Optimal Real-Time Scheduling on Two-Type Heterogeneous Multicore Platforms

Hoon Sung Chwa*, Jaebaek Seo*, Jinkyu Lee†, Insik Shin*‡
*School of Computing, KAIST, Republic of Korea
†Department of Computer Science and Engineering, Sungkyunkwan University, Republic of Korea
insik.shin@cs.kaist.ac.kr

***Abstract*—Motivated by the cutting-edge two-type heterogeneous multicore chips, such as ARM's big.LITTLE, that offer a practical support for migration, this paper studies the global (or fully-migrative) approach to two-type heterogeneous multicore scheduling. Our goal is to design an optimal fully-migrative scheduling framework. To achieve this goal in an efficient and simple manner, we break the scheduling problem into two subproblems: *workload assignment* and *schedule generation*. We propose a per-cluster workload assignment algorithm, called Hetero-Split, that determines the fractions of workload of each task to be assigned to both clusters without losing feasibility with the complexity of $O(n \log n)$, where n is the number of tasks. Furthermore, it provides a couple of important properties (e.g., a dual property) that help to generate an optimal schedule efficiently. We also derive scheduling guidelines to design optimal schedulers for two-type heterogeneous multicore platforms, called Hetero-Fair. By tightly coupling the solutions of Hetero-Split and Hetero-Fair, we develop the first optimal two-type heterogeneous multicore scheduling algorithm, called Hetero-Wrap, that has the same complexity ($O(n)$) as in the identical multicore case. Finally, concerning a practical point of view, we derive the first bounds on the numbers of intra- and inter-cluster migrations under two-type heterogeneous multicore scheduling, respectively.**

I. Introduction

Heterogeneous multicore platforms are composed of different types of cores, each type of which may have special capabilities such that execution rates are both core-type and task-specific. With rapid development in semiconductor technology, heterogeneous multicore designs are becoming an attractive solution to fulfill increasing performance demands while saving energy efficiently. For example, ARM recently has launched a two-type heterogeneous multicore chip, called big.LITTLE [1], which has been deployed in the state-of-the-art smartphones, e.g., Samsung Galaxy S6 and Note 4. The big.LITTLE architecture consists of two types of cores: one with high-performance "big" cores and the other with power-efficient "LITTLE" cores. Since both types of cores share the same instruction set architecture (ISA), tasks can be migrated on the fly from one to the other. Meanwhile, differences in the internal microarchitecture of big and LITTLE cores allow them to provide different execution rates as well as power consumption for each task.

Scheduling on such a heterogeneous multicore platform is much more challenging than scheduling on identical/uniform platforms[1] since the processing speed depends not only on the core type, but also on the task executed. Thus, on heterogeneous multicore platforms, a decision should be made on which type of core will execute a task over time.

There has been a growing interest in real-time heterogeneous multicore scheduling theories. Approaches to heterogeneous multicore real-time scheduling can broadly fall into three classes: *non-migrative*, *intra-migrative*, and *fully-migrative*. Under non-migrative scheduling, every task is assigned to a particular core and execute only on that core. On the other hand, every task is allowed to execute while migrating between cores of only the same type under intra-migrative scheduling, or migrating between cores of different types under fully-migrative scheduling.

Related Work. Both non-migrative and intra-migrative approaches divide the heterogeneous multicore scheduling problem into one of offline task allocation followed by runtime schedule generation. Once tasks are allocated to cores (for non-migrative scheduling) or core types (for intra-migrative scheduling), the scheduling problem is reduced to a collection of independent unicore or identical-multicore scheduling problems, which have been well studied with a body of successful results. However, the task-to-core and task-to-core-type assignment problems are known to be NP-hard in the strong sense [2], [3]. Consequently, much prior work on non-migrative and intra-migrative scheduling has been based upon heuristic approaches. For example, targeting non-migrative scheduling, some task-to-core assignment algorithms with approximation bounds were proposed for two-type heterogeneous multicore platforms [4], [5], [6] and for general m-type heterogeneous multicore platforms ($m > 0$) [7], [2]. Similarly, under intra-migrative scheduling, a task-to-core-type assignment approximation algorithm for two-type heterogeneous platforms was presented by Raravi et al. [8], and this work was later extended in [9] for m-type heterogeneous platforms.

The fully-migrative scheduling paradigm offers the benefit of enhanced schedulability, compared to the non-migrative/intra-migrative ones from a theoretical viewpoint. Correa et al. showed that if a task set can be scheduled by an optimal algorithm for fully-migrative scheduling, then an optimal algorithm for non-migrative scheduling needs cores four times faster [10]. Similarly, Raravi et al. [11] showed that an optimal algorithm for intra-migrative scheduling needs cores $1+\frac{\beta}{2}$ times faster, where $0 < \beta \leq 1$, against for fully-migrative scheduling on two-type heterogeneous platforms for a special case. In addition, Baruah [12] introduced a polynomial-time exact feasibility analysis method that is able to determine whether a task set can be scheduled on a m-type heterogeneous platform under fully-migrative scheduling via linear programming (LP). In the process of deriving the exact feasibility condition, he also

‡A corresponding author.

[1]On identical multicore platforms, all cores have exactly the same computing capacity. On uniform multicore platforms, different cores may have different processing speeds, but each individual task executing on a given core has the same execution rate (speed).

1052-8725/15 $31.00 © 2015 IEEE
DOI 10.1109/RTSS.2015.19

showed that such a schedule can be constructed by finding a maximal bipartite matching with the complexity of polynomial time. From a practical viewpoint, one of the major concerns with fully-migrative scheduling is the possibility of generating excessive migrations between clusters. However, it is shown that not all but only a bounded number of tasks need to be fully-migrative [12]. Yet, no bound on the number of migration has been derived.

This work. Our work is motivated by the advent of two-type heterogeneous multicore chips, such as big.LITTLE [1]. It offers a practical support for migration: big and LITTLE cores share not only the same ISA but also a specially designed interconnection bus for seamless data transfer between the big and LITTLE cores. This enables each task to migrate between cores of different types in the middle of execution with a practically reasonable cost [13], [14].

Therefore, in this paper, we focus on two-type heterogeneous multicore platforms, where a set of cores of the same type is referred to as a *cluster*. Our goal is to design an optimal fully-migrative scheduling framework. The main challenge on heterogeneous scheduling arises from the fact that the execution time of a task varies depending on how much portion of its workload is assigned to each different cluster, since each task may have a different execution rate on a different cluster. In order to handle this challenge in an easier way, we divide the scheduling problem into two sub-problems: *workload assignment* and *schedule generation*. The former sub-problem determines the amount of partial workload on each cluster for each individual task while preserving feasibility. Building upon a solution to the first sub-problem, the second sub-problem determines an actual schedule where all jobs meet their deadlines in an optimal manner. To address this problem with efficiency and simplicity, we view two-type heterogeneous multicore fully-migrative scheduling as a collection of identical multicore scheduling for each cluster while cooperatively handling what we call the NPE (No Parallel Execution) restriction — no single task executes on two clusters at the same time. Such a perspective makes it easier to develop an optimal scheduling algorithm efficiently, since it allows to build upon a body of successful results on identical multicore scheduling while mostly focusing on the additional issues pertaining to heterogeneous multicore scheduling.

This paper offers the following contributions. First, we propose an algorithm, called Hetero-Split, that finds a feasibility-optimal workload assignment, with the complexity of $O(n \log n)$, where n is the number of tasks. In the workload assignment, the tasks migrating between clusters hold a *dual* property except at most one, while utilizing the total computing capacities minimally. The dual property of a task plays a critical role in the second sub-problem of schedule generation, since it allows to reduce the difficulty of addressing the NPE restriction as well as provide more rooms to schedule tasks within a cluster.

Second, we derive simple guidelines to design optimal schedulers, called Hetero-Fair, by extending existing scheduling rules for optimal identical multicore scheduling toward two-type heterogeneous multicore scheduling.

Third, we develop the first optimal two-type heterogeneous multicore scheduling algorithm, called Hetero-Wrap, which exploits McNaughton's wrap-around rule [15], one of the simplest optimal identical multicore scheduling algorithms, and handles the NPE restriction with no extra cost mainly due to the dual property; the dual property of a task represents that a task can execute on a cluster exactly when it is idle on the other cluster, and vice versa. Thus, if a task has a dual property, its schedule on a cluster is simply obtained by reversing the schedule on the other cluster. Thereby, an optimal schedule on a two-type heterogeneous multicore platform can be obtained with the same complexity ($O(n)$) as in the identical multicore case. This is made possible by tightly coupling the solutions of Hetero-Split and Hetero-Fair.

Fourth, with the consideration on the heterogeneity of computing components, we separate task migration into two kinds: *intra-cluster migration* and *inter-cluster migration* and derive the first upper-bounds on the numbers of intra- and inter-cluster migrations under two-type heterogeneous multicore scheduling, respectively.

Finally, our simulation study demonstrates quantitative schedulability improvement by the proposed fully-migrative scheduling framework compared with the state-of-the-art approaches for intra-migrative and non-migrative scheduling.

II. Problem Statement and Approach Overview

We consider the problem of scheduling a task set τ of n implicit-deadline periodic tasks on a two-type heterogeneous multicore platform π under fully-migrative scheduling paradigm.

System model. We consider a two-type heterogeneous platform π consists of two clusters π_1 and π_2, where each cluster π_k comprises m_k identical cores of type-k ($k = 1, 2$). Each task $\tau_i \in \tau$ is characterized by a period T_i, an execution requirement C_i, and a pair of execution rates, r_i^1 and r_i^2. Such a task τ_i is assumed to generate a potentially infinite sequence of jobs every T_i time-units, with each job needing to complete C_i units of work within a relative deadline of T_i time-units.

Important parameters to describe the execution behavior of τ_i on heterogeneous platforms are its execution rates, r_i^1 and r_i^2. The value of r_i^k denotes the rate (or speed) at which work of task τ_i is completed on a core of type-k, indicating that executing τ_i on a core of type-k completes r_i^k units of work per unit of time. We assume that type-1 and type-2 cores are unrelated in the sense that r_i^1 and r_i^2 can be assigned any arbitrary positive real values for all tasks $\tau_i \in \tau$.

We assume that tasks are independent and preemptible, and migrate from one core to another within or across the cluster boundary during execution. Although our work takes both intra- and inter-cluster migrations into consideration, we employ the standard (incorrect) assumption that these occur at no cost or penalty. In actual systems, measured preemption and migration overheads can be accommodated by adjusting worst-case execution time requirements when upper-bounds on the numbers of preemptions and migrations are available. We also assume that a single job cannot be executed simultaneously upon more than one core, regardless of core type.

Two-type heterogeneous multicore scheduling. In this paper, we study the fully-migrative approach to two-type heterogeneous multicore scheduling, aiming to address the following problem, which we call the *optimal two-type heterogeneous multicore fully-migrative scheduling* problem:

Definition 1: Given a task set τ running on a two-type heterogeneous multi-core platform π, determine a schedule, if

one exists, under the fully-migrative scheduling paradigm such that all jobs of each task meet their deadlines for all possible legal job arrival sequences.

Heterogeneous multicore fully-migrative scheduling is intrinsically a much more difficult problem than identical/uniform multicore global scheduling due to the simple fact that each task may have a different execution rate for each core type. On a heterogeneous multicore platform, the execution time of a task varies depending on the amount of workload assigned to each core type. Thus, it is important to decide the "right" fraction of workload of each task to execute on each core type. Then, we also need to determine which tasks run on which types of core at any given instant under the restriction that no single task can run on multiple cores at the same time. In order to deal with such issues in a conceptually easier manner, we break the problem into two sub-problems: *workload assignment* and *schedule generation*. The workload assignment problem (P1) determines the amount of partial workload on each core type for each individual task while preserving feasibility, and the schedule generation problem (P2) generates an actual schedule where all jobs meet their deadlines in an optimal manner, when the workload assignment is addressed properly.

Now, we explain our approach to solve the two-type heterogeneous multicore fully-migrative scheduling problem. We first address the following problem, which we call the *feasible per-cluster task workload assignment* problem, with x_i^1 and x_i^2 denoting the fractions of workload C_i of τ_i that are assigned to type-1 and type-2 clusters, respectively, where $x_i^1 + x_i^2 = 1$.

Definition 2 (P1): Given a task set τ running on a two-type heterogeneous multicore platform π under fully-migrative scheduling, determine x_i^1 and x_i^2 for every $\tau_i \in \tau$ such that if τ is feasible on π, τ remains feasible on π even with the workload assignment of $\{x_i^1\}$ and $\{x_i^2\}$.

In this paper, we propose a feasibility-optimal per-cluster workload assignment algorithm, called Hetero-Split, that addresses Problem P1 correctly without losing feasibility. It is worth to note that a previous study [12] introduced a feasibility analysis method that is able to address the same problem for general m-type heterogeneous platforms (for some arbitrary $m > 0$) via linear programming (LP). In this paper, we show that an exact solution to Problem P1 can be obtained with a much lower complexity of $O(n \log n)$ when tailored to the two-type heterogeneous platform. In addition, we remark the solution preserves a couple of important properties that help to solve Problem P2 efficiently. Specifically, Hetero-Split enforces that the tasks migrating between clusters hold the dual property except at most one (Property 1) and bounds the number of such tasks (Property 2). Property 1 makes it easy to handle the NPE restriction, and Property 2 makes it possible to bound the number of inter-cluster migration.

Building upon an exact solution to Problem P1, we then seek to address the following problem, which we call the *optimal schedule generation* problem.

Definition 3 (P2): Given the per-cluster workload assignments $\{x_i^1\}$ and $\{x_i^2\}$ (derived by Hetero-Split), determine a schedule where all jobs of all tasks $\tau_i \in \tau$ meet their deadlines for all possible legal job arrival-sequences on a two-type heterogeneous platform π.

We note that, for any feasible workload assignment of a task set on a m-type heterogeneous multicore platform, Problem P2 can be transformed to one instance of the classical shop scheduling problems — preemptive open shop scheduling to minimize makespan [16], [12]. Its solution is known to have the complexity of polynomial time [16]. In this paper, we show that Problem P2 can be addressed in an optimal manner with a much lower cost of $O(n)$ — even with the same complexity as in the identical multicore case. This is made possible by exploiting the important properties that Hetero-Split derives.

By addressing the above two sub-problems: P1 and P2, we can achieve optimal fully-migrative scheduling for two-type heterogeneous multicore platforms. Since Problem P1 determines the per-cluster workload of each individual task while holding feasibility and Problem P2 generates schedules in an optimal way, our solution schedules all jobs in a task set without any deadline miss of a job as long as there exists a feasible solution.

III. Feasible Task Workload Assignment

In this section, we present our approach to determine the fractions of workload of each task on both clusters (i.e., $\{x_i^1\}$ and $\{x_i^2\}$), explained in Problem P1 in Section II. To this end, we introduce exact conditions for a task workload assignment to be feasible on a two-type heterogeneous multicore platform, present our feasibility-optimal per-cluster task workload assignment algorithm, and derive two key properties.

A. Feasibility conditions

We introduce necessary and sufficient conditions for a per-cluster task workload assignment to be feasible on a two-type heterogeneous multicore platform. We recall that x_i^1 and x_i^2 denote the fractions of workload C_i of τ_i that are assigned to type-1 and type-2 clusters, respectively, where $x_i^1 + x_i^2 = 1$. For a given value of x_i^j of task τ_i ($j \in [1, 2]$), a job of τ_i would execute $x_i^j \cdot C_i$ units of work on the type-j cluster at the execution rate of r_i^j with the execution time of $x_i^j \cdot C_i / r_i^j$. Then, we define the utilization of task τ_i on type-1 and type-2 clusters (denoted by u_i^1 and u_i^2) as follows:

$$u_i^1 = x_i^1 \cdot \frac{C_i}{r_i^1} \cdot \frac{1}{T_i}, \quad u_i^2 = x_i^2 \cdot \frac{C_i}{r_i^2} \cdot \frac{1}{T_i}. \tag{1}$$

Note that u_i^1 with $x_i^1 = 1$ is said to be the maximum possible utilization of τ_i's execution on the type-1 cluster, and denoted by $u_i^{1,max}$; likewise, $u_i^{2,max}$ denotes u_i^2 with $x_i^2 = 1$.

From the exact feasibility test [12] of fully-migrative heterogeneous multicore scheduling, the following conditions must hold in order for a per-cluster task workload assignment (i.e., $\{x_i^1\}$ and $\{x_i^2\}$) to be feasible on a two-type heterogeneous multicore platform, as follows:

$$\begin{aligned}
&\text{C1: } \forall \tau_i \in \tau, x_i^1 + x_i^2 = 1, \\
&\text{C2: } \forall \tau_i \in \tau, u_i^1 + u_i^2 \leq 1, \\
&\text{C3: } \sum_{\tau_i \in \tau} u_i^1 \leq m_1, \\
&\text{C4: } \sum_{\tau_i \in \tau} u_i^2 \leq m_2, \\
&\text{C5: } \forall \tau_i \in \tau, 0 \leq x_i^1, x_i^2 \leq 1.
\end{aligned}$$

Constraint C1 specifies that every task must receive its appropriate amount of execution. Constraint C2 represents

a necessary condition for each task to meet its deadline. Constraints C3 and C4 assert that total workload allocated on each cluster should be less than or equal to the capacity of each cluster. Baruah [12] proved that there is a per-cluster task workload assignment satisfying the conditions C1–C5 if and only if the task set τ on the two-type heterogeneous multicore platform π is feasible.

Since all those conditions are linear equalities/inequalities, finding a feasible per-cluster task workload assignment can be formulated as linear programming. We note that, in general, a linear program may have multiple optimal solutions that minimize (maximize) an objective function, and solving a linear program requires a polynomial time complexity. We aim to find a "particular" feasible per-cluster task workload assignment, which facilitates generating a schedule (addressed in P2), and find such a solution in an efficient manner without losing optimality for a two-type heterogeneous multicore platform.

B. Optimal per-cluster workload assignment algorithm

We develop a linearithmic time complexity ($O(n \log n)$) per-cluster task workload assignment, called Hetero-Split, that guarantees to find a feasible workload assignment on type-1 and type-2 clusters satisfying C1–C5, as long as there exists such a solution. According to the feasibility conditions C1–C5, there are two kinds of inequalities: one is related to the deadline constraint of each task (i.e., C2); the other is related to the capacity constraint of each cluster (i.e., C3 and C4). In order to find a feasible workload assignment in an efficient way, we consider those constraints, one by one. First, we only focus on the deadline constraint of each task and calculate the minimum fraction of workload of each task that should be assigned to either type-1 or type-2 cluster so as to satisfy the deadline constraint. Then, we consider assigning the rest of workload fractions solely focusing on the capacity constraint of each cluster.

Consideration on the deadline constraint. As a first step, we calculate the minimum fractions of workload of τ_i on type-1 and type-2 clusters in order to satisfy constraint C2. By the deadline constraint C2, if there exists a task τ_i such that $u_i^{1,max} > 1$ and $u_i^{2,max} > 1$ hold, the task cannot satisfy the constraint, which leads to infeasibility. However, if $u_i^{2,max} > 1$ and $u_i^{1,max} \leq 1$ hold, we may find a feasible solution by assigning some workload of τ_i to type-1 cluster. In this case, there exists the minimum fraction of workload of τ_i that should be assigned to the type-1 cluster so as to satisfy constraint C2. Conversely, if $u_i^{2,max} \leq 1$ and $u_i^{1,max} > 1$ hold, there exists the minimum fraction of workload of τ_i that should be assigned to the type-2 cluster with the same reasoning. If $u_i^{2,max} \leq 1$ and $u_i^{1,max} \leq 1$ hold, a task τ_i always satisfies constraint C2. Then, the following lemma calculates the minimum fractions of workload of τ_i that should be assigned to the type-1 and type-2 clusters (denoted by lo_i^1 and lo_i^2) in order to satisfy the constraint C2.

Lemma 1: For each task τ_i, the deadline constraint C2 is always satisfied, if

$$x_i^1 \geq lo_i^1, \;\; x_i^2 \geq lo_i^2$$

where lo_i^1 and lo_i^2 are calculated by

$$lo_i^1 = \begin{cases} \frac{u_i^{2,max}-1}{u_i^{2,max}-u_i^{1,max}}, & \text{if } u_i^{2,max} > 1, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

$$lo_i^2 = \begin{cases} \frac{u_i^{1,max}-1}{u_i^{1,max}-u_i^{2,max}}, & \text{if } u_i^{1,max} > 1, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Proof: In constraint C2, if we substitute x_i^2 to $1-x_i^1$ based on constraint C1, we can calculate lo_i^1. The same holds for lo_i^2. Due to space limitation, we refer readers to Appendix A in the supplementary file [17] for a full proof. ∎

Once lo_i^1 and lo_i^2 are calculated, the rest of workload fraction (i.e., $1 - lo_i^1 - lo_i^2$ by constraint C1) for each task can be properly allocated to each cluster as long as the cluster capacity constraints (C3 and C4) meet. We let y_i^1 and y_i^2 denote the workload fractions excluding lo_i^1 and lo_i^2 (i.e., $x_i^1 = y_i^1 + lo_i^1$ and $x_i^2 = y_i^2 + lo_i^2$), respectively. Then, the constraints C1–C5 for guaranteeing feasibility can be reduced as

$$\begin{aligned}
&\overline{\text{C1}}\text{: } \forall \tau_i \in \tau, y_i^1 + y_i^2 = 1 - lo_i^1 - lo_i^2, \\
&\overline{\text{C3}}\text{: } \sum_{\tau_i \in \tau} y_i^1 \cdot u_i^{1,max} \leq m_1 - \sum_i lo_i^1 \cdot u_i^{1,max}, \\
&\overline{\text{C4}}\text{: } \sum_{\tau_i \in \tau} y_i^2 \cdot u_i^{2,max} \leq m_2 - \sum_i lo_i^2 \cdot u_i^{2,max}, \\
&\overline{\text{C5}}\text{: } \forall \tau_i \in \tau, 0 \leq y_i^1, y_i^2 \leq 1 - lo_i^1 - lo_i^2.
\end{aligned}$$

Note that constraint C2 is removed. This is because constraint C2 is never violated if x_i^1 and x_i^2 are assigned at least lo_i^1 and lo_i^2 by Lemma 1, respectively.

Consideration on the capacity constraint. Since we reduce the problem by allocating lo_i^1 and lo_i^2 amount of fractions of workload to type-1 and type-2 clusters, respectively, the remaining step is to determine $\{y_i^1\}$ and $\{y_i^2\}$ such that $\overline{\text{C1}}$–$\overline{\text{C5}}$ is satisfied.

Each task has different execution behavior between clusters. A task τ_i consumes the capacity of $u_i^{1,max}$ if fully allocated on the type-1 cluster, and $u_i^{2,max}$ on the type-2 cluster. We define cf_i as τ_i's capacity efficiency ratio of type-1 cluster to type-2 cluster, expressed as

$$cf_i = \frac{u_i^{1,max}}{u_i^{2,max}}. \tag{4}$$

If $cf_i > 1$, executing τ_i on the type-2 cluster is more capacity-efficient than the type-1 cluster; on the contrary, if $cf_i < 1$, the converse holds. Thereby, if there is no capacity limit for each cluster, allocating all of the remaining workload of τ_i to its capacity-efficient cluster consumes the least capacity.

However, each cluster has its capacity limit as shown in constraints $\overline{\text{C3}}$ and $\overline{\text{C4}}$, so it might be impossible to allocate all τ_i with $cf_i > 1$ on the type-2 cluster (or all τ_i with $cf_i < 1$ on the type-1 cluster). Consequently, we need to rearrange each task workload allocation in order to satisfy cluster capacity limits.

The Hetero-Split algorithm. We now design our optimal per-cluster workload assignment algorithm (Hetero-Split) based on the understanding of per-task capacity efficiency on each cluster (see Algorithm 1). Hetero-Split works in three stages: 1) allocating lo_i^1 to type-1 and lo_i^2 to type-2 clusters to meet deadline constraints, 2) allocating the rest of workload fractions so as to consumes the minimum capacity assuming infinite capacity of both clusters, and 3) rearranging the workload

$m_1 = 3, m_2 = 2$	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
$(u_i^{1,max}, u_i^{2,max})$	(1.0, 0.3)	(0.9, 0.3)	(1.4, 0.6)	(1.8, 0.9)	(1.2, 0.8)	(0.8, 0.9)	(0.4, 0.7)
cf_i	3.3	3.0	2.3	2.0	1.5	0.9	0.6
u_i^1	0	0.3	0.7	0.2	0.6	0.8	0.4
u_i^2	0.3	0.2	0.3	0.8	0.4	0	0

TABLE I. TASK SET EXAMPLE

Algorithm 1 Optimal-Workload-Assignment (Hetero-Split)

1: stage 1:
2: **if** $\exists \tau_i : u_i^{1,max} > 1$ and $u_i^{2,max} > 1$ **then**
3: return not feasible
4: **end if**
5: Allocate $\{lo_i^1\}$, $\{lo_i^2\}$ according to Lemma 1
6: **if** $\sum_i lo_i^1 \cdot u_i^{1,max} > m_1 \vee \sum_i lo_i^2 \cdot u_i^{2,max} > m_2$ **then**
7: return not feasible
8: **end if**
9: stage 2:
10: $\Gamma^1 \leftarrow \{\tau_i | cf_i < 1\}$
11: $\Gamma^2 \leftarrow \{\tau_i | cf_i \geq 1\}$
12: Allocate $y_i^1 \leftarrow 1 - lo_i^1 - lo_i^2$, $y_i^2 \leftarrow 0$ for all tasks in Γ^1
13: Allocate $y_i^1 \leftarrow 0$, $y_i^2 \leftarrow 1 - lo_i^1 - lo_i^2$ for all tasks in Γ^2
14: stage 3:
15: **if** Both $\overline{C3}$ and $\overline{C4}$ are satisfied **then**
16: return $\{x_i^1 | x_i^1 = y_i^1 + lo_i^1\}$, $\{x_i^2 | x_i^2 = y_i^2 + lo_i^2\}$
17: **else if** Both $\overline{C3}$ and $\overline{C4}$ are not satisfied **then**
18: return not feasible
19: **else if** Only $\overline{C3}$ is satisfied **then**
20: **repeat**
21: find τ_k with the closest cf_k to 1 in Γ^2
22: **if** $\sum_i y_i^2 \cdot u_i^{2,max} - y_k^2 \cdot u_k^{2,max} > m_2 - \sum_i lo_i^2 \cdot u_i^{2,max}$ **then**
23: $y_k^1 \leftarrow 1 - lo_k^1 - lo_k^2$
24: $y_k^2 \leftarrow 0$
25: $\Gamma^2 \leftarrow \Gamma^2 \setminus \{\tau_k\}$
26: **else**
27: $y_k^1 \leftarrow \frac{\sum_i y_i^2 \cdot u_i^{2,max} - (m_2 - \sum_i lo_i^2 \cdot u_i^{2,max})}{u_k^{2,max}}$
28: $y_k^2 \leftarrow 1 - lo_k^1 - lo_k^2 - y_k^1$
29: **end if**
30: **if** $\overline{C3}$ is violated **then**
31: return not feasible
32: **end if**
33: **until** $\overline{C4}$ is satisfied
34: **else if** Only $\overline{C4}$ is satisfied **then**
35: Do the corresponding process to lines 20–33.
36: **end if**
37: return $\{x_i^1 | x_i^1 = y_i^1 + lo_i^1\}$, $\{x_i^2 | x_i^2 = y_i^2 + lo_i^2\}$

fractions excluding lo_i^1 and lo_i^2 to satisfy cluster capacity constraints.

In stage 1), if there exists a task τ_i such that $u_i^{1,max} > 1$ and $u_i^{2,max} > 1$, there is no feasible workload allocation, meaning that the task set is not feasible (lines 2–4). We calculate the minimum workload fractions (lo_i^1, lo_i^2) that should be allocated to type-1 and type-2 clusters according to Lemma 1 and allocate them on each cluster (line 5). After that, if no capacity remaining for accommodating the rest of workload for each cluster, there is no feasible workload allocation (lines 6–7).

In stage 2), Algorithm 1 partitions a task set into two groups according to capacity efficiency on a cluster. Let Γ^1 and Γ^2 denote a collection of tasks that are more capacity-efficient when executing on the type-1 and type-2 clusters, respectively (lines 10–11). Then, we allocate the rest of workload fractions (except lo_i^1, lo_i^2) of all tasks in Γ^1 to the type-1 cluster and the rest of workload fractions of all tasks in Γ^2 to the type-2 cluster (lines 12–13).

In stage 3), we check whether the allocation done by stage 2) satisfies cluster capacity constraints $\overline{C3}$ and $\overline{C4}$. There are 4 cases: i) if both $\overline{C3}$ and $\overline{C4}$ are satisfied, the allocation done by stage 2) is an optimal solution satisfying all feasibility conditions (lines 15–16); ii) if both $\overline{C3}$ and $\overline{C4}$ are not satisfied, there is no feasible workload allocation (lines 17–18); iii) if only $\overline{C3}$ is satisfied, it requires to move some workload fractions allocated on the type-2 cluster to the type-1 cluster until it satisfies $\overline{C4}$ (lines 19–33); and iv) if only $\overline{C4}$ is satisfied, it requires to move some workload fractions allocated on the type-1 cluster to the type-2 cluster until it satisfies $\overline{C3}$ (lines 34–36). In the process of rearranging workload fractions for cases iii) and iv), if no available type-1 cluster capacity to accommodate more remaining workload fractions, there is no feasible workload allocation (lines 30–32). The key issue in rearranging workload fractions is to choose some tasks whose workload fractions will be re-allocated. When some workload fractions are moved from the capacity-efficient cluster to the other one, capacity consumption is supposed to increase. Thus, we need to move the workload fractions in a way that the amount of increased capacity consumption arising from workload migration is minimized. We show that choosing tasks in the order of the closest cf_i to 1 not only minimizes the amount of increased capacity consumption, but also is beneficial for feasibility in Theorem 1.

Example 3.1: Let us consider a set of seven tasks and a two-type heterogeneous multicore platform comprising three cores of type-1 and two cores of type-2 (i.e., $m_1 = 3, m_2 = 2$). Each task is characterized by $(u_i^{1,max}, u_i^{2,max})$ and its capacity efficiency ratio cf_i as shown in Table I. According to the value of cf_i, we have $\Gamma^1 = \{\tau_6, \tau_7\}$, $\Gamma^2 = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$. After stage 2) of Algorithm 1, $\overline{C4}$ is not satisfied, while $\overline{C3}$ is satisfied. Then, at stage 3), the algorithm reallocates workload fractions in Γ^2 by moving from the type-2 cluster to the type-1 cluster in an increasing order of cf_k (i.e., the order of τ_5, τ_4, τ_3, and τ_2) until the total utilization on the type-2 cluster is equal to the capacity of the type-2 cluster (m_2). Since τ_5, τ_4 and τ_3 have $lo_5^2 = 1/2$, $lo_4^2 = 8/9$, and $lo_3^2 = 1/2$, the rest of workload except lo_i^2 will be moved to the type-1 cluster, that is $y_5^1 = 1/2$, $y_4^1 = 1/9$, and $y_3^1 = 1/2$. After that, the algorithm leaves some of workload of τ_2 such that the entire capacity of the type-2 cluster (m_2) is used and moves the rest of workload to the type-1 cluster, that is $y_2^1 = 1/3$ and $y_2^2 = 2/3$. The final results $(\{u_i^1\}, \{u_i^2\})$ is shown in Table I.

We now prove that Hetero-Split achieves *feasibility-optimality*.

Theorem 1: For a given task set τ running on a two-type heterogeneous multicore platform, Hetero-Split presented in Algorithm 1 always find $\{x_i^1\}$ and $\{x_i^2\}$ satisfying all feasibility requirements if such a solution exists.

Proof: We will show that if Algorithm 1 finds a solution, the solution satisfies all feasibility conditions, and otherwise, there is no feasible workload allocation.

In stage 1), according to Lemma 1, constraint C2 is never violated if x_i^1 and x_i^2 are assigned at least lo_i^1 and lo_i^2. After stage 1), Algorithm 1 allocates the rest of workload fractions (except lo_i^1 and lo_i^2) to the remaining capacity of type-1 and type-2 clusters. Then, there are 4 cases: i) both $\overline{\text{C3}}$ and $\overline{\text{C4}}$ are satisfied; ii) both $\overline{\text{C3}}$ and $\overline{\text{C4}}$ are not satisfied; iii) only $\overline{\text{C3}}$ is satisfied; iv) only $\overline{\text{C4}}$ is satisfied. We prove the feasibility-optimality for each case of i) – iv). For any variable X, we denote by ΔX the amount of the variation of X through the remaining of this proof. We consider the following task allocation process A:

A1. allocating workload of all tasks to the type-2 cluster
A2. moving tasks in the order of the smallest cf_i from the type-2 cluster to the type-1 cluster until $\sum_i u_i^2 = W_2$.

We prove that (a) process A minimizes $\sum_i u_i^1$ when the value of $\sum_i u_i^2$ is fixed as W_2 and (b) stage 3) of Algorithm 1 satisfies the feasibility-optimality for each case of i) – iv).

Proof of (a): we denote by $\Delta U^2 = \sum_i \Delta u_i^2$ the amounts of workload moved from the type-2 cluster to the type-1 cluster, where Δu_i^2 is each amount of workload of τ_i in ΔU^2. If we move ΔU^2 to the type-1 cluster during process A2, the change amount of the workload on the type-1 cluster (denoted as ΔU^1) is calculated as

$$\begin{aligned}\Delta U^1 &= \sum_i \Delta u_i^1 = \sum_i u_i^{1,max} \cdot \Delta y_i^1 \\ &= \sum_i u_i^{1,max} \cdot (-\Delta y_i^2) = \sum_i u_i^{1,max} \cdot (-\frac{\Delta u_i^2}{u_i^{2,max}}) \\ &= \sum_i -\Delta u_i^2 \cdot \frac{u_i^{1,max}}{u_i^{2,max}} \\ &= -\sum_i \Delta u_i^2 \cdot cf_i. \end{aligned} \tag{5}$$

For the same amount of $\Delta U^2 < 0$, in order to minimize ΔU^1, we should move tasks in the order of the smallest cf_i from the type-2 cluster to the type-1 cluster. Therefore, (a) is true.

Proof of (b):
In case i), all feasibility conditions are already satisfied.

In case ii), by (a), moving tasks in the order of the smallest cf_i from the type-2 cluster to the type-1 cluster minimizes U^1 but $\Delta U^1 > 0$ ($\because \Delta u_i^2 < 0 \rightarrow \Delta u_i^1 > 0$). Therefore, there is no way to decrease U^2 and U^1 at the same time (i.e., τ is not feasible in case ii)).

In case iii), the closest cf_i to 1 in the type-2 cluster corresponds to the smallest cf_i in the cluster since $\tau_i \in \Gamma_2$. By (a), moving tasks in the order of the closest cf_i to 1 in Γ^2 to the type-1 cluster minimizes U^1. Therefore, if $\overline{\text{C3}}$ is violated in the process of A2 where $W_2 = m_2$, τ is not feasible.

In case iv), Algorithm 1 satisfies the feasibility-optimality based on the same reasoning shown in case iii).

Therefore, Algorithm 1 satisfies feasibility-optimality. ■

C. Properties of Hetero-Split

Hetero-Split guarantees the following two key properties.

Property 1: Among tasks that are fractionally assigned to both type-1 and type-2 clusters (i.e., $0 < x_i^1, x_i^2 < 1$), there exists at most one task that has $u_i^1 + u_i^2 < 1$. The other tasks have $u_i^1 + u_i^2 = 1$.

Proof: In Algorithm 1, there are 4 cases: i) both $\overline{\text{C3}}$ and $\overline{\text{C4}}$ are satisfied; ii) both $\overline{\text{C3}}$ and $\overline{\text{C4}}$ are not satisfied; iii) only $\overline{\text{C3}}$ is satisfied; iv) only $\overline{\text{C4}}$ is satisfied.

In case i), all tasks in Γ^1 are entirely assigned to the type-1 cluster (i.e., $x_i^1 = 1$), and all tasks in Γ^2 are entirely assigned to the type-2 cluster (i.e., $x_i^2 = 1$). Therefore, there is no task that are fractionally assigned to both clusters.

In case ii), the task set is not feasible.

In case iii), Algorithm 1 iteratively finds τ_k with the closest cf_k to 1 in Γ^2 and moves some workload of τ_k from the type-2 to the type-1 cluster (lines 20–33). In each iteration, there exists two situations: the one (denoted by S1) where $\overline{\text{C4}}$ is not satisfied even if all of τ_k's workload except lo_k^2 are moved to the type-1 cluster (line 22); the other (denoted by S2) where $\overline{\text{C4}}$ is satisfied if some of τ_k's workload except lo_k^2 are moved to the type-1 cluster (line 26). Since the iteration ends when $\overline{\text{C4}}$ is satisfied, all tasks selected during the iteration except the last selected one belong to S1. In S1, all of τ_k's workload except lo_k^2 are moved to the type-1 cluster (lines 23–24). If $lo_k^2 = 0$ then τ_k is entirely assigned to the type-1 cluster (i.e., $x_k^1 = 1$). Otherwise, τ_k is fractionally assigned to both clusters with $u_k^1 + u_k^2 = 1$, because if $x_k^2 = lo_k^2$ and $x_k^1 = 1 - lo_k^2$ then $u_k^1 + u_k^2 = 1$ by Lemma 1. While assigning the last selected task, the algorithm assigns as much fraction of the task as possible to type-2 (i.e., the entire remaining capacity of type-1 cores is used), and the remaining fraction is assigned to type-1 cores. In this case, the task may have $u_k^1 + u_k^2 < 1$, because $x_k^2 > lo_k^2$. We note that the tasks that are not selected during iteration in Γ^2 are entirely assigned to the type-2 cluster. Therefore, the last selected task may have $u_k^1 + u_k^2 < 1$, and the other tasks that are fractionally assigned to both clusters have $u_k^1 + u_k^2 = 1$.

In case iv), Property 1 is true based on the same reasoning shown in case iii). ■

We note that a task τ_i has $u_i^1 + u_i^2 = 1$ if and only if a dual property holds for τ_i.

Property 2: The number of tasks that are fractionally assigned to both type-1 and type-2 clusters is at most $m_1 + m_2$.

Proof: We prove this property by contradiction. Suppose that there are $m_1 + m_2 + 1$ tasks that are fractionally assigned to both clusters. Then, by Property 1, at least $m_1 + m_2$ tasks have $u_i^1 + u_i^2 = 1$. The sum of utilization of those tasks on both clusters is $m_1 + m_2$. Then, there is no capacity remaining for the other task. Therefore, the task set is not feasible. ■

These two properties play a key role in developing the Hetero-Wrap scheduling algorithm and deriving the bound on the number of inter-cluster migrations presented in Section IV.

Complexity. Recall that we denote by n the number of tasks in a task set. In stages 1) and 2), calculating lo_i^1, lo_i^2 and determining Γ_1, Γ_2 require $O(n)$. In stage 3), sorting a task set requires $O(n \log n)$. Therefore, the time-complexity of Algorithm 1 is at most $O(n \log n)$.

IV. Optimal Schedule Generation

In the previous section, we discussed how to determine the fractions x_i^k of workload C_i of each task τ_i to execute

exclusively on the type-k cluster ($k = 1, 2$). In this section, we study how to schedule those workload fractions $x_i^k \cdot C_i$ on the type-k cluster, respectively, under the NPE restriction — no single task τ_i executes on two clusters at the same time. This may allow a perspective that considers the scheduling of τ on a two-type heterogeneous platform π as a collection of identical multicore scheduling of workload fractions ($x_i^k \cdot C_i$) on π_k under the NPE restriction. Such a perspective makes it easier to develop solutions efficiently, since it allows to build upon and elaborate a body of successful results on identical multicore scheduling while mostly focusing on how to handle the NPE restriction effectively. As a first attempt to explore such a perspective, we extend scheduling guidelines and algorithms for optimal identical multicore scheduling toward two-type heterogeneous scheduling. Then, as a result, we are able to give the first upper-bound on the number of inter-cluster migration under heterogeneous scheduling, extending the technique to bound the number of (intra-cluster) migration under identical scheduling.

A. The Hetero-Fair Guidelines

In this section, we aim to derive simple guidelines to design optimal schedulers for implicit-deadline periodic tasks running on a two-type heterogeneous multicore platform. To this end, we discuss how to extend the DP-Fair guidelines [18] that are designed for optimal identical multicore scheduling.

DP-Fair relies on the requirement of proportionate fairness that each task should execute proportionally to its utilization. DP-Fair shows that enforcing the fairness requirement only at job deadlines suffices to achieve optimal scheduling. It partitions time into slices based on deadlines of all jobs invoked by a task set (referred to as *deadline partitioning*). To ensure the fairness requirement at every deadline, each job is assigned its execution requirement proportional to its utilization within each time slice. We note that if every job can be executed constantly at a rate equal to its utilization (referred to as a *fluid* scheduling model), the fairness requirement can be easily satisfied for all jobs. However, it is impossible to implement such a fluid schedule on practical platforms since one core cannot execute more than one task simultaneously. Thereby, DP-Fair suggests three scheduling rules for designing practical schedulers to guarantee the optimality. Specifically, Rules 1 and 2 examine each individual task τ_i in order to determine whether τ_i must and must not execute at a given time instant, respectively, based on the concept of *local laxity*. Rule 3 compares the total amount of workloads remaining with the total available resource capacity so as to determine at least how many tasks must execute.

Now, we discuss how to extend the DP-Fair guidelines towards two-type heterogeneous multicore scheduling, when the workload of each task τ_i is split into two clusters at the ratio of x_i^1:x_i^2. There are two key issues to consider: 1) each workload portion ($x_i^k \cdot C_i$) should execute only on the type-k cluster, and 2) each cluster has its own processing capacity to execute the total amount of workloads assigned to it. Without proper scheduling decisions, it may end up with some undesirable situations, including (a) ones where a single task cannot complete a remaining workload at some point unless it executes on two clusters at the same time and (b) the others where the total amount of workload remaining on a cluster is larger than the total available capacity of the cluster. To resolve such issues, we extend DP-Fair in the following way. We extend Rules 1 and 2 to address the former (a) situations by introducing the concept of *task-level local laxity* and expand Rule 3 to deal with the latter (b) situations with the notion of *cluster-level local laxity*.

We here present our Hetero-Fair guidelines building upon DP-Fair. After deadline partitioning, the k-th time slice (denoted by σ_k) is $[t_{k-1}, t_k)$ of length $l_k = t_k - t_{k-1}$. Within the time slice σ_k, each task τ_i is then assigned its *local execution requirement* $u_i^1 \cdot l_k$ and $u_i^2 \cdot l_k$ on both type-1 and type-2 clusters, respectively. As scheduling decisions are made over time, the *local remaining execution* of task τ_i at time t in σ_k on type-1 and type-2 clusters is denoted by $R_i^1(t)$ and $R_i^2(t)$, respectively. At each time t, a task is said to be a *migrating* task when its remaining execution is on both type-1 and type-2 clusters (i.e., $R_i^1(t) > 0$ and $R_i^2(t) > 0$), and a task is said to be an *partitioned* task when its remaining execution is solely on either type-1 or type-2 cluster (i.e., $R_i^1(t) = 0$ or $R_i^2(t) = 0$). A migrating task at time t can become a partitioned one whenever no execution remains either type-1 or type-2 cluster after t.

We define the *task-level local laxity* of τ_i at time t (denoted by $L_i(t)$) as the difference between the remaining time in a time slice σ_k and the sum of remaining execution on each cluster before the time slice, and it is presented as

$$L_i(t) = (t_k - t) - (R_i^1(t) + R_i^2(t)). \tag{6}$$

At the beginning of a time slice σ_k, $R_i^1(t_{k-1})$ is $u_i^1 \cdot l_k$, and $R_i^2(t_{k-1})$ is $u_i^2 \cdot l_k$. Once a job of a task has zero task-level local laxity, it should always be executed on either the type-1 or type-2 cluster until the end of time slice; otherwise, the job will miss its deadline.

We define the *cluster-level local laxity* at time t, denoted by $L^1(t)$ ($L^2(t)$), as the difference between the total available capacity of the type-1 (type-2) cluster from t to at the end of a time slice and the total remaining workloads on the type-1 (type-2) cluster, expressed as

$$L^1(t) = m_1 \cdot (t_k - t) - \sum_i R_i^1(t), \tag{7}$$

$$L^2(t) = m_2 \cdot (t_k - t) - \sum_i R_i^2(t). \tag{8}$$

If cluster-level local laxity of the type-1 cluster at t (i.e., L^1(t)) reaches zero, all the type-1 cores should execute jobs until the end of the time slice; otherwise, at least one job will miss its deadline due to insufficient supply. The same holds for cluster-level local laxity of the type-2 cluster.

With the notions of task-level and cluster-level local laxities, we present our Hetero-Fair scheduling rules.

Definition 4: (Hetero-Fair scheduling for time slices) A scheduling algorithm belongs to Hetero-Fair if it schedules jobs within a time slice σ_k according to the following rules:

- Rule 1: Always run all the jobs with zero task-level local laxity (i.e., $L_i(t) = 0$);
- Rule 2: Never run a job with no workload remaining on both clusters in the slice (i.e., $R_i^1(t) = 0$ and $R_i^2(t) = 0$);
- Rule 3: Always allocate m_1 jobs on the type-1 cores at time t if its cluster-level laxity is zero (i.e., $L^1(t) = 0$), and allocate m_2 jobs on the type-2

cores at time t if its cluster-level laxity is zero (i.e., $L^2(t) = 0$);

We now prove that any Hetero-Fair scheduler is optimal on two-type heterogeneous multicore platforms.

Theorem 2: If a periodic implicit-deadline task set τ is feasible, any Hetero-Fair scheduling algorithm always schedules the task set without any deadline miss.

Proof: We prove this theorem by contradiction. Suppose a job of task τ_i misses its deadline when it is scheduled by Hetero-Fair. Then, there must exist the time slice σ_k and the earliest time t' in σ_k such that the job of τ_i has task-level local laxity of -1 at t' (i.e., $L_i(t') < 0$, $t_{k-1} \leq t' < t_k$). The job became zero laxity at $t'-1$, so it should be executed at $t'-1$ according to Rule 1. However, it fails to execute at $t'-1$. We denote by $\tau' = \{\tau_j | L_j(t'-1) = 0\}$ the task set of which a job has zero laxity at $t'-1$.

We consider two cases: (A) $|\tau'| > m_1 + m_2$, (B) $|\tau'| \leq m_1 + m_2$.

Case A: If $|\tau'| > m_1 + m_2$, then the sum of remaining execution on both the type-1 and type-2 clusters in $[t'-1, t_k)$ is larger than $(m_1 + m_2) \cdot (t_k - (t'-1))$ since the remaining execution of a zero laxity job at $t'-1$ is $t_k - (t'-1)$. Moreover, all cores in the type-1 and type-2 clusters should be busy in interval $[t_{k-1}, t'-1)$. If there is an idle core at t'' in $[t_{k-1}, t'-1)$, the only reason that a job of $\tau_j \in \tau'$ cannot be executed on the idle core is that the job is already executed on the other core at t''. It means $|\tau'| \leq m_1 + m_2$, and this contradicts the assumption that $|\tau'| > m_1 + m_2$. Therefore, the sum of the total workload allocated on each cluster in $[t_{k-1}, t'-1)$ and $[t'-1, t_k)$ (i.e. $[t_{k-1}, t_k)$) is larger than the sum of the total capacity of each cluster in $[t_{k-1}, t_k)$, meaning that at least either $\overline{C3}$ or $\overline{C4}$ are violated. This contradicts the assumption that τ is feasible.

Case B: We consider the case that $|\tau'| \leq m_1 + m_2$. The deadline miss job of τ_i was not executed at $t'-1$ even though the job has zero laxity, which implies that there is no job to execute except zero laxity jobs. Moreover, by assumption, the number of zero laxity jobs executed at $t'-1$ should be less than $m_1 + m_2$. Therefore, if $|\tau'| \leq m_1 + m_2$, there must exist at least one idle core at $t'-1$.

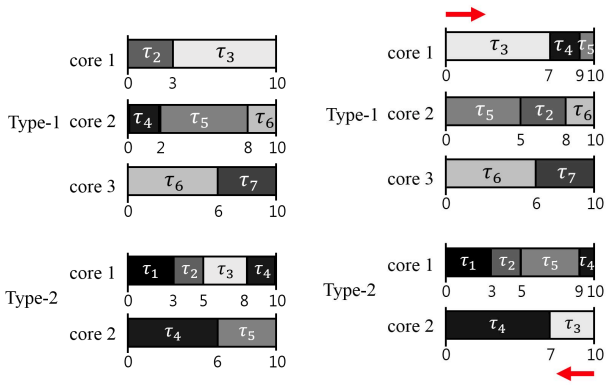
(Case B-1): In the case that task τ_i is a migrating task at $t'-1$ (i.e., $R_i^1(t'-1) > 0$ and $R_i^2(t'-1) > 0$), a job of τ_i should be executed on the idle core at $t'-1$. This contradicts that the job of τ_i has task-level local laxity of -1 at t'.

(Case B-2): In the case that task τ_i is a partitioned task at $t'-1$ (i.e., either $R_i^1(t'-1) = 0$ or $R_i^2(t'-1) = 0$). Without loss of generality, we assume $R_i^2(t'-1) = 0$.

If the type-1 cluster has an idle core at $t'-1$, a job of task τ_i should be executed at $t'-1$ by Rule 1. It contradicts that the job of τ_i has task-level local laxity of -1 at t'.

If the type-1 cluster has no idle core at $t'-1$, there must be more than m_1 partitioned zero laxity jobs on the type-1 cluster at $t'-1$ (If not, a job of τ_i should be executed at $t'-1$ by Rule 1). We define $\tau'' = \{\tau_j | L_j(t'-1) = 0 \wedge R_j^2(t'-1) = 0\}$. The sum of the remaining execution of all jobs of $\tau_j \in \tau''$ in $[t'-1, t_k)$ should be larger than $m_1 \cdot (t_k - (t'-1))$. Since $L^1(t'-1) < 0$, there must exist the latest time $t'' \in [t_{k-1}, t'-1)$ such that $L^1(t'') = 0$ (if not, $\forall t \in [t_{k-1}, t_k), L^1(t) < 0 \rightarrow \tau$ violates $\overline{C3}$). Then, there must be at least one idle core at t'' because $\forall t > t''$, $L^1(t) < 0$. The only reason that a job of $\tau_j \in \tau''$ cannot be executed on the idle core is that the job is already executed on the other core at t''. If such job of τ_j is a migrating one at t'', it must be executed on the type-1 cluster by Rule 3. If such job of τ_j is a partitioned one at t'', it is executed on the type-1 cluster. Thus, all jobs of tasks in τ'' are executed on the type-1 cluster, which implies $|\tau''| \leq m_1$, and this contradicts $|\tau''| > m_1$. This also holds for the case that $R_i^1(t'-1) = 0$.

Therefore, for all cases, we show that if Hetero-Fair fails to schedule a task set, the task set should be not feasible. ■



(a) A naive approach shows an incorrect schedule

(b) Hetero-Wrap shows a feasible schedule

Fig. 1. Schedules of a task set described in Table II in time interval [0,10).

B. The Hetero-Wrap Algorithm

Now, we develop an algorithm that implements the Hetero-Fair guidelines, called Hetero-Wrap. Building upon a per-cluster task workload assignment obtained by Hetero-Split, Hetero-Wrap exploits McNaughton's wrap-around rule [15] which is one of the simplest optimal scheduling algorithms for identical multicore platforms. The McNaughton's wrap-around rule constructs a schedule by sequentially allocating tasks to cores; this way ensures that each job is executed on at most one core at any time. If we apply McNaughton's wrap-around rule to each cluster separately, we can prevent a single job from parallel execution within a cluster. The main challenge is then how to avoid parallel execution of a job on both clusters (i.e., the NPE restriction). The following example shows that if we apply McNaughton's wrap-around rule to both clusters independently, we cannot necessarily avoid parallel execution of a single job on two clusters.

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
u_i^1	0	0.3	0.7	0.2	0.6	0.8	0.4
u_i^2	0.3	0.2	0.3	0.8	0.4	0	0

TABLE II. TASK SET EXAMPLE

Example 4.1: Let us consider a set of seven tasks and a two-type heterogeneous multicore platform comprising three cores of type-1 and two cores of type-2. The utilization of a task on each cluster is shown in Table II, which is assigned by Hetero-Split. Fig. 1(a) shows the schedule in a time slice $[0, 10)$, when we apply McNaughton's wrap-around rule to each cluster in an alphabetical order of tasks. For example, tasks to be executed in the type-1 cluster (i.e., $\{\tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7\}$) are

scheduled on core 1 of the type-1 cluster from time 0 to 10; once core 1 becomes full, the tasks are scheduled on core 2 and then core 3. Tasks to be executed in the type-2 cluster (i.e., $\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$) are scheduled in the same way. Then, the resulting schedule is incorrect in that there are some jobs that execute in both clusters at the same time. For example, τ_3 executes in $[5, 8)$ on both core 1 of type-1 cluster and core 1 of type-2 cluster.

To avoid the situation in the above example, Hetero-Wrap categorizes tasks into four subsets as follows: (i) a set M_a of migrating tasks τ_i with $u_i^1 + u_i^2 = 1$, (ii) a set M_b of migrating tasks τ_k with $u_k^1 + u_k^2 < 1$, (iii) a set P_1 of partitioned tasks on the type-1 cluster, and (iv) a set P_2 of partitioned tasks on the type-2 cluster.

For the same situation as Example 4.1, suppose that we apply McNaughton's wrap-around rule to the type-1 cluster from time 0 to 10, and the type-2 cluster from time 10 to 0 (the reverse order). To decide the order of tasks to be scheduled without parallel execution on two clusters, we utilize the following properties. First, each task τ_i in M_a has $u_i^1 + u_i^2 = 1$. Then, τ_i can execute on the type-1 cluster, exactly when it is idle on the type-2 cluster, and vice versa, which we call a dual property. Therefore, we can schedule a task in M_a without parallel execution, only when the current starting point[2] of the type-1 cluster is the same as that of the type-2 cluster. Also, after scheduling a task in M_a on both clusters, the new current starting point of the type-1 cluster is also the same as that of the type-2 cluster. For example, after we schedule τ_3 on both clusters, the current starting point for both clusters is 7, as shown in Fig. 1(b). Then, we can successfully schedule $\tau_4 \in M_a$ next; the new starting points for both cluster are 9.

This means that we should schedule tasks in M_a consecutively. Otherwise, before we schedule a task in M_a, we should find some tasks that make the current starting points of both clusters equal, which is difficult or impossible depending on utilization of unscheduled tasks. Then, the remaining issue is to determine the order of tasks in M_b and P_1 (or P_2). Since each task in M_b does not satisfy the dual property, we always check whether scheduling a task in M_b yields parallel execution on two clusters or not. If so, we need to some tasks to be scheduled prior to the task in M_b, which is also difficult or impossible. However, a task in M_b does not result in parallel execution on two clusters, as long as the task is followed by scheduling all tasks in M_a. This is because, after scheduling all tasks in M_a, the next starting point of the type-1 cluster is the same as that of the type-2 cluster. Therefore, we can easily schedule a single task in M_b; when it comes to multiple tasks in M_b, however, it is non-trivial to schedule those tasks with no parallel execution. To this end, we already designed Hetero-Split so as to yield at most one task in M_b by Property 1. Once we finish scheduling all tasks in M_a and a single task in M_b, we can schedule remaining tasks in P_1 and P_2 with any order. This is because, the tasks are irrelevant to cluster-level parallel execution.

Motivated by this, Hetero-Wrap constructs the offline schedules for the unit-time interval $[0, 1)$ for type-1 and type-2 clusters as follows. During the time interval, each task demands u_i^1 and u_i^2 time-unit executions on type-1 and type-2 clusters, respectively. Hetero-Wrap basically employs McNaughton's wrap-around rule [15] as follows:

- For type-1 cluster, we schedule tasks in the order of M_a, M_b, and then P_1. It schedules the i-th task on the first non-empty core, packing tasks from left to right. Suppose (i-1)-th task was scheduled on core k up to time instant t ($0 \leq t \leq 1$). Then, up to $(1 - t)$ time units of the i-th task are scheduled on core k and the remaining time units are scheduled on core $k + 1$ starting from time 0 (see Example 4.2).
- For type-2 cluster, we schedule tasks in the order of M_a, M_b, and then P_2[3]. It constructs the offline schedule in the same manner as in type-1 cluster scheduling, except only that the packing direction for each core is the other way around (i.e., from right to left).

Example 4.2: Fig. 1(b) depicts how the Hetero-Wrap algorithm generates a schedule in a time slice. The task set described in Table II can be partitioned as follows: $M_a = \{\tau_3, \tau_4, \tau_5\}$, $M_b = \{\tau_2\}$, $P_1 = \{\tau_6, \tau_7\}$, and $P_2 = \{\tau_1\}$. Fig. 1(b) shows the actual schedule in a time slice of length 10. The migrating tasks in M_a (i.e., τ_3, τ_4, τ_5) are always executed on either a type-1 or type-2 core at any time. For example, τ_3 is executed on core 1 of type-1 in $[0, 7)$ and on core 2 of type-2 in $[7, 10)$. We note that the tasks split into two (i.e., τ_5, τ_6, and τ_4) will be preempted and executed on two different cores in the same cluster. If the split task is migrating one (i.e., τ_5 and τ_4), inter-cluster migration will occur twice. For example, τ_5 will be executed first on core 2 of type-1 in $[0, 5)$, then migrated from type-1 to type-2, executed on core 1 of type-2 in $[5, 9)$, again migrated from type-2 to type-1, and finished core 1 of type-1. The other migrating tasks which are not split (i.e., τ_3, τ_2) will migrate between clusters only once.

Again, we would like to emphasize that Hetero-Wrap works because of the following reasons. First, Hetero-Wrap itself determines a proper order of tasks to be scheduled, by exploiting properties of task groups, e.g., the dual property for M_a. Second, Hetero-Split assigns proper workload such that Hetero-Wrap can generate schedule easily, e.g., the number of tasks in M_b is at most one.

We will show that our Hetero-Wrap algorithm satisfies all the Hetero-Fair guidelines in the following theorem. Specifically, we prove that Hetero-Wrap can provide a schedule such that each task executes on at most one core at each time instant, while all tasks finish all execution requirements on both type-1 and type-2 clusters at the end of each time slice.

Theorem 3: For a given feasible per-cluster workload assignment satisfying Property 1, Hetero-Wrap schedules all jobs in a task set without any deadline miss of a job.

Proof: We prove that all tasks can be correctly scheduled on each cluster in an interval of length 1. Then, it is clear that this is also true for any time slice. To this end, we show that each task is executed at most one core (regardless of core type) at each time instant.

Within a cluster, it is clear that each task is executed at most one core at each time instant, because Hetero-Wrap uses McNaughton's wrap-around rule for each cluster, separately. Since $u_i^1 \leq 1$ and $u_i^2 \leq 1$, $\forall \tau_i$, each task can be split onto at most two cores within a cluster and will not be executed on those cores at the same time.

[2]The current starting point is the time instant to start packing the current task when applying McNaughton's wrap-around rule.

[3]The task ordering in M_a is consistent with the type-1 cluster case.

Now, we prove that each migrating task will not be executed on two cores in different clusters at the same time. Without loss of generality, we assume that migrating tasks in M_a are indexed according to the order where McNaughton's wrap-around rule is used. We denote by s_i^j the time instant to start allocating τ_i and f_i^j the time instant to fish allocating τ_i on a type-j cluster when applying McNaughton's wrap-around rule ($0 \leq s_i^j, f_i^j \leq 1$). For each migrating task in M_a, we show that $s_i^1 = s_i^2$ and $f_i^1 = f_i^2$ by mathematical induction.

- Base step: for $i = 1$,

$$s_1^1 = 0 \bmod 1 \ , \ s_1^2 = 1 \bmod 1$$
$$f_1^1 = u_1^1 \ , \ f_1^2 = 1 - u_i^2$$

Then, $s_1^1 = s_1^2$ and $f_1^1 = f_1^2$, since $u_1^1 + u_1^2 = 1$.

- Inductive step: for some $i \geq 1$, suppose $s_i^1 = s_i^2$ and $f_i^1 = f_i^2$. Then, for $i+1$, we have

$$s_{i+1}^1 = f_i^1 \ , \ s_{i+1}^2 = f_i^2$$
$$f_{i+1}^1 = (s_{i+1}^1 + u_{i+1}^1) \bmod 1 \ , \ f_{i+1}^2 = (s_{i+1}^2 - u_{i+1}^2) \bmod 1.$$

According to distributive law, we have

$$\begin{aligned} f_{i+1}^1 &= (s_{i+1}^1 + u_{i+1}^1) \bmod 1 = (f_i^1 + u_{i+1}^1) \bmod 1 \\ &= f_i^1 \bmod 1 + u_{i+1}^1 \bmod 1, \end{aligned}$$

$$\begin{aligned} f_{i+1}^2 &= (s_{i+1}^2 - u_{i+1}^2) \bmod 1 = (f_i^2 - u_{i+1}^2) \bmod 1 \\ &= f_i^2 \bmod 1 - u_{i+1}^2 \bmod 1. \end{aligned}$$

Thus, $s_{i+1}^1 = s_{i+1}^2$ and $f_{i+1}^1 = f_{i+1}^2$, since $(u_{i+1}^1 + u_{i+1}^2) \bmod 1 = (f_i^1 - f_i^2) \bmod 1$.

Since, for each migrating task in M_a, $s_i^1 = s_i^2$ and $f_i^1 = f_i^2$ and McNaughton's wrap-around rule is applied to both clusters from opposite directions to each other, all migrating tasks in M_a always run on either type-1 or type-2 core without overlapping each task's running times on two clusters. After allocating tasks in M_a, at most one migrating task is left according to Property 1, which is in M_b. Then, its running times on two clusters will not overlap, because $s_i^1 = s_i^2$ and $u_i^1 + u_i^2 < 1$. ■

Migration bounds. We now calculate the maximum number of intra-cluster and inter-cluster migrations. To reduce task migrations, we use the mirroring technique [19] by reversing the order of tasks on each core in odd-numbered slices. Then, the tasks that execute last on each core at an even-numbered time slice will execute first at the next time slice without any migration. Therefore, task migration only occurs in the middle of a time slice, not between time slices. Looking at the example in Fig. 1(b), τ_5 runs for the first 5 of the time slice on core 2 of type-1, for the second 4 on core 1 of type-2, and for the last 1 on core 1 of type-1. If we reverse the ordering within each core for the next slice, then τ_5 will start on core 1 of type-1 for 1, execute on core 1 of type-2 for 4 and finish on core 2 of type-1 for 5.

According to Hetero-Wrap, there exist at most $m_1 - 1$ and $m_2 - 1$ tasks that are split into two when constructing a schedule for the type-1 and type-2 clusters, respectively. Those tasks will be preempted and executed on two different cores in the same cluster. In addition, inter-cluster migration is inevitable for all migrating tasks. According to Property 2, there are at most $m_1 + m_2$ migrating tasks. If any migrating task is split when constructing a schedule, the task will migrate between clusters twice: once from an initial cluster where they first run to the other one and again for returning to the initial cluster. Otherwise, migrating tasks will migrate between clusters only once. Then, we now provide upper-bounds on the intra-cluster and inter-cluster migrations in the following lemma.

Lemma 2: The Hetero-Wrap algorithm will produce at most $m_1 - 1$ and $m_2 - 1$ intra-cluster migrations on the type-1 and type-2 clusters, respectively, and $2 \cdot (m_1 + m_2) - 1$ inter-cluster migrations per slice.

Proof: With the mirroring technique, task migration only occurs in the middle of a time slice, never at the end. For each cluster, only split task will migrate once between two cores within the same cluster. There are at most $m_1 - 1$ and $m_2 - 1$ split tasks for type-1 and type-2 clusters. Therefore, at most $m_1 - 1$ and $m_2 - 1$ intra-cluster migrations will occur on the type-1 and type-2 clusters, respectively, per slice. All migrating tasks should migrate once between clusters. We note that the migrating task that is split into two when constructing a schedule will migrate once again between clusters. According to Property 2, there are at most $m_1 + m_2$ migrating tasks, and, in the worst case, all migrating tasks except the first task in the task ordering can be split into two. Therefore, at most $2 \cdot (m_1 + m_2) - 1$ inter-cluster migrations will occur per slice. ■

Time complexity. Partitioning a task set into M_a, M_b, P_1, and P_2 requires $O(n)$, and applying McNaughton's wrap-around rule to each cluster requires $O(n)$. Therefore, the time-complexity of Hetero-Wrap is $O(n)$.

V. Evaluation

In this section, we evaluate the performance of the proposed fully-migrative scheduling framework in comparison with the existing intra-migrative and non-migrative approaches for two-type heterogeneous multicore platforms. We first explain how to generate task sets for simulation and then present simulation results to compare their schedulability performance.

Simulation environment. We generate task sets and their running platforms as follows. We have three input parameters: a) the number of cores on each cluster (m_1, m_2), b) the number of tasks (n), and c) individual task parameters ($u_i^{1,max}$, $u_i^{2,max}$). The number of cores on each cluster is uniformly chosen in $\{2, 3, 4\}$. The number of tasks is uniformly chosen in $[m_1 + m_2, 25]$. For each task τ_i, $u_i^{1,max}$ and $u_i^{2,max}$ are uniformly chosen in $[0.1, 2.0]$. We note that the maximum utilization of τ_i on each cluster can be greater than one.

We generate $80,000$ feasible task sets by using an LP solver with the exact feasibility test [12]. We define the *total utilization rate* ($U_{\tau,\pi}$) of a task set on its running platform as $\frac{\sum_i (u_i^1 + u_i^2)}{m_1 + m_2}$. The minimum value of $U_{\tau,\pi}$ (denoted by $U_{\tau,\pi}^{min}$) can be obtained by setting the objective function of LP as minimizing $U_{\tau,\pi}$. We generate $10,000$ feasible task sets whose $U_{\tau,\pi}^{min} \in (p - 0.1, p]$, where we increase p from 0.3 to 1.0 in the step of 0.1, resulting in $80,000$ task sets.

Simulation results. For the generated task sets, we perform simulations for our fully-migrative scheduling framework (denoted by OUR). We compare OUR with two state-of-the-art approaches [8], [11] for intra-migrative and non-migrative

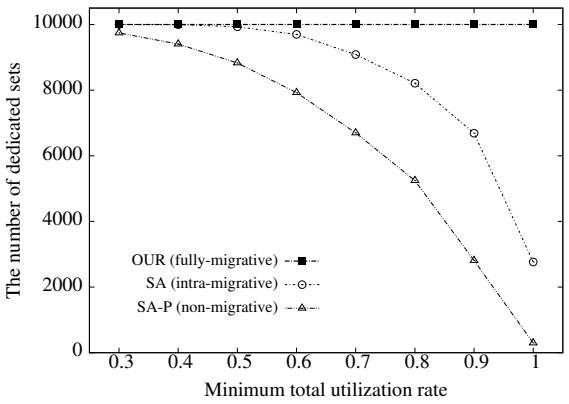


Fig. 2. The number of schedulable task sets by OUR, SA, and SA-P

scheduling, respectively, in terms of how many task sets are deemed schedulable by their own approaches.

In [8], [11], intra-migrative assignment (called SA) and non-migrative assignment (called SA-P) algorithms were proposed. If there exists a feasible intra-migrative assignment, SA (SA-P) finds such an intra-migrative (non-migrative) assignment if cores are $1+\frac{\alpha}{2}$ $(1+\alpha)$ times faster, where $0 < \alpha \leq 1$. Both SA and SA-P have the complexity of $O(n \log n)$.

Fig. 2 plots the number of task sets deemed schedulable by OUR, SA, and SA-P with different values of $U_{\tau,\pi}^{min}$. Basically, since OUR achieves optimality, OUR guarantees to find a feasible solution for all generated task sets, while SA and SA-P find 17% and 36% less task sets schedulable than OUR does. The performance gap between OUR and SA/SA-P becomes larger as $U_{\tau,\pi}^{min}$ increases. In particular, when $U_{\tau,\pi}^{min}$ is close to one, the improvements of OUR over SA and SA-P are 72% and 97%, respectively. We can interpret such a gap as the benefit of using fully-migrative scheduling compared to intra-migrative and non-migrative scheduling. OUR utilizes two-type heterogeneous resources more effectively by allowing task migration between cores of any type. As a result, OUR outperforms both intra-migrative and non-migrative approaches while retaining the same time-complexity.

VI. Conclusion

We addressed the optimal two-type heterogeneous multicore fully-migrative scheduling problem. We proposed Hetero-Split as a feasibility-optimal per-cluster workload assignment algorithm and introduced the Hetero-Fair scheduling guidelines to correctly schedule all tasks without any deadline miss on a two-type heterogeneous multicore platform. We then presented the Hetero-Wrap optimal two-type heterogeneous multicore scheduling algorithm that implements Hetero-Fair by exploiting McNaughton's wrap-around rule based on a property (i.e., a dual property) derived in Hetero-Split. Finally, we provided the first upper-bounds on the numbers of intra- and inter-cluster migrations under two-type heterogeneous multicore scheduling, respectively.

In this paper, we viewed fully-migrative scheduling on a two-type heterogeneous multicore platform from the perspective of a collection of identical multicore scheduling for each cluster while cooperatively handling the NPE restriction. As a first attempt to explore such a perspective, we extended one of the simplest optimal scheduling algorithms for identical multicore platforms toward two-type heterogeneous scheduling. Recently, new optimal scheduling algorithms, such as Bfair [20], RUN [21], U-EDF [22], and QPS [23], were proposed for identical multicore platforms with the aim of reducing the number of preemptions and migrations. As a future work, we will extend those advanced scheduling techniques to two-type heterogeneous scheduling with the additional consideration on two types of migration showing different costs, aiming at reducing the overall preemption and migration overheads. In addition, we only focused on implicit-deadline periodic task systems. We plan to extend the proposed approach to more general task systems such as constrained-deadline sporadic task systems, which is another direction of future work.

Acknowledgement

This work was supported in part by BSRP (NRF-2010-0006650, NRF-2012R1A1A1014930, NRF-2014R1A1A1035827), KEIT(2011-10041313), NCRC (2010-0028680), and IITP (B0101-15-0557) funded by the Korea Government (MEST/MSIP/MOTIE).

References

[1] ARM, "big.little technology: The future of mobile," 2013. [Online]. Available: http://www.arm.com/files/pdf/big-LITTLE-Technology-the-Futue-of-Mobile.pdf
[2] S. Baruah, "Task partitioning upon heterogeneous multiprocessor platforms," in *RTAS*, 2004.
[3] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*. Springer, 2012.
[4] B. Andersson, G. Raravi, and K. Bletsas, "Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors," in *RTSS*, 2010.
[5] G. Raravi and V. Nelis, "A PTAS for assigning sporadic tasks on two-type heterogeneous multiprocessors," in *RTSS*, 2012.
[6] B. Andersson and G. Raravi, "Provably good task assignment for two-type heterogeneous multiprocessors using cutting planes," *ACM Transactions on Embedded Computing Systems*, vol. 13(5), pp. 160:1–160:25, 2014.
[7] S. Baruah, "Partitioning real-time tasks among heterogeneous multiprocessors," in *Proceedings of the 2004 International Conference on Parallel Processing (ICPP)*, 2014.
[8] G. Raravi, B. Andersson, K. Bletsas, and V. Nelis, "Task assignment algorithms for two-type heterogeneous multiprocessors," in *ECRTS*, 2012.
[9] G. Raravi and V. Nelis, "Task assignment algorithms for heterogeneous multiprocessors," *ACM Transactions on Embedded Computing Systems*, vol. 13(5), pp. 159:1–159:26, 2014.
[10] J. R. Correa, M. Skutella, and J. Verschae, "The power of preemption on unrelated machines and applications to scheduling orders," *Mathematics of Operations Research*, vol. 37(2), pp. 379–398, 2012.
[11] G. Raravi, B. Andersson, V. Nelis, and K. Bletsas, "Task assignment algorithms for two-type heterogeneous multiprocessors," *Real-Time Syst*, vol. 50, pp. 87–141, 2014.
[12] S. Baruah, "Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms," in *RTSS*, 2004.
[13] ARM, "big.little processing with arm cortex-a15 and cortex-a7," 2011. [Online]. Available: http://www.arm.com/files/downloads/big_LITTLE_Final_Final.pdf
[14] T. S. Muthukaruppan, A. Pathania, and T. Mitra, "Price theory based power management for heterogeneous multi-cores," in *ASPLOS*, 2014.
[15] R. McNaughton, "Scheduling with deadlines and loss functions," *Management Science*, vol. 6, no. 1, pp. 1–12, 1959.
[16] T. Gonzalez and S. Sahni, "Open shop scheduling to minimize finish time," *Journal of the Association for Computing Machinery*, vol. 23(4), pp. 665–679, 1976.
[17] H. S. Chwa, J. Seo, J. Lee, and I. Shin, "Supplement: Optimal real-time scheduling on two-type heterogeneous multicore platforms," 2015. [Online]. Available: http://cps.kaist.ac.kr/CSL15/supplement.pdf
[18] G. Levin, F. Shelby, S. Caitlin, P. Ian, and B. Scott, "DP-Fair: A simple model for understanding optimal multiprocessor scheduling," in *ECRTS*, 2010.
[19] B. Andersson and E. Tovar, "Multiprocessor scheduling with few preemptions," in *RTCSA*, 2006, pp. 322–334.
[20] D. Zhu, X. Qi, D. Mosse, and R. Melhem, "An optimal boundary fair scheduling algorithm for multiprocessor real-time systems," *Journal of Parallel and Distributed Computing*, vol. 7, pp. 1411–1425, 2011.
[21] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt, "RUN: Optimal multiprocessor real-time scheduling via reduction to uniprocessor," in *RTSS*, 2011.
[22] G. Nelissen, V. Berten, V. Nelis, J. Goossens, and D. Milojevic, "U-EDF: An unfair but optimal multiprocessor scheduling algorithm for sporadic tasks," in *ECRTS*, 2012.
[23] E. Massa, G. Lima, P. Regnier, G. Levin, and S. Brandt, "Optimal and adaptive multiprocessor real-time scheduling: The quasi-partitioning approach," in *ECRTS*, 2014.