i}{2}) - g_{i,j} e_{i,j} \leq g_{i,j} - \lambda c_j \\ &\Longrightarrow \lambda c_i + \frac{\lambda(c_j - c_i)}{2} \leq o_j - o_i - g_{i,j} e_{i,j} \leq g_{i,j} - \lambda c_j + \frac{\lambda(c_j - c_i)}{2} \\ &\Longrightarrow \frac{\lambda(c_i + c_j)}{2} \leq o_j - o_i - g_{i,j} e_{i,j} \leq g_{i,j} - \frac{\lambda(c_i + c_j)}{2} \\ &\Longrightarrow \frac{\lambda(c_i + c_j)}{2} \leq (o_j - o_i) mod(g_{i,j}) \leq g_{i,j} - \frac{\lambda(c_i + c_j)}{2} \\ &\Longrightarrow \lambda \leq \min\left(\frac{2(o_i - o_j) mod(g_{i,j})}{c_i + c_j}, \frac{2\big(g_{i,j} - (o_i - o_j) mod(g_{i,j})\big)}{c_i + c_j}\right) \end{aligned} \tag{19}$$

We use $\lambda^p_{i,j}$ to denote the largest scaling factor for τ_i and any task τ_j assigned to the processor p. Then according to Condition (19):

$$\lambda^p_{i,j} = \min\left(\frac{2(o_i - o_j) mod(g_{i,j})}{c_i + c_j}, \frac{2\big(g_{i,j} - (o_i - o_j) mod(g_{i,j})\big)}{c_i + c_j}\right) \tag{20}$$

Now we extend τ_j to all the tasks assigned to the processor p except τ_i (i.e., $\tau_j \in T_p^{-i}$), and use λ^p_i to denote the largest scaling factor that the computation times of all tasks can be multiplied by. Then,

$$\lambda^p_i = \min_{\tau_j \in T_p^{-i}} \lambda^p_{i,j} \tag{21}$$

We use $m\lambda^p_i$ to denote the largest permissible scaling factor when τ_i can changes its center freely on the processor p. Then

$$m\lambda^p_i = \max_{0 \leq o_i \leq p_i} \lambda^p_i = \max_{0 \leq o_i \leq p_i} \min_{\tau_j \in T_p^{-i}} \lambda^p_{i,j} \tag{22}$$

The best center procedure $BC(i, p)$ performs this calculation and stops after all valid values of o_i have been considered. Its pseudo-code is shown in Algorithm 8.

The main computation part of Algorithm 8 is from line 3 to 14, which has a structure of double closed loops. The inner loop (from line 5 to 10) at most repeats n times. Given that the outer loop repeats at most p_i times, the total running time of Algorithm 8 is $O(np_i)$. Since P_{max} is used to denote the maximum period of all tasks, the running time complexity of Algorithm 8 is $O(nP_{max})$.

C. BEST RESPONSE PROCEDURE ON A MULTI-CORE PLATFORM

In this section, we extend the best center procedure $BC(i, p)$ to a multi-core platform, and present a best response procedure $BR(i)$ to find the best assignment besides the best center

Algorithm 8 Best Center Procedure $BC(i, p)$

```
Input: $\tau_i$ and a processor $p$
Output: the largest permissible scaling factor $m\lambda_i^p$, and
        the best center for $\tau_i$
1  $m\lambda_i^p \leftarrow -1; bo_i \leftarrow -1;$
2  $T_p^{-i} \leftarrow T_p \setminus \{\tau_i\};$
3  for $k = 0$ to $p_i$ do
4      $o_i \leftarrow k; t \leftarrow p_i/c_i;$
5      foreach $\tau_j \in T_p^{-i}$ do
6          $\lambda_{i,j}^p = \min\left(\frac{2(o_i-o_j)mod(g_{i,j})}{c_i+c_j}, \frac{2\left(g_{i,j}-(o_i-o_j)mod(g_{i,j})\right)}{c_i+c_j}\right);$
7          if $\lambda_{i,j}^p < t$ then
8              $t \leftarrow \lambda_{i,j}^p;$
9          end
10     end
11     if $t > m\lambda_i^p$ then
12         $m\lambda_i^p \leftarrow t; bo_i \leftarrow k;$
13     end
14 end
15 return $(m\lambda_i^p, bo_i);$
```

for a given task τ_i. The best assignment and the best center guarantee that τ_i has the largest scaling factor according to the current allocations.

From Sect. VII-B we know, when τ_i is assigned to the processor p, the permissible factor by which the computation times of all tasks can be multiplied, is $m\lambda_i^p$ and can be calculated by the best center procedure $BC(i, p)$. In order to choose the best assignment, we need to compute the permissible factor on each processor and select the largest one. We use λ_i to denote the maximum permissible scaling factor for τ_i when only the center and assignment of τ_i change on a multi-core platform, then

$$\lambda_i = \max_{1 \le p \le m} m\lambda_i^p \tag{23}$$

The pseudo-code for this best response procedure is given in Algorithm 9. Since the best center procedure $BC(i, p)$ has a complexity of $O(nP_{max})$, the complexity of the best response procedure $BR(i)$ is $O(mnP_{max})$.

D. EQUILIBRIUM-BASED HEURISTIC

Now we present a heuristic to calculate the maximum scaling factor for the computation times of all tasks based on Game Theory. We think of tasks as players and their strategies are the modification of their centers and assignments. All tasks take turns to use the best response procedure $BR(i)$ to update their strategies such that their computation times can be scaled as much as possible. When no task in the set T can improve its center or assignment using the best response procedure, an equilibrium state is reached and the iterative process stops. At this time, the maximum scaling factor λ

Algorithm 9 Best Response Procedure $BR(i)$

```
Input: $\tau_i$ in a task set $T$
Output: the largest factor $\lambda_i$ for $\tau_i$, the corresponding
        center $bo_i$ and assignment $ba_i$
1  $\lambda_i \leftarrow 0; bo_i \leftarrow -1; ba_i \leftarrow -1;$
2  for $p = 1$ to $m$ do
3      $(t, to_i) \leftarrow BC(i, p);$
4      if $t > \lambda_i$ then
5          $\lambda_i \leftarrow t; bo_i \leftarrow to_i; ba_i \leftarrow p;$
6      end
7  end
8  return $(\lambda_i, bo_i, ba_i);$
```

is the minimum value of the permissible factors of all tasks. i.e., $\lambda = \min_{1 \le i \le n} \lambda_i$.

We use λ_i^k, o_i^k and a_i^k to denote the permissible scaling factor, the corresponding center and assignment obtained from the best response procedure $BR(i)$ when τ_i update its allocation in the kth ($k \ge 1$) iteration. As the authors did in Sheikh *et al.* [5], we assume that τ_i does not change its center or assignment if the best response procedure does not improve its current scaling factor. That is to say: if $\lambda_i^k \le \lambda_i^{k-1}$, $o_i^k = o_i^{k-1}$ and $a_i^k = a_i^{k-1}$. The pseudo-code for this heuristic is given in Algorithm 10.

Algorithm 10 Maximum Scaling Factor Algorithm (MSFA)

```
Input: a task set $T$
Output: the maximum scaling factor $\lambda$ for all tasks
1  $k \leftarrow 1;$
2  repeat
3      for $i = 1$ to $n$ do
4          $(\lambda_i^k, bo_i, ba_i) \leftarrow BR(i);$
5          if $\lambda_i^k > \lambda_i^{k-1}$ then
6              $o_i \leftarrow bo_i; a_i \leftarrow ba_i;$
7          end
8          else
9              $o_i^k \leftarrow o_i^{k-1}; a_i^k \leftarrow a_i^{k-1};$
10         end
11     end
12     $k \leftarrow k + 1;$
13 until $T^k = T^{k-1};$
14 $\lambda \leftarrow \min_{1 \le i \le n} \lambda_i^k;$
15 return $\lambda$
```

E. CONVERGENCE AND COMPLEXITY ANALYSIS

According to the Proposition 4 presented in Sheikh *et al.* [29], this heuristic converges and reaches one or more fixed points in at most $\binom{n+h}{h}n$ iterations where h is the maximum number of increasing steps and calculated by $h = \lceil \alpha_{max}\Delta^{-1} \rceil$, $\alpha_{max} = \max_i \min_{j \ne i} \frac{g_{i,j}}{c_i+c_j}$ and $\Delta = \min_{j,k} \frac{1}{lcm(c_j,c_k)}$. In each iteration, the best response procedure $BR(i)$ is used to select the best center and assignment. As we pointed out in Sect. VII-C, the best response procedure $BR(i)$ runs in $O(mnP_{max})$. Hence, the running time complexity of the heuristic is $O\big(mn^2P_{max}\binom{n+h}{h}\big)$.

The running time cost of this heuristic depends on the number of processors m, the number of tasks n, the largest period P_{max}, and the maximum number of increasing steps h calculated according to the periods of all tasks. Similarly, this heuristic runs in pseudo-polynomial time and its complexity explodes as soon as n becomes large. It cannot be deduced that our heuristic is better than MILP or other solutions whose computational complexity is NP-Complete. However, the bound on the worst case number of steps of our heuristic considers a very pessimistic situation, and our heuristic typically performs better to solve the allocation problem on the same task sets.

VIII. EXPERIMENTAL RESULTS

In this section, we conduct simulation experiments to evaluate the performance of the proposed heuristics. We compare our experimental results with those of the exact MILP or MIQCP formulations.

The proposed heuristics have a wide range of applications and can be adapted to tasks with both harmonic and non-harmonic periods. However, all of them provide approximate but not exact solutions because they are based on Game Theory and stop when equilibrium states are reached. Compared with the exact formulations, they do not require completely searching the solution space and take less time to find solutions.

We evaluate the performances of our approaches from two aspects: time consumption which shows the speeds of the heuristics, and relative error ratios which represent the errors on results calculated by the heuristics. In addition, we put a 300 seconds time limit on the exact formulation execution of each input task set, for little improvements on the solutions could be made even the solver remains running to the end. In the case of a time out, the solution of the exact formulation is the best solution found up until the time limit.

A. TASK GENERATION

Tasks adopted in experiments were generated with the UUnifast-Discard algorithm [32], which is a simple extension of UUniFast [33] on a multi-core platform and produces a random utilization for each task under a given system utilization. There were three parameters for each task set: the number of tasks n, the system utilization u and the task type (harmonic or non-harmonic).

The generation procedure was the same as that described in Chen *et al.* [27]. First we adopted the UUnifast-Discard algorithm to produce a random utilization u_i $(1 \le i \le n)$ for each task τ_i, i.e., $u = \sum_{1 \le i \le n} u_i$. Then, we chose a random value p_0 from 5 to 9 as a base period, i.e., $p_0 = U[5, 9]$. Subsequently, for non-harmonic tasks, periods were chosen randomly from the set $\{2^x 3^y 5^z p_0 : x, y, z \in [0, 3]\}$, which was inspired from Eisenbrand *et al.* [34]. For harmonic tasks, p_0 was the period of the first task and a period ratio k_i $(1 < i \le n)$ was selected randomly from [1,5]. The periods of the harmonic tasks were constructed as $p_i = k_i p_{i-1}$. Finally, the computation time of each task was constructed as $c_i = p_i u_i$.

B. MAXIMUM COMPUTATION TIME ALGORITHM EVALUATION

We start by evaluating the performance of the maximum computation time algorithm (MCTA) proposed in Sect. V-D in terms of execution time. With a logarithmic scale, Figure 4 shows the execution times required to compute the sensitivity in the domain of the computation times with different methods. The task sets were generated when the system utilization was 0.5. ''MILP_H'', ''MILP_NH'', ''MCTA_H'' and ''MCTA_NH'' represent the average times required by the MILP formulation and our approach for tasks with harmonic and non-harmonic periods.

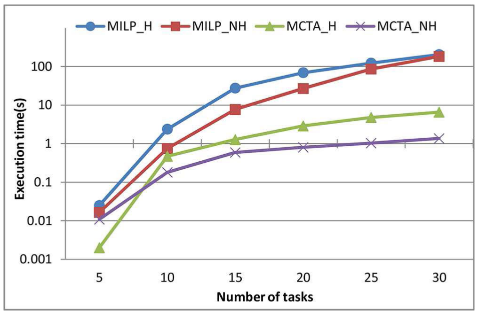

FIGURE 4. Execution times required for the sensitivity analysis of computation times using the MILP formulation and the maximum computation time algorithm (MCTA).

We can see that the time consumption of two solutions has a similar changing tendency that it grows gradually along with the increase of the number of tasks. This is because that the more tasks are tested on in the experiments, the more possible offset and assignment allocations should be considered to find an optimal solution. Compared with the MILP formulation, our approach has lower time consumption and a smoother growth because it considers the tasks one by one and does not completely search the solution space. When the number of tasks is 30, the execution times required by our approach for harmonic and non-harmonic tasks are 6.55 and 1.36 seconds, which are 32 and 133 times less than those required by the MILP formulation respectively. This demonstrates that our method is faster in analyzing the sensitivity of computation times.

In the experiment of Fig. 5, the speed of our approach is evaluated by varying system utilizations under which non-harmonic tasks are generated. We can find that the time consumption of our approach grows along with the increase of the number of tasks and the system utilizations. As pointed out in Sect. V, our approach has a pseudo-polynomial complexity, which depends on not only the number of tasks n, but also the largest period P_{max}. Since a larger system utilization leads to

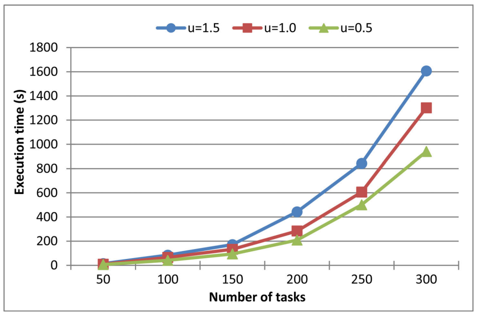

FIGURE 5. Time consumption of our approach (MCTA) on non-harmonic tasks generated with different system utilizations.

the generation of tasks with larger periods, the time consumption conforms with the computation complexity analyzed in Sect. V.

Figure 6 shows the relative errors of our approach on the harmonic tasks through varying the number of tasks and the system utilizations. The relative error ratio er represents the relative difference between the computation time calculated by our approach and by the exact formulation, i.e., $er = (MILP - MCTA)/MILP \times 100\%$. The number of tasks increases from 5 to 30 and the system utilization is tuned from 0.5 to 1.5. We can see that, when the number of tasks is 15, the largest error ratio appears and its value is 7.1%.

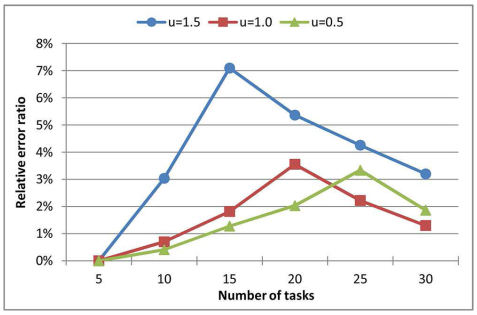

FIGURE 6. Relative error ratios of the values of computation times calculated by our approach on harmonic tasks.

In Fig. 6, all of the three curves have essentially the same changing tendency that they first increase from zero to the maximum values and then drop down gradually. This is because our approach is inspired from Game Theory and just provides approximated solutions. As the number of tasks grows, there are more and more possible allocations that our approach does not consider, thus the relative error ratio increases in the first phase. Meanwhile, the increasing number of tasks results in an explosive growth in solution space of the constrained maximization problem. There are more and more task sets that the MILP formulation fails to find optimal solutions under a time limit. Then, according to the definition of er, the error ratios decrease in the second phase.

In order to quantify the amount of variation of the relative error ratios of our approach, we record the relative error ratio on each task set, select the minimum and the maximum values, and calculate the standard deviations. Table 2 shows the relative error ratios statistic of our approach on harmonic tasks when the system utilization is 0.5. As can be seen that, when the number of tasks is 25, the largest average relative error ratio and the largest standard deviation appear, and their values are 0.0334 and 0.1855 respectively.

TABLE 2. Relative error ratio statistic of our approach (MCTA) on harmonic tasks when the system utilization is 0.5.

Task number	5	10	15	20	25	30
Minimum value	0	0	0	0	0.0005	0.0013
Maximum value	0.0021	0.1822	0.9778	0.9667	0.9903	0.3447
Average value	0.0001	0.0041	0.0127	0.0203	0.0334	0.0186
Standard deviation	0.0005	0.0248	0.0985	0.1033	0.1855	0.0607

C. MINIMUM PERIOD ALGORITHM EVALUATION

In this section, we conduct experiments to evaluate the time consumption and relative error ratios of the minimum period algorithm (MPA) proposed in Sect. VI-D. The exact method, which our approach is compared with, is an MIQCP formulation and presented in Sect. VI-A.

In the experiment of Fig. 7, tasks were generated when the system utilization was 1.5, and we recorded the execution times required to analyze the sensitivity in the domain of the periods. The values of "MIQCP_NH", "MIQCP_H", "MPA_NH" and "MPA_H" represent the average execution times required by the MIQCP formulation and our approach for tasks with non-harmonic and harmonic periods.

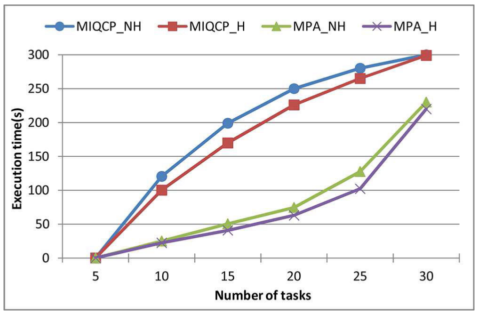

FIGURE 7. Execution times required for the sensitivity analysis of periods using the MIQCP formulation and the minimum period algorithm (MPA).

Similar to the results shown in Fig. 4, the execution times of the two solutions grow gradually along with the increase of the number of tasks, no matter the tasks are with harmonic or non-harmonic periods. Our approach has a better performance than the MIQCP formulation in terms of time consumption when they are used on the same task sets. When the number of tasks is 20, the execution times required by our approach for harmonic and non-harmonic tasks are 63 and 74 seconds, which are 3.6 and 3.4 times less than those required by the MIQCP formulation respectively.

In order to quantify the amount of variation of the time consumption of the minimum period algorithm proposed, we record the time consumption results on each task sets, select the minimum and the maximum values, and calculate the standard deviations. The statistical results on harmonic tasks generated with $u = 1.5$ are shown in Table 3. We can find that, along with the increase of the number of tasks, the average time cost and standard deviations become higher.

TABLE 3. Time consumption statistic of our approach (MPA) on harmonic tasks when the system utilization is 0.5.

Task number	5	10	15	20	25	30
Minimum time (s)	0.01	0.01	0.08	0.10	1.28	1.39
Maximum time (s)	1.83	242.15	289.99	319.61	577.92	759.89
Average time (s)	0.09	22.43	48.37	63.04	102.31	227.89
Standard deviation	0.25	58.57	65.38	108.36	150.87	332.65

Figure 8 demonstrates the time consumption of our minimum period algorithm on the task sets that are determined schedulable on a four processors platform according to MIQCP. Tasks are generated with harmonic periods and different system utilizations. As expected, the time consumption becomes higher when more tasks are tested on. When the number of tested tasks is fixed, the time cost of our approach grows with the increase of the system utilization. The time consumption of our approach explodes as soon as the number of tasks n and the system utilization become large.

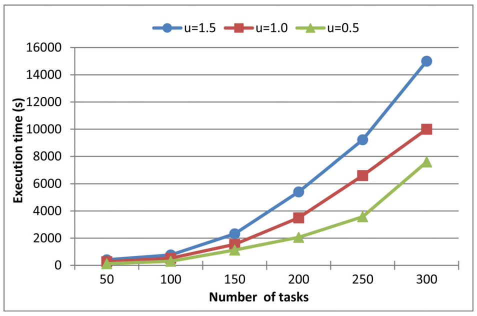

FIGURE 8. Time required by our approach (MPA) to calculate the minimum periods of harmonic tasks.

Through varying the system utilizations, the relative error ratios of periods computed by our approach are shown in Fig. 9. The tasks are chosen from non-harmonic types and the system utilization is tuned from 0.5 to 1.5. As can be seen that all of the three curves grow first and then drop down along with increase of the number of tasks. The reason is similar to that we explained in the previous section. We can also find that when the number of tasks is fixed, the results of our approach on the task instances with $u = 0.5$ have the lowest relative error ratios, which means that the values of periods calculated by our approach on this series of task sets are the closest to the exact solutions.

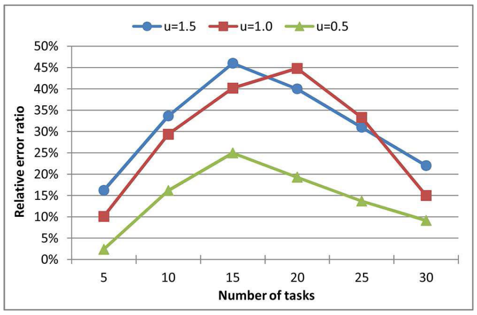

FIGURE 9. Relative error ratios of the values of periods calculated by our approach.

D. MAXIMUM SCALING FACTOR ALGORITHM EVALUATION

The final experiment we carried out is calculating the sensitivity of scaling factors for the computation times of all tasks. As did in Sect. VIII-B and VIII-C, we show the performance of our algorithm from two aspects: time consumption and relative error ratio.

With a logarithmic scale, Fig. 10 shows the execution times used to analyze the sensitivity of scaling factors by our approach (MSFA) and the MILP formulation. The task sets were generated when the system utilization was 1.0. The fields of "MILP_H", "MILP_NH", "MSFA_H" and "MSFA_NH" represent the average execution times required by the MILP formulation and our approach on tasks chosen from harmonic and non-harmonic period types.

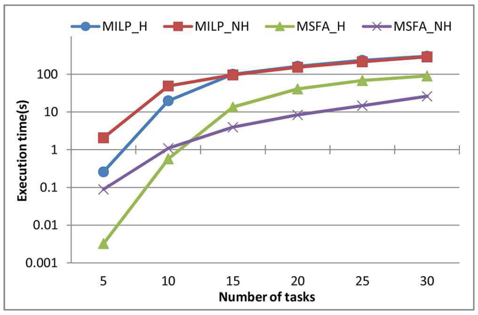

FIGURE 10. Execution times required for the sensitivity analysis of scaling factors using the MILP formulation and the maximum scaling factor algorithm (MSFA).

From Fig. 10 we can know that, our methods result in much shorter times for all experiments, not only in the domains of computation times and periods, but also in the domain of scaling factors. For each task set, our approach has a better performance than the MILP formulation in terms of time consumption. When the number of tasks is 10, the execution time required by our approach for harmonic and non-harmonic tasks are 0.5 and 1.1 seconds, which are 34 and 45 times less than those required by the MILP formulation.

Figure 11 shows the time cost of our approach (MSFA) in calculating the maximum scaling factor for tasks with non-harmonic periods. The number of tasks increases from 50 to 300 and the system utilization is tuned from 0.5 to 1.5. Similar to the results shown in Fig. 5 and Fig. 8, the time consumption of our approach grows along with the increase of the number of tasks and the system utilizations. As can

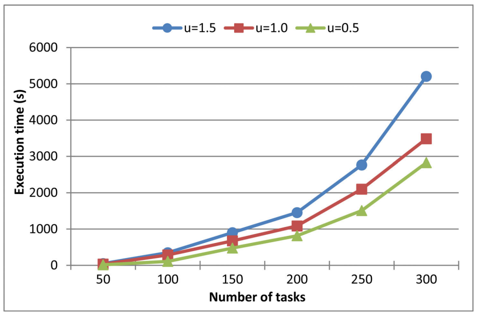

FIGURE 11. Execution times required by our approach (MSFA) to calculate the maximum scaling factor for non-harmonic tasks with different system utilizations.

be seen that the time cost explodes as soon as the number of tasks n becomes large. Meanwhile, when the number of tasks is fixed, a larger system utilization leads to a higher time cost. As pointed out in Sect. VII, our approach runs in pseudo-polynomial time and its complexity depends the number of tasks and the largest period P_{max}. The results conforms with the computation complexity analyzed in Sect. VII.

Figure 12 demonstrates the relative error ratios on the scaling factors calculated by our approach with respect to the MILP formulation. The tasks were generated with harmonic periods under different system utilizations. For all of the considered task instances, the relative error ratios on the results of our approach, as compared to the exact solutions, remain below 7%. The largest relative error ratio appears when the number of tasks is 10 and its value is 6.5%. At the same time, the time consumption of our approach is 0.9 seconds, which is 73 times less than that of the MILP formulation. The results are within the allowable range and can be accepted when taking into account the time cost.

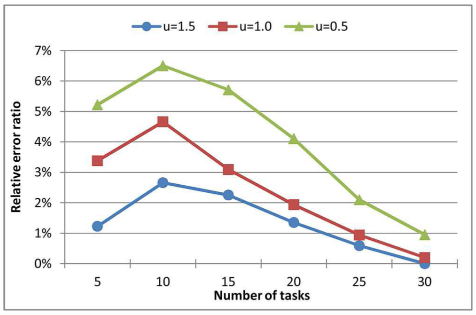

FIGURE 12. Relative error ratios of the values of scaling factor calculated by our approach on harmonic tasks.

Table 4 shows the time relative error ratio statistic of our approach on harmonic tasks when the system utilization is 1.0. Similar to the results shown in Table 2, the relative error ratios and standard deviations change along with the increase of the number of tasks. When the number of tasks is 10, the largest average relative error ratio and the largest standard deviation appear, and their values are 0.0466 and 0.1050 respectively. The results indicate that the relative error ratios of our approach are spread out over a wider range of values.

TABLE 4. Relative error ratio statistic of our approach (MFSA) on harmonic tasks generated with $u = 1.0$.

Task number	5	10	15	20	25	30
Minimum value	0	0.0006	0.0004	0.0012	0.0001	0
Maximum value	0.3846	0.5921	0.4452	0.2265	0.2401	0.4762
Average value	0.0337	0.0466	0.0309	0.0194	0.0091	0.0008
Standard deviation	0.0758	0.1050	0.0811	0.0539	0.0371	0.0627

IX. CONCLUSIONS AND FUTURE STUDIES

In this paper, we aimed at analyzing the possible changes in task computation times, periods and scaling factors on a multi-core platform in real-time systems. The tasks have strict periods and can be non-preemptively scheduled with proper offset and assignment allocations. Using the multi-task sensitivity analysis, we presented heuristics to compute the maximum computation time and the minimum period for a single task, and obtain the maximum scaling factor for the computation times of all tasks. Experiments with randomly generated task sets show that our approach is faster and more efficient than the existing solutions to solve the sensitivity problems in large scale multi-core real-time systems.

There are two possible directions for our future work. First, although it has been shown that our heuristics can provide reasonable solutions in a relatively short amount of time, we would like to consider some aggressive notions of approximation and see whether or not the heuristics' performance could be improved. Second, we are interested in studying the schedulability and sensitivity analysis of non-strictly periodic tasks, and would like to see whether some of the results in this paper can be used to find the permissible changes in the timing parameters of non-strictly periodic tasks.

REFERENCES

[1] E. Bini, M. Di Natale, and G. Buttazzo, "Sensitivity analysis for fixed-priority real-time systems," *Real-Time Syst.*, vol. 39, nos. 1–3, pp. 5–30, Aug. 2008.

[2] R. Racu, M. Jersak, and R. Ernst, "Applying sensitivity analysis in real-time distributed systems," in *Proc. 11th IEEE Real Time Embedded Technol. Appl. Symp.*, Mar. 2005, pp. 160–169.

[3] F. Zhang, A. Burns, and S. Baruah, "Sensitivity analysis of arbitrary deadline real-time systems with EDF scheduling," *Real-Time Syst.*, vol. 47, no. 3, pp. 224–252, 2011.

[4] F. Eisenbrand, N. Hähnle, M. Niemeier, M. Skutella, J. Verschae, and A. Wiese, "Scheduling periodic tasks in a hard real-time environment," in *Proc. 37th Int. Colloq. Automata, Lang., Programming* Berlin, Germany: Springer, 2010, pp. 299–311.

[5] A. A. Sheikh, O. Brunl, P. L. Hladik, and B. J. Prabhu, "Strictly periodic scheduling in IMA-based architectures," *Real-Time Syst.*, vol. 48, no. 4, pp. 359–386, Jul. 2012.

[6] O. Kermia and Y. Sorel, "Schedulability analysis for non-preemptive tasks under strict periodicity constraints," in *Proc. 14th IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2008, pp. 25–32.

[7] P. Tendulkar, P. Poplavko, and O. Maler, "Strictly periodic scheduling of acyclic synchronous dataflow graphs using SMT solvers," Verimag, Saint-Martin-d'Hères, France, Tech. Rep. TR-2014-5, 2014.

[8] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *Proc. Real-Time Syst. Symp.*, Dec. 1989, pp. 166–171.

[9] S. Rani, R. Talwar, J. Malhotra, S. H. Ahmed, M. Sarkar, and H. Song, "A novel scheme for an energy efficient Internet of things based on wireless sensor networks," *Sensors*, vol. 15, no. 11, pp. 28603–28626, 2015.

[10] L. Ren, H. Liao, M. Castillo-Effen, B. Beckmann, and T. Citriniti, "Chapter 22-transformation of mission-critical applications in aviation to cyber-physical systems," in *Cyber-Physical System*. Jan. 2017, pp. 339–362.

[11] M. Shojafar, S. Javanmardi, S. Abolfazli, and N. Cordeschi, "FUGE: A joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method," *Cluster Comput.*, vol. 18, no. 2, pp. 829–844, 2015.

[12] R. Racu, A. Hamann, and R. Ernst, "Sensitivity analysis of complex embedded real-time systems," *Real-Time Syst.*, vol. 39, nos. 1–3, pp. 31–72, 2008.

[13] S. K. Baruah and S. Chakraborty, "Schedulability analysis of non-preemptive recurring real-time tasks," in *Proc. 20th IEEE Int. Parallel Distrib. Process. Symp.*, Apr. 2006, p. 8.

[14] J. Chen, C. Du, and P. Han, "Scheduling independent partitions in integrated modular avionics systems," *PLoS ONE*, vol. 11, no. 12, 2016, Art. no. e0168064.

[15] C. Pira and C. Artigues, "Line search method for solving a non-preemptive strictly periodic scheduling problem," *J. Scheduling*, vol. 19, no. 3, pp. 227–243, Jun. 2016.

[16] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proc. 22nd IEEE Real-Time Syst. Symp.*, Dec. 2001, pp. 95–105.

[17] S. Vestal, "Fixed-priority sensitivity analysis for linear compute time models," *IEEE Trans. Softw. Eng.*, vol. 20, no. 4, pp. 308–317, Apr. 1994.

[18] S. Punnekkat, R. Davis, and A. Burns, "Sensitivity analysis of real-time task sets," in *Advances in Computing Science* (Lecture Notes in Computer Science), vol. 1345, R. Shyamasundar and K. Ueda, Eds. Berlin, Germany: Springer, 1997, pp. 72–82.

[19] S. Baruah and G. Fohler, "Certification-cognizant time-triggered scheduling of mixed-criticality systems," in *Proc. IEEE 32nd Real-Time Syst. Symp.*, Nov./Dec. 2011, pp. 3–12.

[20] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. New York, NY, USA: Springer, 2011.

[21] J. Korst, E. Aarts, J. Lenstra, and J. Wessels, "Periodic multiprocessor scheduling," in *Parallel Architectures and Languages Europe* (Lecture Notes in Computer Science), vol. 505, E. Aarts, J. van Leeuwen, and M. Rem, Eds. Berlin, Germany: Springer, 1991, pp. 166–178.

[22] M. Marouf and Y. Sorel, "Scheduling non-preemptive hard real-time tasks with strict periods," in *Proc. IEEE 16th Conf.*, Sep. 2011, pp. 1–8.

[23] T. Zhang, N. Guan, Q. Deng, and W. Yi, "Start time configuration for strictly periodic real-time task systems," *J. Syst. Archit.*, vols. 66–67, pp. 61–68, May 2016.

[24] J. Chen, C. Du, F. Xie, and B. Lin, "Scheduling non-preemptive tasks with strict periods in multi-core real-time systems," *Journal Syst. Archit.*, vol. 90, pp. 72–84, Oct. 2018.

[25] G. H. Golub and C. F. Van Loan, *Matrix computations. 1996*. Baltimore, MD, USA: Johns Hopkins Univ., Press, 1996, pp. 374–426.

[26] J. Chen, C. Du, F. Xie, and B. Lin, "Allocation and scheduling of strictly periodic tasks in multi-core real-time systems," in *Proc. IEEE 22nd Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2016, pp. 130–138.

[27] J. Chen, C. Du, F. Xie, and Z. Yang, "Schedulability analysis of non-preemptive strictly periodic tasks in multi-core real-time systems," *Real-Time Syst.*, vol. 52, no. 3, pp. 239–271, May 2016.

[28] A. Lodi and J. Linderoth, "Milp software," *Encyclopedia of Operations Research and Management Science*. Hoboken, NJ, USA: Wiley, 2011.

[29] A. A. Sheikh, O. Brun, P. Hladik, and B. J. Prabhu, "A best-response algorithm for multiprocessor periodic scheduling," in *Proc. 23rd Euromicro Conf. Real-Time Syst.*, Jul. 2011, pp. 228–237.

[30] C. Pira and C. Artigues, "An efficient best response heuristic for a non-preemptive strictly periodic scheduling problem," in *Learning and Intelligent Optimization* (Lecture Notes in Computer Science), G. Nicosia and P. Pardalos, Eds. Berlin, Germany: Springer, 2013, pp. 281–287.

[31] S. Burer and A. Saxena, "The MILP road to MIQCP," in *Mixed Integer Nonlinear Programming* (The IMA Volumes in Mathematics and its Applications), vol. 154, J. Lee and S. Leyffer, Eds. New York, NY, USA: Springer 2012, pp. 373–405.

[32] R. I. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," *Real-Time Syst.*, vol. 47, no. 1, pp. 1–40, 2011.

[33] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, nos. 1–2, pp. 129–154, May 2005.

[34] F. Eisenbrand, K. Kesavan, and R. Mattikalli, "Solving an avionics real-time scheduling problem by advanced IP-methods," in *Algorithms* (Lecture Notes in Computer Science), vol. 6346, M. de Berg and U. Meyer, Eds. Berlin, Germany: Springer, 2010, pp. 11–22.

JINCHAO CHEN received the Ph.D. degree in computer science from Northwestern Polytechnical University, Xi'an, China, in 2016, where he is currently an Assistant Professor with the Department of Computer Science. His research interests include multiprocessor scheduling theory, real-time systems design, and modeling and verification of embedded systems.

CHENGLIE DU received the Ph.D. degree in computer science from Northwestern Polytechnical University, China, in 1999, where he is currently a Professor with the Department of Computer Science. His research and teaching interests include scheduling theory, real-time distributed computing systems, design and verification of cyber-physical systems, and domain software engineering.

PENGCHENG HAN received the M.S. degree in computer science from Northwestern Polytechnical University, Xi'an, China, in 2015, where he is currently pursuing the Ph.D. degree with the Department of Computer Science. His research interests include parallel and distributed systems, workflow scheduling, resource management, and cloud computing.

YONG ZHANG is currently a Professor and the Head of the Department of Software System Development, North Automatic Control Technology Institute, Taiyuan, China. His research interests include parallel and distributed systems, scheduling theory, real-time-embedded systems, control systems, and resource management.