Resource-Aware Partitioned Scheduling for Heterogeneous Multicore Real-Time Systems

Jian-Jun Han

Huazhong Univ. of Sci. and Tech.
Wuhan, Hubei, China

Wen Cai

Huazhong Univ. of Sci. and Tech.
Wuhan, Hubei, China

Dakai Zhu

The Univ. of Texas at San Antonio
San Antonio, Texas, USA

ABSTRACT

Heterogeneous multicore processors have become popular computing engines for modern embedded real-time systems recently. However, there is rather limited research on the scheduling of real-time tasks running on heterogeneous multicore systems with shared resources. Note that, different partitionings of tasks upon heterogeneous cores can affect the synchronization overheads of tasks (and thus the system schedulability). Focusing on the partitioned-EDF scheduling and resource access protocol MSRP (Multiprocessor Stack Resource Policy), this paper proposes an effective synchronization aware task partitioning algorithm for heterogeneous multicores (SA-TPA-HM). Several resource-oriented heuristics are exploited to tighten the bound on the synchronization costs of tasks through dynamic task prioritization and to find an appropriate core for each task that can minimize the system utilization increment. The simulation results show that our proposed SA-TPA-HM scheme can achieve higher acceptance ratio (e.g., 60% more), when compared to the existing schemes designed for homogeneous multicores.

KEYWORDS

Embedded real-time systems, Heterogeneous multicore processor, Partitioned scheduling, Shared resources, Task synchronization

ACM Reference Format:

Jian-Jun Han, Wen Cai, and Dakai Zhu. 2018. Resource-Aware Partitioned Scheduling for Heterogeneous Multicore Real-Time Systems. In *DAC'18: The 55th Annual Design Automation Conference 2018, June 24–29, 2018, San Francisco, CA, USA*. ACM, New York, NY, USA, 6 pages.

1 INTRODUCTION

With the goal of satisfying application performance while maintaining power efficiency of computing systems, heterogeneous multicore processors (such as Intel's QuickIA platform [4] and ARM big.LITTLE [15]) have emerged as popular computing engines for diverse embedded real-time applications, e.g., consumer multimedia [8] and video object tracking [10]. For tasks co-running on multicore systems, they may need to access shared resources (e.g., shared data objects) exclusively and such resource access contention can lead to the problem of *task synchronization*. For real-time applications, the task synchronization problem becomes more salient as

it can cause *priority inversion* [14]. The priority inversion stemmed from the resource access race can delay the execution of high-priority tasks and thus may result in degraded schedulability of tasks.

To address the problem of task synchronization, several lock based resource access protocols have been investigated. For uniprocessors, Sha *et al.* developed a Priority Ceiling Protocol (PCP) [14] for fixed priority scheduling and Baker proposed a Stack Resource Policy (SRP) [1] for dynamic priority scheduling. Based on the principle of these protocols for uniprocessors, MSRP [5], MPCP [11] and several protocols [2, 3] have been extended and explored for multiprocessors. Moreover, several studies on the resource-aware real-time scheduling have been reported in [6, 7, 17]. Although the real-time scheduling for heterogeneous multicores has been studied [12, 16], there is very rare work on the scheduling of real-time tasks that access shared resources in heterogeneous multicore systems.

It is noteworthy that improper mappings of tasks on heterogeneous cores can cause larger synchronization overheads of tasks and inflated system utilization, which can degrade the system schedulability significantly. In this work, for periodic real-time tasks that access shared resources, by incorporating several resource-oriented heuristics, we study a synchronization-aware task partitioning algorithm for heterogeneous multicores (SA-TPA-HM). We adopt the resource access protocol MSRP [5] since it has simpler implementation and better schedulability performance than other protocols (such as MPCP [11]). Specifically, with the reduced synchronization costs of tasks, SA-TPA-HM exploits a dynamic priority method to order those unmapped tasks by computing their maximum estimated task utilizations in an iterative manner. We further study an effective task allocation heuristic for better schedulability, where each task is assigned onto the core that leads to the minimum system utilization increment under partitioned-EDF and MSRP.

The remainder of this paper is organized as follows. Section 2 presents system models and necessary preliminaries about resource access protocol MSRP. In Section 3, the task mapping algorithm SA-TPA-HM is proposed. Simulation results are given and discussed in Section 4. Section 5 concludes the paper.

2 SYSTEM MODELS AND PRELIMINARIES

We consider a heterogeneous multicore system that consists of M processor cores ($\{\mathcal{P}_1, \dots, \mathcal{P}_M\}$) with different execution frequencies (speeds) [12, 16], where the speed of processor core $\mathcal{P}_i$ is denoted as f_i . For ease of presentations, we assume that the processor core with smaller index has slower speed (i.e., $f_1 \leq f_2 \leq \dots \leq f_M$) and f_1 is *normalized* to 1.

A set of N periodic real-time tasks $\Psi = \{\tau_1, \dots, \tau_N\}$ run on the system with their initial arrivals at time 0. Every task τ_i has a period (i.e., relative deadline) p_i . That is, the j^{th} job of task τ_i arrives at time $(j-1) \cdot p_i$ and has to finish by its absolute deadline $j \cdot p_i$. The worst-case execution time (WCET) of task τ_i based on the *lowest*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
DAC '18, June 24–29, 2018, San Francisco, CA, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5700-5/18/06...\$15.00
<https://doi.org/10.1145/3195970.3196103>

speed of cores (i.e., 1) is denoted as c_i . Then, the WCET of task τ_i on core $\mathcal{P}_m$ is $\frac{c_i}{f_m}$. With the focus on the partitioned scheduling, the EDF scheduler is deployed on each core $\mathcal{P}_m$ ($m = 1, \dots, M$) and schedules all its tasks in the subset Ψ_m . Then, we have $\Psi = \bigcup_{m=1}^M \Psi_m$. As there is at most one active job for any real-time task at any time, we use *task* and *job* interchangeably in the remainder of this paper without specified otherwise.

This work considers shared data (e.g., global variables) among tasks, instead of hardware resources (such as bus, cache and network) that can be handled by their corresponding schemes [9, 18]. With shared resources being considered, the system is assumed to have a set of R resources $\mathcal{R} = \{\mathcal{R}_1, \dots, \mathcal{R}_R\}$ that can be shared by all tasks without deploying dedicated cores for resources. The access to any resource by a task is exclusive and non-preemptable, i.e., a resource can be held and accessed by only *one* task at any time. Moreover, we focus on *non-nested* critical sections as the nested ones occur rarely in practice and can be tackled by group locks [2]. There are n_i critical sections in task τ_i and its x^{th} critical section is denoted as $z_{i,x}$, where task τ_i accesses resource $r_{i,x} (\in \mathcal{R})$ for the maximum time $c_{i,x}$ based on the lowest speed of cores.

2.1 Resource Access Protocol MSRP

We first briefly review the principles of MSRP [5] where its basic rules can be summarized as follows:

- When a task τ_i on core $\mathcal{P}_m$ (i.e., $\tau_i \in \Psi_m$) issues a request for a resource $\mathcal{R}_a$, if resource $\mathcal{R}_a$ is free, task τ_i locks and then accesses it non-preemptively; otherwise, if resource $\mathcal{R}_a$ is currently held and accessed by another task from different core, task τ_i is appended to the FIFO queue of resource $\mathcal{R}_a$ and the core $\mathcal{P}_m$ enters a busy-waiting status;
- Once task τ_i completes accessing resource $\mathcal{R}_a$, it unlocks resource $\mathcal{R}_a$ and becomes preemptable again. If resource $\mathcal{R}_a$'s FIFO queue is not empty (i.e., there exist tasks on other cores waiting for accessing resource $\mathcal{R}_a$), its header task is de-queued and then locks the resource.

From the above rules of MSRP, we can see that the execution of a task τ_i on core $\mathcal{P}_m$ can be blocked due to resource access contention at two different scenarios as follows:

- When task τ_i requests access to a resource $\mathcal{R}_a$ that is currently accessed by a task from another core, it has to wait in the FIFO queue of $\mathcal{R}_a$ and such duration caused by inter-core synchronization interference is denoted as *global waiting time* (or *busy-waiting time*);
- When a low-priority task on core $\mathcal{P}_m$ is busy-waiting for or holding a resource, task τ_i is locally blocked and such duration is denoted as *local blocking time*.

For periodic tasks that access shared resources (protected within their non-nested critical sections) and are scheduled by EDF and MSRP, we can have the following properties [5]:

PROPERTY 1. *For any core at any given time, there exists at most one task that is either holding a resource or busy-waiting for a resource that is currently held by a task on different core.*

PROPERTY 2. *Any job of a task can be blocked by a low-priority job on the same core at most once.*

PROPERTY 3. *A job's local blocking time is bounded by the longest duration of any low-priority job on the same core that busy-waits for (if applicable) and accesses a resource once.*

2.2 Schedulability Condition for MSRP

For a given mapping of tasks upon cores $\Pi = \{\Psi_1, \dots, \Psi_M\}$, where Ψ_m is the subset of tasks assigned to core $\mathcal{P}_m$, we next give the sufficient schedulability condition of partitioned-EDF with MSRP for heterogeneous systems. For the ease of descriptions, some important notations used in this paper are defined below:

- $BW_{i,x}$: the maximum amount of *global waiting time* that task τ_i on core $\mathcal{P}_m$ spends on busy-waiting for resource $\mathcal{R}_a$ in its critical section $z_{i,x}$ ($r_{i,x} = \mathcal{R}_a$). Recall that the WCET of every task is based on f_1 ($f_1 = 1$) here. Based on Property 1, with the different speeds of cores being considered, $BW_{i,x}$ can be calculated as [5, 6]:

$$BW_{i,x} = \sum_{k=1, \dots, M}^{k \neq m} tp_k^{max}(\mathcal{R}_a), \quad (1)$$

$$tp_k^{max}(\mathcal{R}_a) = \max\{tt_l^{max}(\mathcal{R}_a) | \forall \tau_l \in \Psi_k\},$$

$$tt_l^{max}(\mathcal{R}_a) = \max\left\{\frac{c_{l,y}}{f_k} \mid \forall z_{l,y} : r_{l,y} = \mathcal{R}_a\right\},$$

where $tp_k^{max}(\mathcal{R}_a)$ is the longest time for any task from another core $\mathcal{P}_k$ to access resource $\mathcal{R}_a$ *once*, $tt_l^{max}(\mathcal{R}_a)$ denotes the maximum time for $\tau_l (\in \Psi_k)$ to access $\mathcal{R}_a$ *once*;

- BW_i : the maximum *total global waiting time* taken by task τ_i to access all its resources, i.e., $BW_i = \sum_{x=1}^{n_i} BW_{i,x}$;
- B_i : the maximum *local blocking time* experienced by task τ_i on core $\mathcal{P}_m$. Based on Property 3, by taking into account the speed of core $\mathcal{P}_m$, we can have [1, 5]:

$$B_i = \max\left\{BW_{j,y} + \frac{c_{j,y}}{f_m} \mid \forall z_{j,y} : \tau_j \in \Psi_m \wedge p_j > p_i\right\}. \quad (2)$$

Based on the results from [1, 5], a partition Π for periodic tasks that access shared resources in a *homogeneous* multicore system is feasible under the partitioned-EDF scheduling with MSRP if, for every core $\mathcal{P}_m$ ($m = 1, \dots, M$), there is:

$$\forall \tau_i \in \Psi_m, \frac{B_i}{p_i} + \sum_{\substack{p_j \leq p_i \\ \forall \tau_j \in \Psi_m}} \frac{c_j + BW_j}{p_j} \leq 1.$$

With the focus on different speeds of cores, for a given task-to-core mapping $\Pi (\{\Psi_1, \dots, \Psi_M\})$, the synchronization aware *core utilization* of core $\mathcal{P}_m$ is defined as:

$$U(\Psi_m) = \max\left\{\frac{B_i}{p_i} + \sum_{\substack{p_j \leq p_i \\ \forall \tau_j \in \Psi_m}} \frac{\frac{c_j}{f_m} + BW_j}{p_j} \mid \forall \tau_i \in \Psi_m\right\}. \quad (3)$$

The *system utilization* is defined as $U(\Pi) = \max\{U(\Psi_m) | \forall \mathcal{P}_m\}$. Based on the same reasonings as in [1, 5], we can directly have the following proposition concerning the feasibility of partitioned-EDF with MSRP for heterogeneous multicore systems.

PROPOSITION 2.1. *For a set Ψ of periodic real-time tasks executing on a heterogeneous shared resource multicore system that are scheduled by partitioned-EDF and MSRP, a given task partition Π is feasible when there is $U(\Pi) \leq 1$.*

ALGORITHM 1: Outline of SA-TPA-HM

Input: Ψ (the task set);**Output:** A feasible task-to-core partition Π or FAIL;

```

1  $\Pi = \emptyset; \Psi_k = \emptyset$  ( $k = 1, \dots, M$ );  $\Gamma = \Psi$ ;
2 while ( $\Gamma \neq \emptyset$ ) do
3    $\tau_i = \text{SelectTask}(\Gamma)$ ; //Section 3.3;
4    $m = \text{FindCore}(\tau_i, \Pi)$ ; //Section 3.4;
5   if ( $m == \text{null}$ ) then
6      $\Pi = \emptyset$ ; exit;
7   else
8      $\Psi_m = \Psi_m \cup \{\tau_i\}$ ;  $\Gamma = \Gamma - \{\tau_i\}$ ;  $\Pi = \{\Psi_1, \dots, \Psi_M\}$ ; Update
       $U(\Psi_k)$  ( $k = 1, \dots, M$ ); //Section 3.2;
9   end
10 end
11 Return ( $\Pi \neq \emptyset ? \Pi$ : FAIL);

```

Notice that Equation (3) considers the worst case synchronization overheads of tasks on any core. Therefore, if a partition fails $U(\Pi) \leq 1$, it may not mean the task set is not schedulable. Here, the task set is deemed as NOT feasible under the condition of Proposition 2.1.

Problem Description: for a set Ψ of periodic real-time tasks that access shared resources in a heterogeneous multicore system, find a feasible task-to-core mapping Π where tasks on every core are schedulable under EDF and MSRP.

Clearly, when the system is homogeneous without shared resources, the problem actually reduces to the classical bin-packing problem, which is known to be NP-hard in the strong sense. Hence, the partition problem considered in this work is NP-hard as well.

3 PARTITIONING ALGORITHM SA-TPA-HM

For partitioned scheduling, there are generally two essential phases when designating tasks to cores: a) determine the order of tasks to be allocated; and b) find appropriate cores to accommodate tasks. Considering the varying synchronization overheads of tasks for different mappings of tasks on heterogeneous cores, we propose a synchronization-aware task partitioning algorithm for heterogeneous multicores (SA-TPA-HM) to improve tasks' schedulability.

3.1 Overview of SA-TPA-HM

The outline of SA-TPA-HM is summarized in Algorithm 1. First, the corresponding parameters are initialized in line 1. For all un-mapped tasks in the set Γ , we select the highest-priority task one at a time, where the priorities of un-mapped tasks are updated subject to their iteratively estimated utilizations (line 3). Then, the target core is chosen based on the system utilization increment for M possible task allocations (line 4). If the current partial mapping is feasible, the utilizations of all cores are updated accordingly (line 8). Finally, SA-TPA-HM either fails or returns the feasible partition Π (line 11). We present the detailed heuristics in what follows.

3.2 Reduced Waiting Times for Allocated Tasks

For a task τ_i , different from the rather pessimistic approach to calculating its total global waiting time by simply accumulating $BW_{i,x}$ (see (1)) as in [5, 6], SA-TPA-HM intends to reduce the maximum busy-waiting times and maximum estimated ones for allocated tasks

ALGORITHM 2: CalBWMax($\tau_i (\in \Psi_m), \mathcal{R}_a (\in \Phi_i)$)

Input: $\tau_i, \mathbb{S}_a^i, \Pi = \{\Psi_1, \dots, \Psi_M\}$;**Output:** $BW_{i,\mathcal{R}_a}^{max}$;

```

1  $BW_{i,\mathcal{R}_a}^{max} = 0$ ;  $limit[x] = |\mathcal{S}_{i,a}|$  ( $x = 1, \dots, M \wedge x \neq m$ );
2 for ( $z_{j,y} \in \mathbb{S}_a^i$ ) do
3   if ( $\exists x, \tau_j \in \Psi_x \wedge x \neq m$ ) then
4      $num = \min\{\pi_{i,j}, limit[x]\}$ ;
5     if ( $num == 0$ ) then
6       continue;
7     else
8        $BW_{i,\mathcal{R}_a}^{max} + = \frac{num \cdot c_{j,y}}{f_x}$ ;  $limit[x] - = num$ ;
9     end
10  end
11 end

```

and un-mapped tasks respectively in a **resource-oriented** way. Recall that the initial arrival time of any task is 0. We first analyze the limits on inter-core synchronization interference (due to accessing the same resource) for tasks on different cores. Such inter-core interference among tasks can be presented as follows [7].

PROPOSITION 3.1. For two periodic tasks τ_i and τ_j on different cores scheduled by partitioned-EDF, the maximum number of task τ_j 's jobs that can interfere with the execution of task τ_i 's any job due to resource access contention is:

$$\pi_{i,j} = \begin{cases} 1, & p_i < p_j \wedge \text{mod}(p_j, p_i) = 0 \\ \frac{p_i}{p_j}, & p_i \geq p_j \wedge \text{mod}(p_i, p_j) = 0 \\ \left\lceil \frac{p_i}{p_j} \right\rceil + 1, & \text{otherwise} \end{cases}$$

where $\text{mod}(x, y)$ indicates the remainder of dividing x by y .

Define Φ_i as the subset of resources that task τ_i needs to access. The set of task τ_i 's critical sections where task τ_i accesses resource $\mathcal{R}_a (\in \Phi_i)$ is denoted as $\mathcal{S}_{i,a} = \{z_{i,j} | r_{i,j} = \mathcal{R}_a; j = 1, \dots, n_i\}$. Then, based on Property 1, the number of inter-core synchronization interference that can be experienced by any job of task $\tau_i (\in \Psi_m)$ due to accessing resource $\mathcal{R}_a$ has the following two limits:

- $|\mathcal{S}_{i,a}|$: the number of critical sections in the set $\mathcal{S}_{i,a}$;
- $\pi_{i,j}$: the maximum number of any task τ_j 's jobs ($\tau_j \notin \Psi_m$) that need to access resource $\mathcal{R}_a$ during the execution of task τ_i 's any job.

With the focus on the limits of inter-core interference among tasks, we explore a resource-oriented approach that reduces the maximum total busy-waiting time $BW_{i,\mathcal{R}_a}^{max}$ for *allocated* task $\tau_i (\in \Psi_m)$ to access its **each resource** $\mathcal{R}_a (\in \Phi_i)$. We further define $\mathbb{S}_a^i$ as the subset of critical sections of all tasks other than task τ_i where resource $\mathcal{R}_a$ can be accessed, i.e., $\mathbb{S}_a^i = \cup_{\tau_j \in \Psi, j \neq i} \mathcal{S}_{j,a}$. Here, the critical sections in the set $\mathbb{S}_a^i$ are ordered by their non-increasing sizes and the tie is broken arbitrarily.

For a given task-to-core mapping Π , the detailed steps to compute $BW_{i,\mathcal{R}_a}^{max}$ for task τ_i on core $\mathcal{P}_m$ to access resource $\mathcal{R}_a$ are shown in Algorithm 2. The remaining limits on the number of interference from other cores are first initialized (line 1). For critical section $z_{j,y}$ where task τ_j accesses $\mathcal{R}_a$ on different core $\mathcal{P}_x$ ($x \neq m$), the number of interference on task τ_i follows the limits from τ_j (i.e.,

$\pi_{i,j}$) and the remaining limits from $\mathcal{P}_x$ (i.e., $\text{limit}[x]$) (line 4). If there is no such interference (line 5), the next critical section in the set $\mathbb{S}_a^i$ is to be checked (line 6); otherwise, $BW_{i,\mathcal{R}_a}^{max}$ accumulates and the remaining interference limits from core $\mathcal{P}_x$ (i.e., $\text{limit}[x]$) is updated (line 8). Finally, $BW_{i,\mathcal{R}_a}^{max}$ is returned once all critical sections in the set $\mathbb{S}_a^i$ have been tested.

Then, the reduced maximum total busy-waiting time BW_i^{max} of allocated task τ_i can be found as $BW_i^{max} = \sum_{\mathcal{R}_a \in \Phi_i} BW_{i,\mathcal{R}_a}^{max}$. Clearly, we know that BW_i^{max} obtained by Algorithm 2 is no larger than that computed based on (1). Hence, with the tightened bound on the busy-waiting times of tasks, we can have lower core utilizations to improve the feasibility test as given in Proposition 2.1.

3.3 Utilization Estimates of Unmapped Tasks

It is well-known that the partitions generated by the utilization decreasing policy can achieve better schedulability [13]. Based on the feasibility condition for MSRP as presented in Proposition 2.1, we see that for each task, in addition to its WCET, its global waiting time and local blocking time can also affect the core utilization. Intuitively, the global waiting time plays a more important role in the core utilization compared to local blocking time based on their definitions. Then, following the principle of utilization decreasing policy, the **estimated utilization** for an *unmapped* task τ_i can be defined as $u_i^{est} = (c_i + BW_i^{est})/p_i$, where BW_i^{est} is the estimated total busy-waiting time for task τ_i to access all its resources.

For unmapped task τ_i , we try to compute the worst-case BW_i^{est} , where task τ_i is assumed to be assigned to the fastest core (i.e., $\mathcal{P}_M$) and can experience the worst-case global waiting caused by the tasks from other slower cores (i.e., $\mathcal{P}_1, \dots, \mathcal{P}_{M-1}$). As the global waiting time of a task relies on a specific task-to-core partition, getting minimum BW_i^{est} of unmapped task τ_i has the same time complexity as the partitioned problem. To reduce BW_i^{est} with acceptable time complexity, we first obtain the set of critical sections $C_{i,a}$, which can cause the worst-case inter-core interference on each resource $\mathcal{R}_a (\in \Phi_i)$ that task τ_i needs to access. Here, a critical section may appear multiple times in the set $C_{i,a}$. The basic steps to obtain the set $C_{i,a}$ are shown in Algorithm 3.

Similar to Algorithm 2, Algorithm 3 relies on the current mapping Π and inter-core interference limits. Specifically, an additional total remaining limit (i.e., *total*) is employed since the number of inter-core interference on task τ_i when it accesses resource $\mathcal{R}_a$ is at most $(M-1) \cdot |S_{i,a}|$ based on Property 1. First, the remaining limits on the number of interference from slower cores (i.e., $\mathcal{P}_1, \dots, \mathcal{P}_{M-1}$) and the set $C_{i,a}$ are initialized (line 1). Algorithm 3 omits the critical sections of tasks that have been allocated to core $\mathcal{P}_M$ (line 4), and handles the interference limits from mapped tasks (line 6) and unmapped ones (line 8) differently. Here, for the critical section in un-allocated task τ_j whose exact assignment is unknown till now, the number of its interference on task τ_i complies with the limits from $\pi_{i,j}$, $|S_{i,a}|$ and *total*. Moreover, for those critical sections considered, they are added to the set $C_{i,a}$ individually subject to the interference limits (lines 10 to 12). As the critical sections in the set $\mathbb{S}_a^i$ are sorted by their non-ascending sizes, the critical sections in the set $C_{i,a}$ can be obtained by the same order. When there exists no remaining interference (line 13) or all critical sections in the set $\mathbb{S}_a^i$ have been tested, the resulting $C_{i,a}$ is returned.

ALGORITHM 3: GetBWSet($\tau_i, \mathcal{R}_a (\in \Phi_i)$)

Input: $\tau_i, \mathbb{S}_a^i, \Pi = \{\Psi_1, \dots, \Psi_M\}$;
Output: $C_{i,a}$;
1 $C_{i,a} = \emptyset$; $\text{limit}[x] = |S_{i,a}|$ ($x = 1, \dots, M-1$);
2 $\text{total} = (M-1) \cdot |S_{i,a}|$;
3 **for** ($\forall z_{j,y} : z_{j,y} \in \mathbb{S}_a^i \wedge \tau_j \notin \Psi_M$) **do**
4 **if** ($\tau_j \notin \Psi_M$) **then**
5 **if** ($\exists x, \tau_j \in \Psi_x$) **then**
6 $\text{num} = \min\{\text{total}, \pi_{i,j}, \text{limit}[x]\}$; $\text{limit}[x] - = \text{num}$;
7 **else**
8 $\text{num} = \min\{\text{total}, \pi_{i,j}, |S_{i,a}|\}$;
9 **end**
10 **for** ($k = 0; k < \text{num}; k++$) **do**
11 $C_{i,a} = C_{i,a} \cup \{z_{j,y}\}$;
12 **end**
13 **if** ($(\text{total} - = \text{num}) == 0$) **then**
14 **exit**;
15 **end**
16 **end**
17 **end**

ALGORITHM 4: CalBWEst($\tau_i, \mathcal{R}_a (\in \Phi_i)$)

Input: $\tau_i, C_{i,a}, \Pi = \{\Psi_1, \dots, \Psi_M\}$;
Output: $BW_{i,\mathcal{R}_a}^{est}$;
1 $BW_{i,\mathcal{R}_a}^{est} = 0$; $\text{limit}[x] = |S_{i,a}|$ ($x = 1, \dots, M-1$);
2 **for** ($z_{j,y} \in C_{i,a}$) **do**
3 **if** ($\exists x, \tau_j \in \Psi_x$) **then**
4 $BW_{i,\mathcal{R}_a}^{est} + = \frac{c_{j,y}}{f_x}$; $\text{limit}[x] - =$; $C_{i,a} = C_{i,a} - \{z_{j,y}\}$;
5 **end**
6 **end**
7 **for** ($x = 1; x \leq M-1; x++$) **do**
8 **while** ($\text{limit}[x] \neq 0$) **do**
9 **if** ($C_{i,a} \neq \emptyset$ and $c_{j,y}$ is the first element in $C_{i,a}$) **then**
10 $BW_{i,\mathcal{R}_a}^{est} + = \frac{c_{j,y}}{f_x}$; $\text{limit}[x] - =$; $C_{i,a} = C_{i,a} - \{z_{j,y}\}$;
11 **else**
12 **return**;
13 **end**
14 **end**
15 **end**

Again, to find the estimated busy-waiting time of task τ_i in the worst case, the task with larger critical sections in $C_{i,a}$ needs to be allocated to a slower core subject to its interference limits. Based on $C_{i,a}$ obtained by Algorithm 3, the detailed steps to compute $BW_{i,\mathcal{R}_a}^{est}$ for un-allocated task τ_i to access its resource $\mathcal{R}_a$ are shown in Algorithm 4. Algorithm 4 is comprised of two basic steps: step one computes the estimated global waiting time $BW_{i,\mathcal{R}_a}^{est}$ caused by tasks that are already allocated (lines 2 to 6); second step calculates $BW_{i,\mathcal{R}_a}^{est}$ contributed by unmapped tasks (lines 7 to 15). Recall that the critical sections in the set $C_{i,a}$ are ordered by their non-increasing sizes. Algorithm 4 tries to obtain the maximum $BW_{i,\mathcal{R}_a}^{est}$ by assuming that the unmapped task with larger critical sections is allocated to slower core (line 7). Based on the inter-core interference limit, if there is no interference from a core, the remaining critical sections in the set $C_{i,a}$ are processed on the next faster core

(line 8). Otherwise, if the set $C_{i,a}$ is not empty (i.e., there still exists inter-core interference), $BW_{i,\mathcal{R}_a}^{est}$ accumulates correspondingly (line 10). Once all critical sections in the set $C_{i,a}$ have been processed, the resulting $BW_{i,\mathcal{R}_a}^{est}$ is returned (line 12).

Thus, the maximum estimated total busy-waiting time BW_i^{est} for task τ_i to access all its resources is $BW_i^{est} = \sum_{\mathcal{R}_a \in \Phi_i} BW_{i,\mathcal{R}_a}^{est}$. Then, the estimated utilizations of all unmapped tasks can be updated accordingly and they are prioritized by their non-increasing new estimated utilizations (where the tie is broken arbitrarily). Finally, the task having the highest priority (i.e., the largest estimated utilization) among unmapped tasks is chosen to be allocated next.

3.4 Selection of Target Core

The second phase of partitioned scheduling is the selection of a core for a given task. To find an appropriate core to accommodate the candidate task τ_i , we study an effective heuristic for better schedulability of the task system, where every task allocation should lead to the **minimized increase** in the system utilization.

Based on (3), we define the *new* utilization of core $\mathcal{P}_m$ with a new task τ_i as $U(\Psi_m \cup \{\tau_i\})$. If $U(\Psi_m \cup \{\tau_i\}) > 1$, task τ_i does not fit on core $\mathcal{P}_m$ and there is $U(\Psi_m \cup \{\tau_i\}) = \infty$. Note that, after task τ_i is mapped to a core $\mathcal{P}_m$, the utilizations of other cores may change by this new task due to task synchronization. Then, for another core $\mathcal{P}_x$, the synchronization overheads of all its tasks need to be updated accordingly and its new core utilization is denoted as $U^{\tau_i}(\Psi_x)$. Similarly, $U^{\tau_i}(\Psi_x) = \infty$ if $U^{\tau_i}(\Psi_x) > 1$. Therefore, the *new* system utilization with task τ_i on core $\mathcal{P}_m$ is defined as:

$$U(\Pi \cup \{\tau_i\}) = \max\{U(\Psi_m \cup \{\tau_i\}), \max_{\forall x : x \neq m} \{U^{\tau_i}(\Psi_x)\}\} \quad (4)$$

Based on the feasibility condition as shown in Proposition 2.1, assuming that τ_i is allocated to core $\mathcal{P}_m$, the system utilization increment for this mapping is defined as $\Delta_{\tau_i, \mathcal{P}_m}^{sys} = U(\Pi \cup \{\tau_i\}) - U(\Pi)$. By probing M possible assignments for task τ_i and updating the *new* utilizations for all cores based on (4), the core $\mathcal{P}_k$ that results in *valid* minimum system utilization increment (i.e., $\Delta_{\tau_i, \mathcal{P}_k}^{sys} \leq 1$) is chosen. If there is no feasible core for task τ_i , SA-TPA-HM fails and returns null. When there exists more than one such core, the tie-breaking prefers the core with larger index (i.e., the faster core).

4 EVALUATIONS AND DISCUSSIONS

In this section, we evaluate the performance of the proposed SA-TPA-HM through extensive simulations. For comparison, we extended the existing synchronization-aware WFD (SA-WFD) by taking the different speeds of cores into account [6]. To assess the effectiveness of the task ordering policy exploited by SA-TPA-HM (see Section 3.3), we also implemented SA-TPA-HM without task ordering (SA-TPA-HM-WTO), where those unmapped tasks are sorted by their non-increasing *raw* task utilizations (i.e., $\frac{c_i}{p_i}$ for each task τ_i). We compare these mapping schemes based on the performance metric: *schedulability ratio*, which is defined as the ratio of the number of task sets having feasible partitions under these schemes over the total number of tested task sets generated as follows.

4.1 Simulation Settings

In the experiments, we consider the following parameters as summarized in Table 1. The system parameters include the number of cores (M), the speeds of cores and the number of shared resources (R). The average speed of cores is denoted as $f^{avg} = \frac{\sum_{i=1}^M f_i}{M}$. The *normalized system raw utilization* (which ignores the synchronization costs) is denoted as $NSRU = \frac{\sum_{i=1}^N \frac{c_i}{p_i}}{M \cdot f^{avg}}$. For tasks, we consider the number of tasks N , periods of tasks, the number of critical sections in a task and the critical section ratio CSR (defined as the ratio of the total size of a task's critical sections to its WCET).

For given M , N , $NSRU$, CSR and the speeds of cores, the synthetic task sets are generated as follows. The raw utilization of a task τ_i is set as $u_i = \frac{NSRU \cdot M \cdot f^{avg}}{N}$. Then, the period p_i is randomly chosen from one of the three types of periods as shown in Table 1. The number of critical sections n_i in task τ_i is obtained in the range $[1, 10]$ following a uniform distribution and each resource that task τ_i accesses is randomly selected. Moreover, c_i is obtained in the range $[x \cdot p_i \cdot u_i, (2-x) \cdot p_i \cdot u_i]$ randomly. Similarly, the execution time of a critical section is randomly generated within $[\frac{x \cdot c_i \cdot CSR}{n_i}, \frac{(2-x) \cdot c_i \cdot CSR}{n_i}]$. Unless otherwise noted, the default parameter values are: $x = 0.2$, $M = 8$, $N = 32$, $R = 4$, $f^{avg} = 2.5$, $NSRU = 0.65$ and $CSR = 0.02$. For the results reported below, each data point corresponds to the average of 30,000 task sets.

Table 1: System and task parameters for the simulations

Parameters	Values/ranges
Number of cores (M)	[2, 16]
Speeds of cores	[1, 4]
Norm. sys. raw util. ($NSRU$)	[0.4, 0.75]
Number of resources (R)	[1, 10]
Number of tasks (N)	[8, 64]
Periods of tasks	[50, 200], [200, 500], [500, 2000]
Number of CSs per task (n_i)	[1, 10]
Critical section ratio (CSR)	[0.005, 0.05]

4.2 Simulation Results

Figure 1(a) shows the impacts of different $NSRU$ on the performance of the mapping schemes. To clearly exhibit the performance comparison among the schemes, $NSRU$ is chosen within [0.4, 0.75]. Generally, a larger $NSRU$ represents higher system workload. SA-TPA-HM considers the influence of the system heterogeneity and synchronization costs of tasks on the feasibility. Moreover, it tries to minimize the system utilization increment for each task allocation. Thus, SA-TPA-HM has the highest acceptance ratio, especially for higher system workload (e.g., $NSRU \geq 0.7$). As SA-TPA-HM uses a dynamic task ordering policy, it can obtain better performance compared to SA-TPA-HM-WTO. Moreover, SA-TPA-HM-WTO outperforms SA-WFD, which focuses on the similarity of tasks accessing the same resources in homogeneous multi-cores while employing the pessimistic global waiting times for (unmapped and allocated) tasks and neglecting the effects of system heterogeneity on their synchronization overheads (thus the system utilization) during task-to-core mappings.

Figure 1: Schedulability performance of the mapping algorithms.

Usually, larger CSR indicates higher resource access contention among tasks with other parameters being fixed. As can be seen from Figure 1(b), when CSR becomes larger, the acceptance ratio of SA-WFD drops quickly; in contrast, SA-TPA-HM exhibits relatively steady performance due to its resource-guided heuristics for heterogeneous systems as discussed in Section 3. Similarly, compared to SA-TPA-HM-WTO, SA-TPA-HM can achieve higher acceptance ratio thanks to its iterative prioritization approach for un-assigned tasks to increase the schedulability of the task system.

Figure 1(c) depicts the performance of the mapping schemes when the number of resources R varies. Intuitively, smaller R implies more synchronization requirements in the system when other parameters are fixed. Similar to the trends as shown in Figure 1(b), when R becomes smaller, SA-TPA-HM and SA-TPA-HM-WTO can have much better schedulability performance compared to SA-WFD based on the above-mentioned reasons.

Figure 1(d) further illustrates the performance comparison with varying number of cores M (where $N = 4 \cdot M$). When other parameters are fixed, more cores often means more inter-core synchronization interference among tasks. For this case (e.g., $M = 16$), SA-TPA-HM and SA-TPA-HM-WTO can obtain much higher acceptance ratio (e.g., 60% more) compared to SA-WFD.

5 CONCLUSIONS

For periodic real-time tasks executing on heterogeneous multicore systems with shared resources that are scheduled by partitioned-EDF and MSRP, we first study a resource-guided approach to tightening the bound on the synchronization overheads of tasks. With the different speeds of cores and synchronization costs of tasks being considered, the proposed SA-TPA-HM uses an iterative ordering scheme to prioritize unmapped tasks based on their maximum estimated utilizations. SA-TPA-HM further guarantees that each task mapping can minimize the increment in system utilization. Finally, the simulation results confirm the effectiveness of SA-TPA-HM.

ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China (NSFC) Awards 61472150, the Fundamental Research Funds for the Central Universities (China) (HUST: 2016YXMS081), and US National Science Foundation Awards CNS-1422709 and CNS-1421855.

REFERENCES

- [1] T. P. Baker. 1991. Stack-based scheduling for realtime processes. *Real-Time System* 3, 1 (1991), 67–99.
- [2] A. Block, H. Leontyev, Björn B Brandenburg, and J.-H. Anderson. 2007. A Flexible Real-Time Locking Protocol for Multiprocessors. In *Int'l Conference on Embedded and Real-Time Computing Systems and Applications*. 47–56.
- [3] A. Burns and A. J. Wellings. 2013. A Schedulability Compatible Multiprocessor Resource Sharing Protocol – MrSP. In *Euromicro Conference on Real-Time Systems*. 282–291.
- [4] N. Chitlur, G. Srinivasa, S. Hahn, P K. Gupta, D. Reddy, D. Koufaty, P. Brett, A. Prabhakaran, L. Zhao, and N. Ijhi. 2012. QuickIA: Exploring heterogeneous architectures on real prototypes. In *International Symposium on High-Performance Computer Architecture*. 1–8.
- [5] P. Gai, M. Di Natale, G. Lipari, A. Ferrari, C. Gabellini, and P. Marceca. 2003. A comparison of MPCP and MSRP when sharing resources in the Janus multiprocessor on a chip platform. In *Real-Time and Embedded Technology and Applications Symposium*. 189–198.
- [6] J.-J. Han, X. Wu, D. Zhu, H. Jin, L. T. Yang, and J.-L. Gaudiot. 2012. Synchronization-aware energy management for VFI-based multicore real-time systems. *IEEE Trans. on Computers* 61, 12 (2012), 1682–1696.
- [7] J.-J. Han, D. Zhu, X. Wu, L. T. Yang, and H. Jin. 2014. Multiprocessor Real-Time Systems with Shared Resources: Utilization Bound and Mapping. *IEEE Trans. Parallel Distrib. Syst.* 25, 11 (2014), 2981–2991.
- [8] M. Happe, E. Bbers, and M. Platzner. 2013. A self-adaptive heterogeneous multi-core architecture for embedded real-time video object tracking. *J. of Real-Time Image Processing* 8, 1 (2013), 95–110.
- [9] L. S. Indrusiak, J. Harbin, and A. Burns. 2015. Average and worst-case latency improvements in mixed-criticality wormhole networks-on-chip. In *the Euromicro Conference on Real-Time Systems*. 47–56.
- [10] P. Kollig, C. Osborne, and T. Henriksson. 2009. Heterogeneous multi-core platform for consumer multimedia applications. In *Conference on DATE*. 1254–1259.
- [11] K. Lakshmanan, D.-de Niz, and R. Rajkumar. 2009. Coordinated Task Scheduling, Allocation and Synchronization on Multiprocessors. In *IEEE Real-Time Systems Symposium*. 469–478.
- [12] D. Liu, J. Spasic, P. Wang, and T. Stefanov. 2016. Energy-Efficient Scheduling of Real-Time Tasks on Heterogeneous Multicores Using Task Splitting. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. 149–158.
- [13] J. M. López, J. L. Díaz, and D. F. García. 2004. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real-Time Systems* 28, 1 (2004), 39–68.
- [14] L. Sha, R. Rajkumar, and J. P. Lehoczky. 1990. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computer* 39, 9 (1990), 1175–1185.
- [15] Arm Techcon. 2011. Big.LITTLE processing with ARM Cortex-A15 and Cortex-A7. *Eetimes Com* (2011).
- [16] G. Tong and C. Liu. 2016. Supporting Soft Real-Time Sporadic Task Systems on Uniform Heterogeneous Multiprocessors with No Utilization Loss. *IEEE Trans. on Parallel Distrib. Syst.* 27, 9 (2016), 2740–2752.
- [17] T. H. Tsai, L. F. Fan, Y. S. Chen, and T. S. Yao. 2016. Triple Speed: Energy-Aware Real-Time Task Synchronization in Homogeneous Multi-Core Systems. *IEEE Trans. on Computers* 65, 4 (2016), 1297–1309.
- [18] C. Zoubek and P. Trommler. 2017. Overview of worst case execution time analysis in single- and multicore environments. In *International Conference on Architecture of Computing Systems*. 1–5.