Improved Multi-Core Real-Time Task Scheduling of Reconfigurable Systems With Energy Constraints

HAMZA CHNITER1,2, OLFA MOSBAHI1,2, MOHAMED KHALGUI1,2, MENGCHU ZHOU3,4, (Fellow, IEEE), AND ZHIWU LI5,6, (Fellow, IEEE)

1School of Electrical and Information Engineering, Jinan University, Zhuhai 519070, China
2National Institute of Applied Sciences and Technology (INSAT), University of Carthage, Tunis 1080, Tunisia
3Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, USA
4Center of Research Excellence in Renewable Energy and Power Systems, King Abdulaziz University, Jeddah 21589, Saudi Arabia
5Institute of Systems Engineering, Macau University of Science and Technology, Taipa 999078, Macau
6School of Electro-Mechanical Engineering, Xidian University, Xi’an 710071, China

Corresponding authors: Mohamed Khalgui (khalgui.mohamed@gmail.com) and Zhiwu Li (zhwli@xidian.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1700104, in part by the National Natural Science Foundation of China under Grant 61873342 and Grant 61472295, in part by the Recruitment Program of Global Experts, in part by the Science Technology Development Fund, Macao Special Administrative Region (MSAR), under Grant 0012/2019/A1, and in part by the Deanship of Scientific Research (DSR) with King Abdulaziz University, Jeddah, under Grant RG-48-135-40.

ABSTRACT This paper deals with the scheduling of real-time periodic tasks executed on heterogeneous multicore platforms. Each processor is composed of a set of multi-speed cores with limited energy resources. A reconfigurable system is sensible to unpredictable reconfiguration events from related environment, such as the activation, removal or update of tasks. The problem is to handle feasible reconfiguration scenarios under energy constraints. Since any task can finish execution before achieving its worst-case execution time (WCET), the idea is to distribute this execution on different processor cores for meeting related deadlines and reducing energy consumption. The methodology consists in using lower processor speeds first to consume less energy. If the system is still non-feasible after reconfiguration, then we adjust the task periods as a flexible solution or migrate some of them to the least loaded processors. Accordingly, an integer linear program (ILP) is formulated to encode the execution model that assigns tasks to different cores with optimal energy consumption, thereby realizing energy-efficient computing/green computing. The potency and effectiveness of the proposed approach are rated through simulation studies. By measuring the energy consumption cost, our solution offers better than 11% of gain than recently published methods and improves by 85% the overall number of adjusted periods.

INDEX TERMS Green computing, integer linear program, low power consumption, multi-core processor, optimization real-time scheduling, reconfigurable systems reconfiguration.

NOMENCLATURE

- $m$ Number of networked devices
- $n$ Number of tasks in a considered system
- $c$ Number of cores of each processor
- $h$ Number of segments of each task
- $T$ Set of periodic tasks
- $T_i$ $i$-th periodic task
- $T_i^j$ $j$-th segment of task $T_i$
- $Device_p$ $p$-th device
- $Proc_p$ $p$-th processor
- $Core_{kp}$ $k$-th processor of $Proc_p$
- $f_{kp}$ Frequency of $Core_{kp}$
- $W_{kp}$ Weight of $Core_{kp}$
- $\hat{F}_{kp}$ Normalized frequency of $Core_{kp}$
- $\hat{V}_{kp}$ Normalized voltage of core $Core_{kp}$
- $F_{ikp}$ Effective execution frequency of tasks $T_i$ in $Core_{kp}$
- $V_{ikp}$ Effective execution voltage of tasks $T_i$ in $Core_{kp}$
- $C_{ikp}$ Computation time of tasks $T_i$ in $Core_{kp}$
- $d_i$ Absolute deadline of task $T_i$
- $\pi_i$ Period of task $T_i$
- $\pi_i^M$ Upper bound of period $\pi_i$
- $\zeta_{kp}$ Computation time at the normalized frequency of task $T_i$ in $Proc_p$

The associate editor coordinating the review of this manuscript and approving it for publication was Meng-Lin Ku5.
components, and subsequently some instability problems can
cause each system’s task can cause problems for hardware
frequently a processor’s operating frequency at run-time to exe-
reducing power consumption cost. However, changing fre-
change its execution frequency of tasks with the aim of
amount of work, this technology enables the processor to
considered as the most adopted way to implement an effica-
more processing power to meet the requirements of such
real-time applications became hugely complex and needed
the violation of some task deadlines. During the last decade,
can grow the energy consumption or lead to
devices when a
an activation/deactivation of one or more devices when a
A reconfiguration scenario can also be considered as
in response to user requirements and dynamic changes in
their environment during execution such as unpredictable
activation-deactivation-update of some tasks [1], [3], [4].
order between segments $T_i^s$ and $T_j^s$. $r_{skp}$ Release time of segment $T_i^s$ in $Core_{kp}$
Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
Device $P_i$ Fractional increase of the period of task $T_i$
technologies which can affect the stability of the processors [8]. This is due to the problem of switching processors between different power consumption modes and scaling the speed of the processor according to the workload in a system.

Some studies ([1], [28]) integrate multi-agent architectures requiring fairly good communications among agents, which can increase computational cost. Thus, some [29]–[32] propose models which are linked to a set of precise criteria to an explicit application framework, well-defined input parameters and predetermined environmental constraints which make its improvement difficult.

The developed solution presents an optimization mechanism that affects all the levels, such as partitioning tasks on different cores, and adjusting periods and migrating tasks from one processor to another. This makes the system more flexible and able to handle run-time reconfigurations. Moreover, the produced solutions are optimal. It allows engineers to determine, after each reconfiguration, the assignment of the tasks to different cores by acting on multi-speed cores and specifies their scenarios of migration among processors. Migrating tasks from an over-loaded processor to a less loaded one not only allows engineers to find a schedule that guarantees the feasibility of the system, but also balances the workload among all processors.

An assignment strategy consists in finding an optimal hard real-time workload split among the existing cores. Finding an optimal solution for this problem is NP-hard [20]. We adopt a mathematical formulation based on integer linear program (ILP) to define the execution model that deals with assignment. Since the objective function is rational, it is not possible to solve the initial developed program directly and efficiently by any solver. The non-linearity of the developed program stems from the fact that the execution time of tasks is proportional to the weight of cores whereas energy consumption is inversely proportional to the weight of cores. We propose an approach that includes two stages to solve it. First we transform it into a quadratic program [26]. Then, we transform the quadratic program into a linear one.

The proposed approach is compared with a set of methods reported in [1], [7], [10], [27], [28], and [38] (Table 1). Some of them are not considered to operate in reconfigurable systems. In addition, a few of them do not take into account an energy consumption constraint, which mostly requires more computation time to produce a correct solution. Multicore technology is also absent in some studies, which can have significant influence on job execution performance.

The solution gives the advantage of reducing energy consumption in each processor and then in the whole system. It offers more than 8% of gain compared with the approach in [27] and 15% of gain compared with the work in [1] when 20% of tasks finish execution in WCETs. The proposed approach also decreases by 80% the number of adjusted task periods compared with the work in [27] and by 93% compared with the approach in [1].

The contributions of this paper can be summarized as follows: We propose a scheduling scheme based on multi-speed cores, which consists of three cascading solutions depending on configuration scenarios:

1) We assign each task to different cores with different speeds while wishing to finish them as soon as possible with low-cost cores to save energy and cost,

2) If the first solution is not feasible, we adjust the periods of tasks with a new application of this solution, and

3) We migrate tasks to other lightly-loaded devices.

The rest of this paper is arranged as follows: We present in Section II a formalization of the hardware and software architecture of a system. In Section III, a solution for improving energy consumption in distributed multi-core reconfigurable architectures through integer programming is presented in detail. Finally, we display and discuss the experimental results.
results and compare the proposed contribution with related ones.

II. FORMALIZATION AND PRELIMINARIES

In this section, we formalize a system, hardware architecture, real-time tasks and energy models targeted in this work by using ILP formalization. The produced solution is a set of variables that indicate the task partitioning on different cores and processors. Preliminaries used in a linearization step are also presented. Used notations and their meanings are summarized in the nomenclature. We assume that a system’s tasks are independent.

A. HARDWARE ARCHITECTURE

We consider a distributed hardware platform with $m$ networked devices. Each device incorporates a multi-core processor composed of a set of multi-speed processing units called cores [12]. We denote by $Proc_p$ ($p = 1 \ldots m$) the processor embedded in $Device_p$. We denote by $Core_{kp}$ the $k$-th core of $Proc_p$ ($p = 1 \ldots m$, $k = 1 \ldots c$). Each processor is characterized by a set of frequencies $F = \{f_1, f_2, \ldots, f_k\}$ where $f_k$ is the frequency of $Core_{kp}$ in $Proc_p$. We attribute to each core a relative weight ($W_{kp}$) according to its frequency. The allocation is defined as follows:

$$W_{kp} = \frac{f_k}{\sum_{i=1}^{c} f_i}; \quad k = 1 \ldots c, \quad p = 1 \ldots m \quad (1)$$

B. REAL-TIME TASKS

We denote by $T = \{T_1, T_2, \ldots, T_n\}$ the set of $n$ periodic tasks to be executed upon a multi-core platform. $T_i$ is defined by the following parameters: $i)$ Absolute deadline $d_i$, i.e. $T_i$ must finish before $d_i$, $ii)$ Computation time $\zeta_{ip}$ at the normalized frequency on $Proc_p$, $iii)$ Period $\pi_i$, and $iv)$ Upper bound of each task’s period $\pi^M_i$.

A reconfigurable real-time system is composed of a set of $n$ tasks and $m$ devices. It is feasible if all the tasks meet their corresponding deadlines and no battery exceeds its capacity. Then, we assume a reconfiguration scenario to add a set of $l$ new tasks to the current feasible system. Some problems can arise in the extended system ($n + l$), such as the violation of some task deadlines. Thus, the batteries may not support the required energy consumption that exceeds their capacities; and, subsequently, become weak. This can also disrupt the feasibility in the related processor’s cores and consequently affects the feasibility of the entire system.

C. ENERGY MODEL

The total energy consumed is the sum of the dynamic and static components. Static energy is approximately constant while dynamic power consumption depends on the clock frequency. For this reason, we have targeted the modeling and calculation of dynamic energy in this paper.

The actual execution time of a task is prolonged when voltage is decreased to save the energy. The reason is that the frequency of a processor is approximately linearly proportional to supply voltage. Reducing voltage cuts down energy dissipation, but decreases the operating frequency accordingly. Hence, task execution time is inversely proportional to voltage. In order to provide the required system performance, the supply voltage should be scaled as low as possible to minimize energy consumption, while guaranteeing the timely completion of tasks.

Let $\tilde{V}_{kp}$ and $\tilde{F}_{kp}$ be respectively the normalized frequency and voltage of $Core_{kp}$ [28]. We suppose that task $T_i$ is executed at frequency $F_{kp}$ and voltage $V_{kp}$ in $Core_{kp}$. When $Core_{kp}$ is running at frequency $F_{kp}$ and voltage $V_{kp}$, the power consumption is given by $P_{kp} = RV_{kp}^2F_{kp}$, where $R$ is a constant that depends on a hardware circuit [33]. Power $P_{ikp}$ consumed by task $T_i$ in $Core_{kp}$ of $Proc_p$ is given by $P_{ikp} = V_{ikp}F_{ikp}R$, where $V_{ikp} = \tilde{V}_{kp}W_{kp}$. Consequently, the related consumed energy $E_{ikp}$ is expressed by $E_{ikp} = P_{ikp}C_{ikp} = \frac{\tilde{V}_{kp}F_{kp}}{W_{kp}}R$.

Each device is powered by an assigned battery and each one has its predefined capacity denoted by $B_p^M$. The energy consumption in $Device_p$ should not exceed this capacity. Thus, the energy consumption in $Device_p$ is defined as follows:

$$E_p = \sum_{i=1}^{n} \sum_{k=1}^{c} E_{ikp} \leq B_p^M. \quad (2)$$

D. THEORETICAL FOUNDATION

Many approaches have been proposed to solve rational optimization problems. The work [34] proposes a polynomial time algorithm to solve a class of optimization problems using rational objective functions as long as there is a performant algorithm to solve the problem with a linear objective function. Unfortunately, this does not cover our initially developed program (Fig. 1) since it is a mixed integer program with rational objective functions as long as there is a performant time algorithm to solve a class of optimization problems using rational objective functions.

A general scheme is proposed in [36] to approximate a linear rational function in a binary polyhedron. Since the problem is NP-complete, we develop an approximate solution based on linearization and convexification techniques. In this work, the extended model cannot be directly solved by any solver since it contains a rational constraint. The non-linearity of the model stems from the fact that the execution time of tasks is inversely proportional to the deadline adjustment. We propose a two-stage based approach (Fig. 1). First, we transform our model into a quadratic equivalent program, and then a linear one.

III. NEW SOLUTION FOR IMPROVING ENERGY CONSUMPTION

In this section, we detail the developed contribution that deals with three cascading solutions depending on reconfiguration scenarios. It consists in partitioning tasks on different cores,
A. MOTIVATION

The problem can be treated locally or globally depending on reconfiguration scenarios and feasibility tests (Fig. 2). The local treatment assigns each task to different cores while wishing to use as much as possible low-cost cores in terms of energy consumption. We start by executing the first slot of a task by the core with the lowest frequency while respecting deadlines. If ever the available core speeds are not able to guarantee the system feasibility, we adjust the periods of tasks as a flexible solution. Then we repeat the computing of assignment to different cores. This solution keeps a perfect system operation provided that the maximum values of periods are not overtaken. The solution must not exceed the capacities of batteries in devices.

When a local processor cannot support the reconfiguration with the period adjustment, we perform task migration and re-calculate the same assignment until reaching the feasibility of the system. This step consists in selecting the task set to be moved on the one hand and specifying the destination (the least loaded processors that are able to host tasks) on the other hand while keeping an eye on the system feasibility.

B. PARTITIONING OF TASKS ON MULTI-CORE ARCHITECTURES

Reaching an efficient schedule that keeps the system feasible after a considered reconfiguration scenario requires significant task planning and relevant assignment of segments to the available cores.

1) EXECUTION MODEL

The main objective of this step is to define an execution model that uses low-speed cores for reducing the energy consumption. The execution model deals with the assignment of tasks to different cores by acting on multi-speed cores.

In multi-core architectures, an application’s tasks can potentially run in parallel. Each task is divided into a set of segments, each of which can be executed on a processor’s core. Task segments are considered as independent of each other. Thus, they can run in parallel and out-of-order. This parallel execution exploits the advantage of the multi-speed technology with multiple processing cores to decrease the completion time of a task and the related energy consumption especially when a large number of tasks are able to finish their execution before their WCETs.

Before the assignment procedure, the execution time $C_{ni_p}$ of task $T_i$ is split into $h$ segments for execution. Let $T_i^h$ be the
The remaining tasks are distributed in the same way. Core load of CoreKP. The number of all segments is given by $C_{lp} = C_{1lp} + C_{2lp} + \ldots + C_{slp}$. Thus, $C_{slp}$ of task $T_i$ allocated to CoreKP can be determined as

$$C_{slp} = C_{lp} W_{kp}$$

(3)

**Example:** We consider a set of five real-time tasks to be executed upon a multi-core platform as depicted in Table 2. It consists of a set of four multi-core processors. Each processor is composed of four cores each of which has its own speed. $T_i$ is split into four segments $\{T_{i1}, T_{i2}, T_{i3}, T_{i4}\}$.

| Table 2. Basic system’s parameters. |
|-----------------|--|--|--|--|
| Tasks | $\zeta_{lp}$ | $d_i$ | $\pi_i$ | $\pi_i^M$ |
| $T_1$ | 11 | 15 | 15 | 17 |
| $T_2$ | 12 | 17 | 17 | 18 |
| $T_3$ | 10 | 14 | 14 | 14 |
| $T_4$ | 8 | 15 | 15 | 16 |
| $T_5$ | 8 | 10 | 10 | 10 |

The considered core speeds for the case study are 1.8Ghz, 2.4Ghz, 2.8Ghz, and 3.2Ghz. To explain a task splitting scheme, we introduce a task set example. To calculate the weights attributed to processor cores, we use the following formula:

$$W_c = \frac{f_c}{\sum_{k=1}^{m} f_k}$$

(4)

Then we have $W_1 = 0.2, W_2 = 0.23, W_3 = 0.27$, and $W_4 = 0.3$.

The split of WCET of task $T_i$ is given by

$$C_{ic} = C_i W_c$$

(5)

The segments of $C_i$ attributed to each core are described as follows:

- $C_{11} = 11 \times 0.2 = 2.2$,
- $C_{12} = 11 \times 0.23 = 2.53$,
- $C_{13} = 11 \times 0.27 = 2.97$, and
- $C_{14} = 11 \times 0.3 = 3.3$.

$C_{11} + C_{12} + C_{13} + C_{14} = C_i$.

The remaining tasks are distributed in the same way in Table 3.

The CPU load requested by segment $T_{ij}$ to be executed in CoreKP is defined as: $U_{slp} = \frac{C_{slp}}{d_j}$. A task set is deemed schedulable if the load in each processing core is equal to or less than 1 ($\forall k \in \{1 \ldots c\}, p \in \{1 \ldots m\}, U_{kp} \leq 1$). The load of CoreKP can be calculated as:

$$U_{kp} = \sum_{s=1}^{h} \sum_{i=1}^{n} \frac{C_{slp} X_{slp}}{d_i} \leq 1$$

(6)

| Table 3. Tasks set distribution. |
|-----------------|--|--|--|--|
| Tasks | $C_{i1}$ | $C_{i2}$ | $C_{i3}$ | $C_{i4}$ |
| $T_1$ | 11 * 0.2 | 11 * 0.23 | 11 * 0.27 | 11 * 0.3 |
| $T_2$ | 12 * 0.2 | 12 * 0.23 | 12 * 0.27 | 12 * 0.3 |
| $T_3$ | 10 * 0.2 | 10 * 0.23 | 10 * 0.27 | 10 * 0.3 |
| $T_4$ | 8 * 0.2 | 8 * 0.23 | 8 * 0.27 | 8 * 0.3 |
| $T_5$ | 8 * 0.2 | 8 * 0.23 | 8 * 0.27 | 8 * 0.3 |

where $X_{slp}$ is a binary variable that characterizes the allocation constraint between segments and cores.

$$X_{slp} = \begin{cases} 1 & \text{if segment } T_{ij} \text{ is assigned to Core}_{kp} \text{ of Proc}_p \\ 0 & \text{otherwise} \end{cases}$$

(7)

By considering task segmentation, the energy consumed by $T_{ij}$ on CoreKP is given as $E_{slp} = P_{slp} C_{slp}$. The energy consumption in Device$_p$ becomes:

$$E_p = \sum_{s=1}^{h} \sum_{i=1}^{n} \sum_{k=1}^{m} E_{slp} X_{slp}; \quad p = 1 \ldots m.$$  

(8)

To have effective planning, each task’s segment must be assigned to a single core during its execution. Thus, we have the following constraint:

$$\sum_{p=1}^{h} \sum_{k=1}^{m} X_{slp} = 1; \quad s = 1 \ldots h, \quad i = 1 \ldots n.$$  

(9)

**Assignment Constraints:** Let $t_{slp}$ be the real starting date of $T_{ij}$ in CoreKP. The starting time of each segment must have a positive value, i.e.,

$$t_{slp} \geq 0; \quad s = 1 \ldots h, \quad i = 1 \ldots n, \quad k = 1 \ldots c, \quad p = 1 \ldots m.$$  

(10)

To avoid multiple-segment execution in a time, $T_{ij}$ cannot start before $T_{ij}^j$ completes, which means that the difference between the starting times of $T_{ij}$ and $T_{ij}^j$ must necessarily be greater than the execution time of $T_{ij}^j$ or in reverse if $T_{ij}^j$ starts before $T_{ij}^j$. We must ensure that $t_{slp} - t_{slp} = C_{slp} \geq 0$ or $t_{slp} - t_{slp} - C_{slp} \geq 0$ for each segment pair $T_{ij}$ and $T_{ij}^j$, and a binary variable $\alpha_{ij}$ is used to guarantee both inequalities at the same time, i.e.,

$$\alpha_{ij} \in \{0, 1\}; \quad i < j; \quad i, j = 1 \ldots n.$$  

(11)

$\alpha_{ij} = 1$ ensures that $T_{ij}$ is executed before $T_{ij}^j$. Therefore, if $T_{ij}^j$ gets started before $T_{ij}^j$ in CoreKP, $T_{ij}^j$ cannot begin execution if $T_{ij}^j$ is running. The previously added constraints avoid multiple active segments on any core and at any time. The related constraints are:

$$t_{slp} - t_{slp} \geq \zeta_{ij} W_{kp} X_{slp} - M \alpha_{ij}$$  

(12)

$$t_{slp} - t_{slp} \geq \zeta_{ij} W_{kp} X_{slp} - M(1 - \alpha_{ij})$$  

(13)
To assure the fulfillment of (12) and (13), the starting time of each task should be less than or equal to a big constant \( M \).

\[
t_{sikp} \leq M \quad \text{(14)}
\]

In the case where task \( T_i^k \) is executed before \( T_i^k \), (14) keeps (12) valid. In fact, in this case, we have \( \alpha_{ij} = 1 \) and subsequently \( t_{sikp} - t_{sikk} \leq 0 \). Therefore \( \xi_{kp} W_{kp} X_{sikp} - M \) must also be negative and less than \( t_{sikp} - t_{sikk} \). Hence, \( t_{sikp} - t \) and \( t_{sikk} - t \) must necessarily be less than \( M \).

Among the assignment steps, we aim to execute as much as possible the first slot of a task by the core with the lowest frequency to consume less energy. According to this, we look for a feasible solution to ensure that the cores with lowest frequencies are more lightly-loaded than those with the highest frequencies. To reach this objective, the load \( U_{kp} \) of Core\(_{kp} \) should increase as their frequency \( f_{kp} \) decreases since \( f_{kp} < f_{k+i+1} \) \( \forall k \in \{1, 2, \ldots, c\} \). Thus, achieving this goal is tantamount to performing:

\[
\text{Maximize } \sum_{k=1}^{c} \frac{U_{kp}}{f_{kp}} = \sum_{s=1}^{h} \sum_{i=1}^{n} \sum_{k=1}^{c} \sum_{p=1}^{m} \frac{C_{sikp} X_{sikp}}{df_{kp}}.
\]

The previous maximization function can be transformed to a minimization function since maximizing \( f(x) \) equates minimizing \(-f(x)\). At this level the objective becomes multiple, i.e., minimizing the consumed energy and assigning the segments to the processor that has the lowest speed. To ensure a compromise between the weight of an assignment strategy and the energy in the objective function, we add two weight factors \( \sigma \) and \( \lambda \) such that the objective becomes

\[
\text{Minimize } \sum_{s=1}^{h} \sum_{i=1}^{n} \sum_{k=1}^{c} \sum_{p=1}^{m} \frac{\sigma E_{sikp} X_{sikp} - \lambda C_{sikp} X_{sikp}}{df_{kp}} \quad \text{(15)}
\]

Since in a mathematical model, multiplying the objective function by a constant has no effect on the solution, we multiply (15) by \( \frac{1}{\sigma} \) to obtain:

\[
\text{Minimize } \sum_{s=1}^{h} \sum_{i=1}^{n} \sum_{k=1}^{c} \sum_{p=1}^{m} \frac{E_{sikp} X_{sikp} - \lambda C_{sikp} X_{sikp}}{\sigma df_{kp}} \quad \text{(16)}
\]

Let \( \gamma = \frac{\lambda}{\sigma} \). This factor is used in order to favor an objective function against the other as required. To give more importance to energy consumption in the objective function, \( \gamma \) must have a small value. Reversely, \( \gamma \) must be big. The correspondent objective function becomes:

\[
\text{Minimize } \sum_{s=1}^{h} \sum_{i=1}^{n} \sum_{k=1}^{c} \sum_{p=1}^{m} \frac{E_{sikp} X_{sikp} - \gamma C_{sikp} X_{sikp}}{df_{kp}} \quad \text{(17)}
\]

**Timing Constraints:** The timing constraints present a decisive factor when the reliability of a system is constrained by time. Each segment should begin execution next to its release date, i.e.,

\[
t_{sikp} \geq \tau_{sikp} \quad \text{(18)}
\]

and each task should end execution without violating its related deadline, i.e.,

\[
\frac{t_{sikp} + \lambda_{kp} W_{kp} X_{sikp}}{f_{kp}} \leq d_i \quad \text{(19)}
\]

By modeling the previous constraints, we establish the main mathematical program called PI. It is linear since all constraints are so.

\[
\begin{align*}
\text{Minimize} & \sum_{i=1}^{h} \sum_{i=1}^{n} \sum_{k=1}^{c} \sum_{p=1}^{m} E_{sikp} X_{sikp} - \gamma \frac{C_{sikp} X_{sikp}}{df_{kp}} \\
& t_{sikp} - t_{sikk} - C_{nkp} W_{kp} X_{sikp} + M \alpha_{ij} \geq 0 \\
& t_{sikp} - t_{sikk} - C_{nkp} W_{kp} X_{sikp} + M (1 - \alpha_{ij}) \geq 0 \\
& t_{sikp} + \lambda_{kp} W_{kp} X_{sikp} - d_i \leq 0 \\
& f_{kp} \\
& \frac{h}{i} \sum_{i=1}^{h} \sum_{i=1}^{n} \sum_{k=1}^{c} \sum_{p=1}^{m} E_{sikp} X_{sikp} \leq \beta_{ik} \\
& \sum_{i=1}^{h} \sum_{i=1}^{n} \sum_{k=1}^{c} \sum_{p=1}^{m} X_{sikp} = 1 \\
& t_{sikp} - r_{sikp} \geq 0 \\
& t_{sikp} \geq 0 \\
& t_{sikk} - M \leq 0 \\
& W_{kp} \geq 0 \\
& \alpha_{ij} \in \{0, 1\}
\end{align*}
\]

2) EXTENDED EXECUTION MODEL WITH DEADLINE ADJUSTMENT

For more flexibility in a system, we introduce a mechanism of period adjustment. It keeps the system functional even if the available cores speeds are not able to guarantee the system feasibility. We denote by \( \beta_i \) the fractional increase of the period of task \( T_i \). To guarantee the stretching of its deadline, we add the constraint:

\[
\beta_i \geq 1; \quad i = 1 \ldots n. \quad \text{(20)}
\]

By including the period adjustment constraint, we ensure that all tasks are completed before their deadlines, i.e.,

\[
\frac{t_{sikp} + \xi_{kp} W_{kp} X_{sikp}}{f_{kp}} \leq \beta_i d_i \quad \text{(21)}
\]

To monitor the period modification at run-time, we append parameter \( \pi_i^M \) that represents the upper bounds of period \( \pi_i \). We have:

\[
\beta_i d_i \leq \pi_i^M; \quad i = 1 \ldots n. \quad \text{(22)}
\]
We also append constant $\theta$ to characterize a trade-off weight between deepening coefficient $\beta_i$ and energy. The objective function is given as:

$$OF = \theta \sum_{i=1}^{n} \beta_i + \sum_{s=1}^{h} \sum_{i=1}^{n} \sum_{k=1}^{c} \sum_{p=1}^{m} E_{skp}X_{skp} - \gamma \frac{C_{skp}X_{skp}}{d_{jkp}}$$

(23)

By bringing together all these constraints, we obtain an extended model called PE.

**Minimize OF**

$$t_{skp} - t_{skp} - C_{nkp}W_{kp}X_{skp} + M\alpha_{ij} \geq 0$$

$$t_{skp} - t_{skp} - C_{nkp}W_{kp}X_{skp} + M(1 - \alpha_{ij}) \geq 0$$

$$t_{skp} + \xi_{kp}W_{kp}X_{skp} - \beta_i d_{i} \leq 0$$

$$\sum_{i=1}^{h} \sum_{i=1}^{n} \frac{C_{skp}X_{skp}}{\beta_i d_{i}} \leq 1$$

$$\beta_i \geq 1$$

$$\beta_i = d_{i} \leq \pi_i^M$$

**PE**

$$\sum_{k=1}^{c} X_{skp} = 1$$

$$t_{skp} - r_{skp} \geq 0$$

$$t_{skp} \geq 0$$

$$t_{skp} - M \leq 0$$

$$W_{kp} \geq 0$$

$$\alpha_{ij} \in \{0, 1\}$$

We obtain the equivalent quadratic program.

**Minimize OF**

$$t_{skp} - t_{skp} - C_{nkp}W_{kp}X_{skp} + M\alpha_{ij} \geq 0$$

$$t_{skp} - t_{skp} - C_{nkp}W_{kp}X_{skp} + M(1 - \alpha_{ij}) \geq 0$$

$$t_{skp} + \xi_{kp}W_{kp}X_{skp} - \beta_i d_{i} \leq 0$$

$$\sum_{i=1}^{h} \sum_{i=1}^{n} \sum_{k=1}^{c} E_{skp}X_{skp} \leq B_p^M$$

$$\sum_{k=1}^{c} X_{skp} = 1$$

$$t_{skp} - r_{skp} \geq 0$$

$$t_{skp} \geq 0$$

$$t_{skp} - M \leq 0$$

$$W_{kp} \geq 0$$

$$\alpha_{ij} \in \{0, 1\}$$

However PQ is also difficult to solve since its constraint (24) is non-linear as it contains the product of two variables $\beta_i$ and $z_i$. So is (26) because of $X_{skp}z_{i}$.

**Stage 2:** In this stage, we linearize (24) and (26) by using a linearization technique. Typically, the linearization [26] is based on the principle that each product is replaced by a new variable and then a set of new linear constraints is added to guarantee the equality between a new variable and the product. In PQ, we replace $\beta_i z_i$ by $y_i$. Let $U^Z$ be an upper bound on $z_i$. The following inequalities are added to guarantee that $\beta_i z_i = y_i$:

$$\begin{cases} 
 z_i - y_i \leq U^Z(1 - \beta_i) \\
 y_i \leq z_i \\
 y_i \leq U^Z \beta_i \\
 y_i \geq 0
\end{cases}$$

(1)

In a similar way, we linearize (24). We replace $X_{skp}z_{i}$ by $x_i$ and we add the corresponding constraints:

$$\begin{cases} 
 z_i - x_i \leq U^Z(1 - X_{skp}) \\
 x_i \leq z_i \\
 x_i \leq U^Z X_{skp} \\
 x_i \geq 0
\end{cases}$$

(1)
system’s reliability. This step is not interested in assigning a task to another. Migrating tasks consists in finding a new configuration for the whole tasks in the system.

To meet the system’s feasibility without having to touch the configuration, we resume the allocation step while giving tasks the opportunity of migrating among processors in order to find the right segment partitioning on different cores. In the case where the assignment of tasks to different cores is impossible (11), we try to configure a particular device. They try to apply the PI and PE models designed to be applied if a reconfiguration has occurred in a particular device. The task set in Table 4 is ready to be executed in a parallel way on the four cores of each processor. The schedulability of the system can be ensured by the following mathematical program.

\[
\begin{align*}
\text{Minimize} & \quad OF \\
& t_{skp} - c_{skp} - C_{n_jp} W_{kp} X_{skp} + M \alpha_{ij} \geq 0 \\
& t_{skp} - t_{skp} - C_{n_jp} W_{kp} X_{skp} + M (1 - \alpha_{ij}) \geq 0 \\
& t_{skp} + \xi_{ip} W_{kp} X_{skp} - \beta_i d_i \leq 0 \\
& \frac{h}{s=1} \sum_{i=1}^n C_{skp} X_{skp} \leq 1 \\
& z_i \geq 0 \\
& z_i - y_i \leq U^Z (1 - \beta_i) \\
& y_i \leq z_i \\
& y_i \leq U^Z \beta_i \\
& y_i \geq 0 \\
& z_i - x_i \leq U^Z (1 - X_{skp}) \\

\end{align*}
\]

Program PX can be solved by mixed integer program solvers such as CPLEX.\(^1\)

**C. MIGRATION OF RECONFIGURABLE TASKS**

PI and PE models are designed to be applied if a reconfiguration has occurred in a particular device. They try to find the right segment partitioning on different cores. In the case where the assignment of tasks to different cores is inapplicable, and the modification of the task periods on the corresponding processor cannot guarantee the system’s feasibility, we resume the allocation step while giving tasks the opportunity of migrating among processors in order to meet the system’s feasibility without having to touch the whole tasks in the system.

We need specify a way to migrate tasks from one processor to another. Migrating tasks consists in finding a new combination between tasks and processors to ensure the system’s reliability. This step is not interested in assigning segments to cores. Therefore, we replace variable \(X_{skp}\) with \(Y_{ip}\) to characterize the allocation between tasks and processors.

\[
Y_{ip} = \begin{cases} 
1 & \text{if task } T_i \text{ is assigned to } Proc_p \\
0 & \text{otherwise} 
\end{cases}
\]

To ensure the correct migration of a task, we should guarantee that a task is not allocated to two different processors, i.e.,

\[
\sum_{p=1}^m Y_{ip} = 1
\]

We update the starting and release dates of a task in a processor to be denoted by \(t_{ip}\) and \(r_{ip}\), respectively. Minimizing the number of tasks for migration is adopted in this step in order to reduce the related cost. After updating the previous constraints, we obtain the program called PM:

\[
\begin{align*}
\text{Minimize} & \quad \sum_{k=1}^c \sum_{p=1}^m E_{ip} Y_{ip} \\
& t_{ip} - t_{ip} - C_{n_jp} Y_{ip} + M \alpha_{ij} \geq 0 \\
& t_{ip} - t_{ip} - C_{n_jp} Y_{ip} + M (1 - \alpha_{ij}) \geq 0 \\
& t_{ip} + \xi_{ip} Y_{ip} - d_i \leq 0 \\
& \sum_{p=1}^m C_{n_jp} Y_{ip} \leq 1 \\
& \sum_{p=1}^m Y_{ip} = 1 \\
& t_{ip} \geq 0 \\
& t_{ip} \leq 0 \\
& t_{ip} - M \leq 0 \\
\end{align*}
\]

\(\alpha_{ij} \in \{0, 1\}\)

**IV. EXPERIMENTATION**

In this paper, we use the AMPL language and ILOG CPLEX 11.0 solver to model and solve mathematical programs. They provide an analytical decision tool that allows the rapid deployment of optimization models by using mathematical and constraint programming [37].

**A. CASE STUDY**

We consider a real-time reconfigurable system. A local processor supporting the multi-core technology is considered for each device. Each processor has a set of multi-speed cores allowing the execution of implicit-periodic OS tasks. Each task is described by a period, worst acceptable period, deadline, release date, and completion time.

Scenario 1 (Related Processors Can Support a New Reconfiguration): We consider an initial assignment of tasks to different processors (respectively cores) presented in Table 5. The task set in Table 4 is ready to be executed in a parallel way on the four cores of each processor. The schedulability

\[^1\text{IBM ILOG CPLEX optimizer. http://www-03.ibm.com/software/products/en/ibmilogcplex-leoptstud/}\]
TABLE 4. Basic system’s parameters.

| Tasks | $\zeta_{ip}$ | $d_i$ | $\pi_i$ | $\pi_i^M$ |
|-------|--------------|-------|---------|---------|
| $T_1$ | 7            | 11    | 11      | 13      |
| $T_2$ | 9            | 12    | 12      | 15      |
| $T_3$ | 11           | 15    | 15      | 17      |
| $T_4$ | 12           | 17    | 17      | 18      |
| $T_5$ | 10           | 14    | 14      | 16      |
| $T_6$ | 9            | 14    | 14      | 16      |
| $T_7$ | 14           | 20    | 20      | 24      |
| $T_8$ | 13           | 18    | 18      | 22      |
| $T_9$ | 10           | 16    | 16      | 18      |
| $T_{10}$ | 7           | 10    | 10      | 14     |

Table 5. Initial tasks allocation.

| Processor | Assigned tasks |
|-----------|---------------|
| CPU1      | $\{T_1, T_2\}$ |
| CPU2      | $\{T_3, T_4, T_5\}$ |
| CPU3      | $\{T_6, T_7, T_8\}$ |
| CPU4      | $\{T_9, T_{10}\}$ |

Constraint is satisfied at this stage. The utilization is less than or equal to the number of cores in each processor ($U \leq 4$).

We assume a run-time reconfiguration scenario in Device$_p$ that adds two new tasks $T_{11}$ and $T_{12}$ to the queue for execution on CPU$_2$ (Table 6).

TABLE 6. First reconfiguration scenario on CPU$_2$.

| Tasks | $\zeta_{ip}$ | $d_i$ | $\pi_i$ | $\pi_i^M$ |
|-------|--------------|-------|---------|---------|
| $T_{11}$ | 8            | 15    | 15      | 18      |
| $T_{12}$ | 8            | 10    | 10      | 15      |

Table 7 shows the allocation of the different segments of tasks to the cores of CPU$_2$ after applying the execution model PI. Fig. 3 presents the related scheduling plan.

TABLE 7. Applying PI model for CPU$_2$.

| Processor | Core | Segments |
|-----------|------|----------|
| CPU$_2$   | Core1 | $(T_1^{2}, T_1^{3}, T_1^{4}, T_1^{5}, T_2^{3}, T_2^{4}, T_3^{1}, T_3^{2})$ |
|           | Core2 | $(T_2^{3}, T_3^{1}, T_3^{2}, T_4^{1}, T_4^{2})$ |
|           | Core3 | $(T_1^{2}, T_5^{4}, T_2^{2}, T_3^{4}, T_3^{3})$ |
|           | Core4 | $(T_1^{1}, T_1^{2})$ |

We note that $U_{21} = 0.998 \leq 1$, $U_{22} = 0.964 \leq 1$, $U_{23} = 0.997 \leq 1$, and $U_{24} = 0.32 \leq 1$. Hence the system is feasible. The consumed energy in CPU$_2$ is computed 848.73 Joules. We assume now a new reconfiguration scenario that occurs in CPU$_2$ and requests adding two new tasks $T_{13}$ and $T_{14}$ to be executed as depicted in Table 8. After PI, we note that $U_{21} = 1.285 > 1$, $U_{22} = 1.118 > 1$, $U_{23} = 1.261 > 1$, and $U_{24} = 0.773 < 1$. Then the system is infeasible since CPU$_2$ cannot support the new reconfiguration where some tasks may violate their deadlines (Fig. 4). The consumed energy in CPU$_2$ is equal to 1085.76 Joules.

TABLE 8. Second reconfiguration scenario on CPU$_2$.

| Tasks | $\zeta_{ip}$ | $d_i$ | $\pi_i$ | $\pi_i^M$ |
|-------|--------------|-------|---------|---------|
| $T_{13}$ | 9            | 12    | 12      | 16      |
| $T_{14}$ | 6            | 10    | 10      | 13      |

Scenario 2 (Necessity to Adjust Periods): The CPU$_2$ load exceeds its capacity. At this level, a local solution consists in adjusting the periods of few tasks to meet the system’s feasibility without migrating to other processors. By applying PE, we get the following result with the new adjusted periods (Table 9 and Fig. 5) and the allocation table (Table 10) which fixes for each core the related set of tasks to execute.

TABLE 9. Second reconfiguration scenario on CPU$_2$.

| Tasks | $\zeta_{ip}$ | $d_i$ | New Period | $\pi_i^M$ |
|-------|--------------|-------|------------|---------|
| $T_3$ | 15           | 15    | 17         |        |
| $T_4$ | 17           | 17    | 18         |        |
| $T_5$ | 14           | 14    | 16         |        |
| $T_{11}$ | 15           | 15    | 18         |        |
| $T_{12}$ | 10           | 10    | 15         |        |
| $T_{13}$ | 12           | 14    | 16         |        |
| $T_{14}$ | 10           | 13    | 13         |        |
Scenario 3 (Necessity for Task Migration): Table 11 shows the parameters of task $T_{15}$ to be added to CPU$_2$ for execution as a new run-time reconfiguration request. After applying PE, we observe that $U_{21} = 1.496 > 1$, $U_{22} = 1.187 > 1$, $U_{23} = 1.336 > 1$, and $U_{24} = 1.023 > 1$. Thus, the timing constraints of tasks may be violated under this condition and the processor is overloaded. Changing the periods can be a flexible and efficient solution. Yet the adjustment is always limited by a maximum acceptable value. If the modification of the periods of some tasks could not give the desired result, task migration must be conducted to overcome the problem by applying PM (Table 12).

### TABLE 11. Third reconfiguration scenario.

| Tasks  | $\zeta_{T_1}$ | $d_i$ | $\pi_i$ | $\pi^{\#}_i$ |
|--------|---------------|------|--------|-------------|
| $T_{15}$ | 11            | 14   | 14     | 16          |

### B. PERFORMANCE EVALUATION

To evaluate the performance of the proposed approaches in this paper, we take a random test games by generating 80 instances of implicit-deadline real-time tasks that represent reconfiguration scenarios. We consider a multi-processor system with multi-speed core capability. The WCET of each task is randomly generated by using a uniform distribution [38]. Other task parameters such as release time and deadline are also randomly generated. The experiments are performed by using Intel Core® i3-2670QM processors with 4 GB of RAM.

In order to compare the contribution of the proposed solution with the recent approaches in [1], [27] according to the cost of periods adjustment, we try to implement the same case study. Fig. 6 confirms the clear gain of this contribution (bars in red) compared with those reported in [27] (bars in green) and [1] (bars in blue). The cost of period amendment is considered as the sum of the gaps between the basic periods values and those after reconfiguration. We see that the cost of period modification produced by our solution is estimated respectively on the three cores to be 74, 35 and 46. For the same metric, the cost is estimated to 955,840 and 500 when applying the solution in [1] and to 196, 349 and 250 by the solution in [27]. Indeed, the proposed solution is based on a multi-objective mathematical model that allows one to determine the optimal value by minimizing the gap between old and new periods as much as possible to make the system functional after reconfiguration. The proposed approach also improves by 80% the overall number of adjusted task periods compared with the work in [27] and by 93% compared with the approach in [1].

Fig. 7 shows a comparison of the proposed approach with those reported in [1], [27], and [28] based on the energy cost.
The energy consumption ratio confirms the efficiency of using multi-core technology. This technology exploits parallelism in task execution, resulting in more significant power consumption performance with 70% of reduction over a non-multi-core system (Fig. 7). By applying the suggested strategy, the system can save 9% of power. It is obvious that in the case that the multi-core technology is not used, the energy consumption is practically very high because processors operate at their maximum frequency and require the maximum power. Figs. 8–12 describe the power consumption of the proposed solution and those in [27] and [1] for a 2-core, 4-core, 6-core and 8-core processor architecture.

The proposed solution offers better than 4% of gain than the approach in [27] and 6.7% of gain than that in [1] when 100% of tasks finish execution in WCETs, and 8% of gain over the approach in [27] and 15% of gain over that in [1] when 20% of tasks finish execution in WCETs.

The particularity of this work is based on multi-speed cores and lies in using lower core speed first when executing tasks and reducing execution speeds in some processor cores to consume less energy. Migrating tasks from an over-loaded processor to a less loaded one allows to find a schedule that guarantees the feasibility of the system and balance the workload among all processors.

Table 13 and Fig. 13 show that the proposed approach produces better results than those in [10] and [28] in terms of computational time for instances of 10 to 25 tasks with a multi-processor platform composed of 10 and 15 processors. It offers more than 70% of gain over the approach in [10] and a gain of 60% over the approach in [28]. These approaches fail to address the context of reconfigurable systems and multi-core architectures which can act directly on the quality of generated result and how long.

The proposed approach is also compared with [7], [38], and [28] in terms of computational time and makespan. The numerical results are described in Table 14. The first
### TABLE 13. Comparison based on computational time.

| Tasks | Processors | Our solution | Hanza et al. [28] | GA [10] | SA [10] | Lingo [10] |
|-------|------------|--------------|-------------------|---------|---------|------------|
| 10    | 10         | 27           | 80                | 350     | 165     | 100        |
| 20    | 10         | 94           | 205               | 400     | 200     | 164        |
| 20    | 15         | 54           | 135               | 416     | 121     | 850        |
| 25    | 15         | 66           | 460               | 900     | 750     | 940        |

### TABLE 14. Comparison between the proposed solution and those in [28], [38], and [7].

| Tasks | Processors | Our solution | Hanza et al. [28] | MA [7] | TS [38] | SA [38] | Makespan |
|-------|------------|--------------|-------------------|--------|---------|---------|----------|
| 10    | 4          | 47           | 113               | 240    | 1170    | 188     | 37.23    |
| 20    | 8          | 72           | 156               | 1730   | 1739    | 1047    | 16.11    |
| 20    | 16         | 21           | 87                | 2690   | 1559    | 1078    | 9.51     |

### FIGURE 12. Energy evolution of the proposed solution and those in [1], and [27] when 20% of tasks finish execution in WCETs.

### FIGURE 13. Comparison between the proposed solution and those in [28] and [10] based on computational time.

### C. DISCUSSION

The proposed solution offers better than 4% of gain over that in [27], 6.7% of gain over that in [1] when 100% of tasks finish execution in WCETs, 8% of gain over that in [27], and 15% of gain over that in [1] when 20% of tasks finish execution in WCETs. It also improves by 80% the overall advantage of the proposed solution over existing methods. It presents more than 75% of gain over that in [38] and a gain of 80% than that in [7] according to computational time. Observing makespan, the proposed approach can improve results by 61% over that in [38] and 49% against that in [7]. It gives also a gain more than 35% over that in [28]. The integer programming model represents a useful and optimal method but it can be computationally intensive to be used for large instances. However, it is gainful to produce a lower amount of energy consumed by a given system. Our solution combines a kind of constraints such as energy, makespan and computational time to generate a well-balanced solution that respects all constraints and may be more appropriate for reconfigurable systems.
number of adjusted task periods over that in [27] and by 93% over that in [1].

The proposed solution also offers great gain that can reach 60% over those in [10], [28] and 75% over those in [7], [38] according to the computational time. Makespan comparison further validates its superiority to those in [7], [38], and [28] with an improvement of 35% to 49%.

Using lower core speed first when executing tasks, reducing execution speeds in some processor cores to consume less energy and migrating tasks from an over-loaded processor to a less loaded one after reconfiguration scenarios is the most significant contribution of this research. It allows to find a schedule that guarantees the feasibility of the system, reduce energy cost and yield more opportunity to the load balancing among processing cores. Note that some issues like task dependency and resource sharing [41] should be considered in the model development.

V. CONCLUSION
Energy consumption is a decisive metric in the phase of evaluating the performance of a real-time system. Better management of the power sources of a system to reduce this metric requires effective solutions. These solutions depend on task models and considered architectures. In addition, a real-time task scheduling problem on a multi-core architecture is known to be NP-hard and remains difficult to solve.

We propose a scheduling strategy that acts on multi-speed cores and consists of three solutions (Partitioning tasks on different cores, period adjustment and task migration) depending on configuration scenarios. An ILP is formulated to define an execution model that deals with the assignment of tasks to different cores with the optimal energy consumption.

From the experimental results, we conclude that the proposed solution improves the energy consumption in the whole system while keeping its feasibility. In the future work, we plan to focus on an extended execution model that considers mixed categories of tasks (aperiodic and sporadic) related by dependency constraints. Communication and migration costs should also be considered since they are often present in real-time applications. We can transform dependent tasks to independent by using a virtual processor. We intend to implement real-time middleware that embeds the proposed contribution and some intelligent optimization methods [40], [42]. In the future, we plan to use the developed method to the social networks [43], [44] and discreet event systems [45]–[47].

REFERENCES
[1] X. Wang, I. Khemaissa, M. Khalgui, Z. Li, O. Mosbah, and M. Zhou, “Dynamic low-power reconfiguration of real-time systems with periodic and probabilistic tasks,” IEEE Trans. Autom. Sci. Eng., vol. 12, no. 1, pp. 258–271, Mar. 2015.
[2] W. Lakhdir, R. Mizd, M. Khalgui, Z. Li, G. Frey, and A. Al-Ahmar, “Multiobjective optimization approach for a portable development of reconfigurable real-time systems: From specification to implementation,” IEEE Trans. Syst., Man, Cybern. Syst., vol. 49, no. 3, pp. 623–637, Mar. 2019.
[3] F. Jarray, H. Chniter, and M. Khalgui, “New adaptive middleware for real-time embedded operating systems,” in Proc. IEEE/ACIS 14th Int. Conf. Comput. Inf. Sci. (ICIS), Las Vegas, NV, USA, Jun. 2015, pp. 610–618.
[4] H. Chniter, M. Khalgui, and F. Jarray, “A based-optimization scheduling approach for multi-processor platform with DVFS facilities,” in Proc. ESM, Leicester, U.K., Oct. 2015, pp. 164–171.
[5] Y. Chen, Z. Li, A. Al-Ahmar, N. Wu, and T. Qu, “Deadlock recovery for flexible manufacturing systems modeled with Petri nets,” Inf. Sci., vol. 381, pp. 290–303, Mar. 2017.
[6] H. F. Sheikh, I. Ahmad, and D. Fan, “An evolutionary technique for Performance-Energy-Temperature optimized scheduling of parallel tasks on multi-core processors,” IEEE Trans. Parallel Distrib. Syst., vol. 27, no. 3, pp. 668–681, Mar. 2016.
[7] A. Serafettin, “A memetic algorithm for parallel machine scheduling,” in Proc. GEM, Athens, GA, USA, Jul. 2012, pp. 108–113.
[8] S. Wang, Z. Qian, J. Yuan, and I. You, “A DVFS based energy-efficient tasks scheduling in a data center,” IEEE Access, vol. 5, pp. 13090–13102, 2017.
[9] M. Bambagioni, M. Marinoni, H. Aydin, and G. Buttazzo, “Energy-aware scheduling for real-time systems: A survey,” ACM Trans. Embedded Comput. Syst., vol. 15, no. 1, pp. 1–7, 2016.
[10] V. M. Dalfard and G. Mohammad, “Two meta-heuristic algorithms for solving multi-objective flexible job-shop scheduling with parallel machine and maintenance constraints,” Comput. Math. Appl., vol. 64, no. 6, pp. 2111–2117, Sep. 2012.
[11] T.-H. Chen and R.-G. Chang, “A thermal-aware scheduling for multicore architectures,” J. Syst. Archit., vol. 62, pp. 54–62, Jan. 2016.
[12] Y. G. Kim, M. Kim, and S. W. Chung, “Enhancing energy efficiency of multimedia applications in heterogeneous mobile multi-core processors,” IEEE Trans. Comput., vol. 66, no. 11, pp. 1878–1889, Mar. 2017.
[13] K. Chronaki, A. Rico, M. Casas, M. Moreto, R. M. Badia, E. Ayguade, J. Labarta, and M. Valero, “Task scheduling techniques for asymmetric multi-core systems,” IEEE Trans. Parallel Distrib. Syst., vol. 28, no. 7, pp. 2074–2087, May 2017.
[14] M. Han, N. Guan, J. Sun, Q. He, Q. Deng, and W. Liu, “Response time bounds for typed DAG parallel tasks on heterogeneous multi-cores,” IEEE Trans. Parallel Distrib. Syst., vol. 30, no. 11, pp. 2567–2581, Nov. 2019.
[15] S. I. Kim and J.-K. Kim, “A method to construct task scheduling algorithms for heterogeneous multi-core systems,” IEEE Access, vol. 7, pp. 142640–142651, 2019.
[16] J. Chen, C. Du, P. Han, and Y. Zhang, “Sensitivity analysis of strictly periodic tasks in multi-core real-time systems,” IEEE Access, vol. 7, pp. 135005–135022, 2019.
[17] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. Buttazzo, “Schedulability analysis of conditional parallel task graphs in multicore systems,” IEEE Trans. Comput., vol. 66, no. 2, pp. 339–353, May 2017.
[18] A. Lele, O. Moreira, P. J. L. Cuijpers, and K. van Berkel, “Response modeling runtime schedulers for timing analysis of self-timed dataflow graphs,” J. Syst. Archit., vol. 65, pp. 15–29, Apr. 2016.
[19] L. Zhang, “Query schedule algorithm based on deadline and value over data stream system,” in Proc. Int. Conf. Multimedia Technol., Hangzhou, China, Jul. 2011, pp. 134–141.
K.-H. Chen, G. Bruggen, and J.-J. Chen, “Reliability optimization on multi-core systems with multi-tasking and redundant multi-threading,” *IEEE Trans. Comput.*, vol. 67, no. 4, pp. 484–497, Feb. 2018.

A. Naiathani, S. Eyerman, and L. Eeckhout, “Optimizing soft error reliability through scheduling on heterogeneous multicores processes,” *IEEE Trans. Comput.*, vol. 67, no. 6, pp. 830–846, 2018.

M. Schoeberl, L. Pezzarossa, and J. Spars, “A multi-core processor for time-critical applications,” *IEEE Design Test*, vol. 35, no. 2, pp. 38–47, Mar. 2018.

M. Pricopi and T. Mitra, “Task scheduling on adaptive multi-core,” *IEEE Trans. Comput.*, vol. 63, no. 10, pp. 2590–2603, Oct. 2014.

A. Yoosefi and H. R. Naji, “A clustering algorithm for communication-aware scheduling of task graphs on multi-core reconfigurable systems,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2718–2732, Oct. 2017.

R. Pathan, P. Voudouris, and P. Stenstrom, “Scheduling parallel real-time recurrent tasks on multicore platforms,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 4, pp. 915–928, Apr. 2018.

A. Billionnet and K. Djebali, “Solution of a fractional combinatorial optimization problem by mixed integer programming,” *RAIRO Oper. Res.*, vol. 40, no. 2, pp. 97–111, Apr. 2006.

A. Gammoudi, A. Benzina, M. Khalgui, and D. Chillet, “Real-time scheduling of reconfigurable battery-powered multi-core platforms,” in *Proc. IEEE 28th Int. Conf. Tools with Artif. Intell. (ICTAI)*, San Jose, CA, USA, Nov. 2016, pp. 121–129.

H. Chniter, Y. Li, M. Khalgui, A. Koubaa, Z. Li, and F. Jarray, “Multi-agent adaptive architecture for flexible distributed real-time systems,” *IEEE Access*, vol. 6, pp. 23152–23171, 2018.

M. Kang, C. Wen, and C. Wu, “A model predictive scheduling algorithm in real-time control systems,” *IEEE/CACJ J. Automatika Sinica*, vol. 5, no. 2, pp. 471–478, Mar. 2018.

Y. Xia, M. Zhou, X. Luo, S. Pang, and Q. Zhu, “A stochastic approach to analysis of energy-aware DVS-enabled cloud datacenters,” *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 45, no. 1, pp. 73–83, Jan. 2015.

H. Yuan, J. Bi, and M. Zhou, “Temporal task scheduling of multiple delay-constrained applications in green hybrid cloud,” *IEEE Trans. Services Comput.*, early access, Oct. 30, 2018. doi: 10.1109/TSC.2018.2873561.

H. Yuan, J. Bi, and M. C. Zhou, “Spatial task scheduling for Cost Minimization in Distributed Green Cloud Data Centers,” *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 2, pp. 729–740, Apr. 2019.

Y. Quan, W. Chen, Z. Wu, and L. Peng, “Distributed fault detection for second-order delayed multi-agent systems with adversaries,” *IEEE Access*, vol. 5, pp. 16478–16483, 2017.

N. Megiddo, “Combinatorial optimization with rational objective functions,” *Math. Oper. Res.*, vol. 4, no. 4, pp. 414–424, Nov. 1979.

J. R. Correa, C. G. Fernandes, and Y. Wakabayashi, “Approximating a class of combinatorial problems with rational objective function,” *J. Math. Prog.*, vol. 40, no. 2, pp. 97–111, 2006.

T.-H. Wu, “A note on a global approach for general 0–1 fractional programing,” *Eur. J. Oper. Res.*, vol. 101, no. 1, pp. 220–223, Aug. 1997.

CPLEX. (2013). *IBM ILOG CPLEX Optimizer.* [Online]. Available: http://www-03.ibm.com/software/products/en/ibmicpolverstudi/

I. Saricicek and C. Celik, “Two meta-heuristics for parallel machine scheduling with segment splitting to minimize total tardiness,” *J. Appl. Math. Mod.*, vol. 35, no. 1, pp. 4117–4126, 2011.

J. SKORIN-KAPOV and A. J. Vakharia, “Scheduling a flow-line manufacturing system comprising of reconfigurable machines,” *IEEE Trans. Comput.*, vol. 26, no. 2, pp. 164–170, Apr. 1977.

O. MOSBAHI received the B.S. degree in computer science, the M.S. degree, and the Habilitation Diploma degree in computer science from the University of Tunis El Manar, Tunis, Tunisia, in 1999, 2002, and 2018, respectively, and the Ph.D. degree in computer science from the Institut National de Recherche en Informatique et Automatique (INRIA), National Polytechnic Institute of Lorraine, Nancy, France, in 2008. She was a part-time Researcher at INRIA, France, and a temporary Lecturer at Nancy II University, Nancy. She was also a Researcher at the Martin Luther University Halle-Wittenberg, Halle, Germany. She is currently an Associate Professor in computer science with the National Institute of Applied Science and Technology, University of Carthage, Tunis. She is active in several European projects and also in other interesting international collaborations.

MOHAMED KHALGUI received the B.S. degree in computer science from the Institut National de Recherche en Informatique et Automatique (INRIA), Rocquencourt, France, the ITIA-CNR Institute, Vigevano, Italy, and the Systems Control Laboratory, Xidian University, Xi’an, China. He was a Collaborator with the KACST Institute, Riyadh, Saudi Arabia, and the SEG Research Group, Patras University, Patras, Greece. He was a Professor with the National Institute of Applied Sciences and Technology, University of Carthage, Tunisia. He is currently a Professor with Jinan University, Zhuhai, China. He has been involved in various international projects and collaborations. He is a member of different conferences and boards of journals. He received the Humboldt Grant in Germany for preparing the Habilitation Diploma at the Martin Luther University of Halle-Wittenberg. The Chinese Government nominates him as a Talented Full Professor, in 2019. He got also the Outstanding Scientific Research Award SaiLaiLa, China, in 2019.
MENGCHU ZHOU (Fellow, IEEE) received the B.S. degree in control engineering from the Nanjing University of Science and Technology, Nanjing, China, in 1983, the M.S. degree in automatic control from the Beijing Institute of Technology, Beijing, China, in 1986, and the Ph.D. degree in computer and systems engineering from the Rensselaer Polytechnic Institute, Troy, NY, USA, in 1990. He joined the New Jersey Institute of Technology (NJIT), Newark, NJ, USA, in 1990, where he is currently a Distinguished Professor in electrical and computer engineering. He has over 800 publications, including 12 books, 500+ journal articles (400+ in IEEE TRANSACTIONS), and 29 book chapters. He holds 23 patents. His research interests are in Petri nets, intelligent automation, the Internet of Things, big data, Web services, and intelligent transportation. He is the founding Editor of the IEEE Press Book Series on Systems Science and Engineering and Editor-in-Chief of the IEEE/CAA JOURNAL OF AUTOMATICA SINICA. He served as an Associate Editor of the IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, and the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, and Editor of the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING. He served as a Guest-Editor for many journals including the IEEE INTERNET OF THINGS JOURNAL, the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, and the IEEE TRANSACTIONS ON SEMICONDUCTOR MANUFACTURING. He is currently an Associate Editor of the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, the IEEE INTERNET OF THINGS JOURNAL, the IEEE TRANSACTIONS ON SYSTEMS, MAN, and CYBERNETICS: SYSTEMS, AND FRONTIERS OF INFORMATION TECHNOLOGY AND ELECTRONIC ENGINEERING. He was the General Chair of the IEEE Conference on Automation Science and Engineering, Washington D.C., August 23–26, 2008, General Co-Chair of 2003 IEEE International Conference on System, Man and Cybernetics (SMC), Washington DC, October 5–8, 2003 and 2019 IEEE International Conference on SMC, Bari, Italy, October 6–9, 2019, Founding General Co-Chair of 2004 IEEE International Conference on Networking, Sensing and Control, Taipei, March 21–23, 2004, and the General Chair of 2006 IEEE International Conference on Networking, Sensing and Control, Ft. Lauderdale, Florida, USA, April 23–25, 2006. He was Program Chair of 2010 IEEE International Conference on Mechatronics and Automation, August 4–7, 2010, Xi’an, China, 1998 and 2001 IEEE International Conference on SMC and 1997 IEEE International Conference on Emerging Technologies and Factory Automation. He has led or participated in over 50 research and education projects with total budget over $12M, funded by National Science Foundation, Department of Defense, NIST, New Jersey Science and Technology Commission, and industry. He was a recipient of Excellence in Research Prize and Medal from NJIT, Humboldt Research Award for U.S. Senior Scientists from Alexander von Humboldt Foundation, and Franklin V. Taylor Memorial Award and the Norbert Wiener Award from IEEE SMC Society. He has been among most highly cited scholars for years and ranked top one in the field of engineering worldwide in 2012 by Web of Science. He is Fellow of International Federation of Automatic Control (IFAC), American Association for the Advancement of Science (AAAS) and Chinese Association of Automation (CAA).

ZHIWU LI (Fellow, IEEE) received the B.S. degree in mechanical engineering, the M.S. degree in automatic control, and the Ph.D. degree in manufacturing engineering from Xidian University, Xi’an, China, in 1989, 1992, and 1995, respectively. He joined Xidian University, in 1992, and he is currently with the Institute of Systems Engineering, Macau University of Science and Technology, Taipa, Macau. Over the past decade, he was a Visiting Professor at the University of Toronto, the Technion-Israel Institute of Technology, Martin Luther University Halle-Wittenburg, and Meliksah University. His current research interests include Petri net theory and its applications, and the supervisory control of discrete event systems and workflow.