CS-TR-2013-002, Department of Computer Science, UTSA, 2013

Multiprocessor Real-Time Systems with Shared Resources: Utilization Bound and Mapping

Jian-Jun Han, *Member, IEEE,* Dakai Zhu, *Member, IEEE,* Xiaodong Wu, Laurence T. Yang, *Senior Member, IEEE* Hai Jin, *Senior Member, IEEE,*

Abstract—In real-time systems, both scheduling theory and resource access protocols have been studied extensively. However, there is very limited research focusing on scheduling algorithms specifically for real-time systems with shared resources, where the problem becomes more prominent with the emergence of multicore processors. In this paper, focusing on the partitioned-EDF scheduling with the MSRP resource access protocol, we study the utilization bound and efficient task mapping schemes for a set of periodic real-time tasks running on a multicore/multiprocessor system with shared resources. Specifically, we first illustrate the *scheduling anomaly* where the tasks are schedulable when being mapped on to fewer but not more processors due to synchronization overhead. Then, with such synchronization overhead being considered, we develop a *synchronization-cognizant utilization bound*. Moreover, we show that finding the optimal mapping of tasks with shared resources is NP-hard and propose two efficient *synchronization-cognizant task mapping algorithms (SC-TMA)* that rely on the new tightened synchronization overhead and have the goal of achieving better schedulability and balanced workload on deployed processors. Finally, the proposed SC-TMA schemes are evaluated through extensive simulations with synthetic tasks. The results show that, the schedulability ratio and (average) system load under SC-TMA are close to that of an INLP (Integer Non-Linear Programming) based solution for small task systems. When compared to the existing task mapping algorithms, SC-TMA obtain much better schedulability ratio and lower/balanced workload on all processors.

Index Terms—Real-time systems; Multiprocessor; Periodic tasks; Shared resources; Partitioned scheduling; Utilization bound;

1 Introduction

The scheduling theory for real-time systems has been studied for decades and many scheduling algorithms have been developed. Although the uniprocessor scheduling theory has been comprehensively studied where the *earliest deadline first (EDF)* and *rate monotonic scheduling (RMS)* are the well-known optimal scheduling algorithms [30], the scheduling of real-time tasks in multiprocessor systems is still an evolving research field and many problems remain open due to their intrinsic difficulties [15], [40]. Moreover, with the emergence of multicore processors, there is a reviving interest in the multiprocessor real-time scheduling problem and many results have been reported recently [8], [14], [22], [23], [29].

There have been two traditional approaches to the multiprocessor scheduling problem: *partitioned* and *global* scheduling [16], [17]. In partitioned scheduling, tasks are statically assigned to processors and a task can only run on its designated processor where the well-established uniprocessor scheduling algorithms (e.g., EDF and RMS) can be employed on each processor. In comparison, all tasks in global scheduling are put into a shared queue and every idle processor fetches the next highest-priority ready task from the global queue for execution. More recently, as a general and hierarchical approach, *cluster scheduling* has been investigated, where tasks are first partitioned among several clusters of processors and then scheduled with different global scheduling policies (e.g., Global-EDF) within each cluster [7], [37], [42].

To efficiently determine whether a given task set is schedulable, several utilization bounds have been developed for partitioned scheduling based on the simple task mapping heuristics (such as first-fit, best-fit and worst-fit) when both EDF [32] and RMS [31], [35] are considered on each processor. Similarly, by limiting the maximum task utilization, such utilization bounds have also been studied for global-EDF and global-RMS scheduling algorithms [2], [6], [21]. Note that, these utilization bounds can only be applied to task systems that do *not* have shared resources.

In some real-time systems, due to resource limitation and/or synchronization requirements, tasks may need to exclusively access shared resources (such as shared data objects or I/O channels). Such resource access contention can lead to significantly degraded schedulability due to *priority inversion*, where a lower-priority task accesses a shared resource non-preemptively and thus blocks the execution of a higher-priority task [38]. For instance, a recent study pointed out that an application could take up to 30% more time due to access contention for shared data resources on a 16-processor system [10]. As multicore processors, where multiple processing cores are integrated on a single chip with shared last level cache and I/O channels [36], emerge to be the computing engine for modern real-time embedded systems [12], [33], such resource access contention problems will become more prominent and demand for efficient scheduling solutions.

To tackle the priority inversion problem, several lock-based resource access protocols have been investigated, such as Priority Ceiling Protocol (PCP) [41] and Stack Resource Policy (SRP) [5] for uniprocessor systems and the corresponding extensions MPCP [28] and MSRP [20] for multiprocessor

• *J.-J. Han, X. Wu, L. T. Yang and H. Jin are with School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China.*
• *D. Zhu is with Department of Computer Science, The University of Texas at San Antonio, San Antonio, TX 78249, USA.*

systems. These protocols have been exploited to guarantee the timeliness of tasks when accessing shared resources. However, most existing work focused on either scheduling algorithms or resource access protocols. There is only limited research on scheduling algorithms for tasks with shared resources that take resource access protocols into consideration to improve task schedulability [24], [25], [34].

In [25], focusing on fixed priority (i.e., RMS) scheduling, Hsiu *et al.* developed a dedicated-core framework where several service tasks running on a few dedicated cores control the access right for shared resources in critical sections using a RPC-like mechanism and application tasks are mapped to processors with simple heuristics (e.g., *First-Fit*). Considering the RMS scheduling and MPCP, Nemati *et al.* studied a *best-fit decreasing (BFD)* task mapping heuristic based on macrotasks (which contain tasks that directly or indirectly access the same resources) [34]. The scheme tries to map tasks in a macrotask to a processor to reduce synchronization overhead. Moreover, focusing on partitioned-EDF, Han *et al.* proposed a *synchronization-aware worst-fit decreasing (SA-WFD)* task mapping scheme very recently that tries to allocate tasks accessing a similar set of resources to a processor for better schedulability [24].

However, the existing work either focused on simple heuristics that does not directly take synchronization overhead into consideration [25] or adopted very pessimistic estimation for such overhead [24], [34] when making scheduling/mapping decisions. Moreover, to the best of our knowledge, there is no existing work that studied the utilization bound for real-time tasks that access shared resources in multiprocessor systems. In this paper, focusing on partitioned-EDF and MSRP, we study the utilization bound and efficient task mapping schemes for real-time tasks running on a multiprocessor system with shared resources under the MSRP resource access protocol. The **contributions** of this work can be summarized as follows.

- We discover and formally show the **scheduling anomaly** problem where a set of tasks are schedulable when being mapped on to fewer but not more processors due to synchronization overhead for accessing shared resources;
- We develop the *first* **synchronization-cognizant utilization bound** for partitioned-EDF scheduling that takes the synchronization overhead of tasks accessing shared resources into consideration;
- We show that the mapping problem of tasks with shared resources is NP-hard and propose two **synchronization-cognizant task mapping** heuristics aiming at reducing synchronization overhead and obtaining better schedulability and balanced workload on deployed processors.

The remainder of this paper is organized as follows. Section 2 reviews closely-related research. Section 3 presents system models and some preliminaries. The utilization bound is developed in Section 4. Section 5 explores a new method to tighten overhead and Section 6 presents the synchronization-cognizant task mapping algorithms. Simulation results are discussed in Section 7 and Section 8 concludes the paper.

2 Related Work

For a set of periodic real-time tasks running on uniprocessor systems, the optimal static/fixed priority based RMS scheduling algorithm has a utilization bound of $N(2^{\frac{1}{N}}-1)$, where N is the number of tasks under consideration, and the dynamic priority based EDF scheduler can achieve 100% system utilization [30]. However, since a real-time task cannot occupy more than one processor at any time, the global-scheduling based EDF and RMS algorithms could fail to schedule task sets with very low system utilization in multiprocessor systems [17].

When the maximum task utilization for a task set is limited and no more than α (≤ 1), it has been shown that a task set is schedulable under global-EDF if the system utilization does not exceed $M(1-\alpha)+\alpha$, where M is the number of processors in the system [21]. Similarly, for global-RMS scheduling, a system utilization of $(M/2)(1-\alpha)+\alpha$ can be guaranteed [6]. Andersson *et al.* also studied the scheduling algorithm RM-US, where tasks with utilization higher than some threshold θ have the highest priority [2]. When $\theta=\frac{1}{3}$, Baker showed that RM-US can guarantee a system utilization of $(M+1)/3$ [6].

With the goal of achieving 100% system utilization, several optimal global-scheduling based algorithms have been studied. For instance, the well-known *proportional fair (Pfair)* scheduler enforces proportional progress (i.e., *fairness*) for each task at every time quantum [9]. Later, observing that a periodic real-time task can only miss its deadline at its period boundary, the *boundary-fair (Bfair)* scheduling algorithm was studied that makes scheduling decisions and ensures fairness for tasks *only* at the period boundaries to reduce scheduling overhead [43]. For continuous-time based systems, the *T-L plane* based scheduling algorithms were studied in [13], [19] and a generalized deadline-partitioned fair (DP-Fair) scheduling model was investigated in [29]. More recently, an optimal approach RUN was studied that reduces the problem to be a uniprocessor scheduling problem [39]. Although these optimal global schedulers can achieve full system utilization, all of them could incur quite high scheduling overhead due to excessive number of scheduling points and context switches.

For partitioned scheduling, Oh and Baker studied the rate-monotonic first-fit (RMFF) heuristic and showed the utilization bound for RMFF on a M-processor system is $M(2^{1/2}-1)$ [35]. Later, a better bound of $(M+1)(2^{1/(M+1)}-1)$ for RMFF was derived in [31]. In [3], Andersson and Jonsson proved that the system utilization bound can reach 50% for a partitioned RM scheduling by exploiting the harmonicity of tasks' periods. For the partitioned-EDF with reasonable mapping heuristics, Lopez *et al.* showed that any task set can be successfully scheduled if the total utilization is no more than $(\beta\cdot M+1)/(\beta+1)$, where $\beta=\lfloor 1/\alpha\rfloor$ and α is the maximum task utilization [32]. Andersson *et al.* proposed the EKG algorithm based on the concept of *portion tasks* [4], which can trade higher scheduling overhead for better system utilization (within the range of 66% to 100%). Following the same line of research, several semi-partitioning based scheduling algorithms have been proposed that have different handling mechanisms for portion tasks and thus achieve different system utilization bounds [23], [26], [27].

Note that, all the above studies are for real-time tasks that do not have shared resources. As mentioned earlier, several resource access protocols have been studied to control the access of shared resources among tasks and thus guarantee their timeliness. For example, the lock-based Priority Ceiling Protocol (PCP) [41] and Stack Resource Policy (SRP) [5] for uniprocessor systems. Such protocols have been extended for multiprocessor systems to tackle the blocking caused by tasks on different processors that access the same shared resources, such as MPCP (Multiprocessor PCP) [28] and MSRP (Multiprocessor SRP) [20]. Moreover, in [18], Easwaran *et al.* studied a parallel PCP (P-PCP) resource-sharing protocol for fixed-priority tasks in multiprocessor systems and developed the corresponding schedulability conditions. Recently, Block *et al.* introduced a Flexible Multiprocessor Locking Protocol (FMLP) [10] and Brandenburg *et al.* studied an asymptotically optimal locking protocol (OMLP) [11].

To improve the schedulability of real-time tasks that access shared resources in multiprocessor systems, a few research studies have focused on the partitioning heuristics with the resource access protocols being considered. In [25], Hsiu *et al.* developed a dedicated-core framework that delegates the control of resource access to some fixed cores in a multicore real-time system, where service tasks govern the resource access in critical sections using RPC like mechanism. The partitioning of application tasks to cores are transformed to three core-minimization problems (i.e., minimizing the number of the service cores, application cores and total cores) under the time and synchronization constraints. They employed several simple heuristics (e.g., *First-Fit*) for task assignments and tasks on each core are scheduled with RMS [25].

The most related previous studies were recently reported in [24], [34]. Focusing on RMS scheduling on each processor and MPCP, Nemati *et al.* studied a cluster-based approach to allocate tasks that share resources in a multicore system based on best-fit decreasing (BFD) heuristic [34]. They tried to fit a macrotask (which consists of tasks that directly or indirectly access the same resources) in a core. Once a macrotask cannot fit into a core, it is broken and its component tasks will be assigned together with other tasks using the pre-defined weight and attraction functions that measure the degree of resource access contention subject to MPCP. In [24], based on EDF and MSRP, Han *et al.* proposed a *synchronization-aware worst-fit decreasing (SA-WFD)* task partitioning heuristic that tries to allocate tasks accessing a similar set of resources to a single core for better schedulability.

The work reported in this paper is different from the existing work, which either did not directly take the synchronization overhead into consideration [25] or have adopted simple synchronization characteristics (such as resource similarity) and very pessimistic estimation of such overhead when making scheduling/mapping decisions [24], [34]. Observing that the synchronization overhead has a great impact on task schedulability, we exploit an iterative approach to tighten such overhead during the task mapping process, which can significantly improve task schedulability as shown in the evaluation results. Moreover, we developed the first utilization bound for real-time tasks in multiprocessor systems with shared resources.

3 Preliminaries

In this section, we first layout the scope of this study by presenting the system, task and resource models and stating our assumptions. Moreover, the MSRP resource access protocol and the schedulability condition for partitioned-EDF [5], [20] are briefly reviewed, followed by the description of the problem to be addressed in this paper.

3.1 System Models

We consider a homogeneous multiprocessor real-time system that consists of M processors ($\{\mathcal{P}_1, \dots, \mathcal{P}_M\}$) with identical functions and capabilities. The system has a set of R global resources $\Re = \{\mathcal{R}_1, \dots, \mathcal{R}_R\}$, which can be shared by all tasks. A set of N periodic real-time tasks $\Psi = \{\tau_1, \dots, \tau_N\}$ will be executed on the system and each task τ_i has a period p_i, which is also its relative deadline. That is, the j^{th} task instance (or job) of task τ_i arrives at time $(j-1) \cdot p_i$ and has to complete its execution by its absolute deadline $j \cdot p_i$. Note that, there is at most one active job for a task at any time. Thus, without causing ambiguity, we use *task* and *job* exchangeably for the remainder of this paper.

A resource can be accessed by only one task at any time within its critical sections. That is, the access to any resource by a task is *exclusive* and *non-preemptable*. Moreover, we assume that there is no nested critical section as it occurs infrequently in practice and can be tackled by group locks [10]. Therefore, a task is not allowed to request for more than one resource (and thus can only access one resource) at any time.

We assume that there are n_i sections in task τ_i and the j^{th} section ($1 \leq j \leq n_i$) is denoted as $s_{i,j}$. The worst case execution time (WCET) of section $s_{i,j}$ is $c_{i,j}$ and τ_i's WCET is given as $c_i = \sum_{j=1}^{n_i} c_{i,j}$. The utilization of task τ_i is defined as $u_i = \frac{c_i}{p_i}$. The system utilization is given as $U = \sum_{i=1}^{N} u_i$.

Moreover, to precisely model the critical sections of task τ_i, we use a flag $r_{i,j}$ for each section $s_{i,j}$ of τ_i to indicate whether $s_{i,j}$ is a critical section or not. If $s_{i,j}$ is a non-critical section, $r_{i,j} = 0$; otherwise, $r_{i,j}$ denotes the identification of the resource that τ_i accesses during its critical section $s_{i,j}$. Therefore, we have $0 \leq r_{i,j} \leq R$. The subset of resources that are accessed by task τ_i during its execution is denoted as $\Re_i$ ($\subseteq \Re$). Note that, a task may need to access a resource multiple times within its different critical sections.

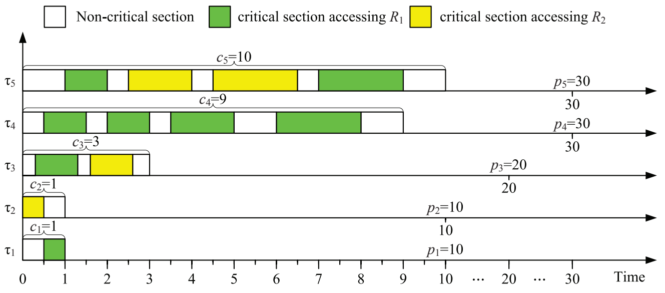


Fig. 1: An example of tasks and resource access patterns.

As an example, consider a system with five tasks $\Psi = \{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$ and two resources $\Re = \{\mathcal{R}_1, \mathcal{R}_2\}$. The sections of tasks and their resource access patterns are shown in Figure 1. Here, we have $n_1 = n_2 = 2$, $n_3 = 5$ and $n_4 = n_5 = 9$. Use task τ_3 as the example, there are $r_{3,1} = r_{3,3} = r_{3,5} = 0$, $r_{3,2} = 1$ and $r_{3,4} = 2$. Therefore, we have $\Re_1 = \Re_4 = \{\mathcal{R}_1\}$, $\Re_2 = \{\mathcal{R}_2\}$ and $\Re_3 = \Re_5 = \Re$.

3.2 Schedulability: Partitioned-EDF and MSRP

When shared resources are considered, the execution of a task, regardless of its priority, can be blocked when it attempts to access a resource that is currently held (and accessed) by another task. Therefore, the exact sequence for tasks to access the resources and the schedulability of tasks rely on not only the scheduling algorithm but also the resource access protocol. In [10], it has been shown that, when there are only global shared resources and non-nested critical sections within tasks (as the case being considered in this work), the global and partitioned scheduling have comparable schedulability.

However, considering its relatively simple per-processor based schedulability analysis [20], we focus on partitioned-EDF in this paper. Moreover, for the resource access protocol, although the suspension-based mechanism adopted in OMLP can improve system efficiency at runtime [11], it cannot improve the off-line schedulability analysis that needs to consider the worst case scenario. Therefore, we focus on the spin-lock based MSRP resource access protocol, where the basic rules can be summarized as follows [20]:

- **Rule 1:** When a task τ_i attempts to access a resource $\mathcal{R}_a$, if $\mathcal{R}_a$ is free (i.e., no other task is accessing it), it will lock and access the resource by executing its critical section non-preemptively; Otherwise, if $\mathcal{R}_a$ is currently held by another task (on a different processor), task τ_i will be added to $\mathcal{R}_a$'s FIFO queue and the processor enters a *non-preemptive* busy-wait state;
- **Rule 2:** Once a task τ_i finishes accessing a resource $\mathcal{R}_a$ at the end of a critical section, it will release $\mathcal{R}_a$ and become preemptable again. If $\mathcal{R}_a$'s FIFO queue is not empty (i.e., there are tasks from other processors waiting for accessing $\mathcal{R}_a$), the queue's header task will start accessing the resource; otherwise, $\mathcal{R}_a$ is unlocked.

From the above rules, we can see that the execution of a task τ_i on processor $\mathcal{P}_k$ can be blocked due to synchronization requirements at two different occasions:

- First, when a low priority task τ_j (which has a deadline later than that of task τ_i) is accessing or busy-waiting for a resource on the same processor $\mathcal{P}_k$, task τ_i is blocked and the duration is denoted as *local blocking time*;
- Second, when τ_i tries to access a resource $\mathcal{R}_a$ that is currently held and accessed by a task on another processor, it has to wait in $\mathcal{R}_a$'s FIFO queue and the duration is denoted as *global waiting time*.

Moreover, for tasks accessing shared resources under MSRP, we can have the following properties [20]:

Property 1. *For any processor at any given time, there exists at most one task that is either a) accessing a resource; or b) busy-waiting for a resource (which is currently held by a task on another processor).*

Property 2. *A task can be blocked by a low priority task on the same processor at most once. Moreover, a task's local blocking time is upper bounded by the* longest duration *of any low priority task on the same processor accessing (and waiting, if applicable) one of its resources* once.

Schedulability Condition: For a given partitioning (or mapping) of tasks to processors $\Pi = \{\Psi_1, \cdots, \Psi_M\}$, where Ψ_k is the subset of tasks that are allocated to processor $\mathcal{P}_k$, we first review the schedulability condition for partitioned-EDF under MSRP [5], [20]. For the ease of presentation and discussion, some necessary notations are defined as follows:

- $BW_{i,x}$: the maximum *global waiting time* that task τ_i can experience when it waits for accessing resource $\mathcal{R}_a$ in its critical section $s_{i,x}$ (where $a = r_{i,x}$) on $\mathcal{P}_k$;
- BW_i: the maximum *total global waiting time* for task τ_i to access its resources in all its critical sections;
- B_i: the maximum *local blocking time* that can be experienced by task τ_i on processor $\mathcal{P}_k$. From Property 2, we know that task τ_i can only be blocked at most once by another task τ_j on the same processor when τ_j (waits and) accesses a resource in one of its critical sections, where $p_j > p_i$ [5], [20]. Therefore, we have

$$B_i = \max\{BW_{j,y} + c_{j,y} | \forall s_{j,y} : \tau_j \in \Psi_k \wedge p_j > p_i \wedge r_{j,y} \neq 0\} \quad (1)$$

Taking the local blocking and global waiting times of tasks into consideration, for a given task-to-processor mapping Π, the *synchronization-cognizant processor load* for $\mathcal{P}_k$ can be defined as:

$$L_k^{sc}(\Pi) = \max\left\{\frac{B_i}{p_i} + \sum_{\forall \tau_j \in \Psi_k}^{p_j \leq p_i} \frac{c_j + BW_j}{p_j} | \forall \tau_i \in \Psi_k\right\} \quad (2)$$

Correspondingly, we define the *synchronization-cognizant system load* as the maximum of all processors' synchronization-cognizant loads as given below:

$$L^{sc}(\Pi) = \max\{L_k^{sc}(\Pi) | \forall \mathcal{P}_k : 1 \leq k \leq M\} \quad (3)$$

With these definitions, we can directly get the following proposition regarding to the feasibility of a given task-to-processor mapping based on the results from [5], [20]:

Proposition 1. *For a set of periodic real-time tasks running on a multiprocessor system with shared resources that are governed by MSRP, a given task-to-processor mapping* Π *is feasible for partitioned-EDF if there is* $L^{sc}(\Pi) \leq 1$.

Global Waiting Time: For a given task-to-processor mapping Π, the global waiting time $BW_{i,x}$ of task τ_i for accessing $\mathcal{R}_a$ in its critical section $s_{i,x}$ can be calculated as [20]:

$$BW_{i,x} = \sum_{j=1,\cdots,M}^{j \neq k} tp_j^{max}(\mathcal{R}_a) \quad (4)$$

where $tp_j^{max}(\mathcal{R}_a)$ is the maximum amount of time for any task on other processor $\mathcal{P}_j$ ($j \neq k$) to access resource $\mathcal{R}_a$ *once*. That is, in the worst case, task τ_i may have to wait for the longest access time of $\mathcal{R}_a$ by tasks on all other processors.

Here, if $s_{i,x}$ is a non-critical section and $r_{i,x} = 0$, $BW_{i,x} = 0$. Moreover, $tp_j^{max}(\mathcal{R}_a)$ can be further calculated as [20]:

$$\begin{aligned} tp_j^{max}(\mathcal{R}_a) &= \max\{tt_i^{max}(\mathcal{R}_a) | \forall \tau_i \in \Psi_j\} \\ tt_i^{max}(\mathcal{R}_a) &= \max\{c_{i,y} | \forall s_{i,y} : r_{i,y} = a\} \end{aligned}$$

where $tt_i^{max}(\mathcal{R}_a)$ denotes the maximum amount of time for task τ_i to access resource $\mathcal{R}_a$ *once*; if τ_i does not access $\mathcal{R}_a$ (i.e., $\mathcal{R}_a \notin \Re_i$), there is $tt_i^{max}(\mathcal{R}_a) = 0$.

Then, based on $BW_{i,x}$, the total global waiting time BW_i for task τ_i ($\in \Psi_k$) can be simply accumulated as [20]:

$$BW_i = \sum_{x=1}^{n_i} BW_{i,x} \qquad (5)$$

3.3 Problem Description

Based on Proposition 1, the problem to be addressed in this paper is: *for a set of periodic real-time tasks running on a homogeneous multiprocessor system with shared resources under partitioned-EDF and MSRP, finding a feasible task-to-processor mapping* Π *such that* $L^{sc}(\Pi)$ *is minimized.*

Note that, when the system has no shared resource, the special case of the problem becomes the traditional partitioned real-time scheduling problem. In [16], [17], it has been shown that finding the optimal task-to-processor mapping for such a problem to minimize the maximum system load is NP-hard. Therefore, the mapping problem to be studied in this paper is NP-hard as well. Hence, in what follows, we will first develop the utilization bound and then focus on efficient task mapping heuristics that *explicitly* take synchronization overhead of tasks accessing shared resources into consideration.

4 Utilization Bound and Anomaly

For real-time systems without shared resources, López *et al.* have investigated the utilization bounds for partitioned-EDF with various mapping heuristics, such as First-Fit (FF), Best-Fit (BF) and Worst-Fit Decreasing (WFD) [32]. As long as the system utilization of a task set is no more than such bounds, the result task-to-processor mappings under these heuristics are guaranteed to be feasible, which can be exploited for efficient schedulability test. Note that, such bounds increase *monotonically* when there are more available processors [32].

However, for systems with shared resources, from Proposition 1 and Equation (3), we can see that the feasibility of a given task-to-processor mapping depends on not only the accumulated task utilizations on every processor but also the local blocking and global waiting time (i.e., *synchronization overhead*) of tasks, which can become larger when tasks are mapped to more processors. Before developing the utilization bound for systems with shared resources, we first illustrate the scheduling *anomaly* through a concrete example.

4.1 Scheduling Anomaly: An Example

Consider a task system with three tasks $\Psi = \{\tau_1, \tau_2, \tau_3\}$ and two resources $\Re = \{\mathcal{R}_1, \mathcal{R}_2\}$. For task τ_1, it accesses the resource $\mathcal{R}_1$ twice within its two critical sections that have the same size of 1. Moreover, there are $c_1 = 4$ and $p_1 = 10$. Similarly, task τ_2 accesses $\mathcal{R}_1$ once within its critical section that has the size of 4 with $c_2 = 5$ and $p_2 = 9$; task τ_3 accesses resource $\mathcal{R}_2$ once within its critical section that has the size of 2 with $c_3 = 8$ and $p_3 = 10$.

If only two processors are deployed, we can allocate the first two tasks on processor $\mathcal{P}_1$ and the third task on processor $\mathcal{P}_2$. That is, we have $\Psi_1 = \{\tau_1, \tau_2\}$ and $\Psi_2 = \{\tau_3\}$ for the mapping. From Equations (1), (4) and (5), we can get $BW_1 = BW_2 = BW_3 = 0$, $B_1 = 0$, $B_2 = 1$ and $B_3 = 0$. Moreover, from Equation (2), we can get $L_1^{sc} = \max\{\frac{1}{9}+\frac{5}{9}, \frac{5}{9}+\frac{4}{10}\} = 0.96$ and $L_2^{sc} = \max\{\frac{8}{10}\} = 0.8$. Thus, there is $L^{sc} = \max\{L_1^{sc}, L_2^{sc}\} = 0.96 < 1$. Therefore, the mapping is feasible under partitioned-EDF and MSRP according to Proposition 1.

However, when three processors are deployed, there is only one possible mapping of tasks to processors, where each processor is assigned one task. Here, we assume that **any deployed processor will be mapped at least one task** (i.e., $\Psi_k \neq \emptyset$). Without loss of generality, suppose that the mapping is $\Psi_k = \{\tau_k\}$ ($k = 1, 2, 3$). We can get $BW_1 = 8$, $BW_2 = 1$, $BW_3 = 0$ and $B_1 = B_2 = B_3 = 0$. Furthermore, we can find out that $L_1^{sc} = \max\{\frac{4+8}{10}\} = 1.2$. That is, the mapping fails to meet the schedulability condition for partitioned-EDF and MSRP according to Proposition 1. Here, the problem comes from the increased synchronization overhead (i.e., the global waiting time) when more processors are deployed. Hence, we can have the following observation.

Observation 1. *For a set of real-time tasks that access shared resources and are scheduleable under the partitioned-EDF and MSRP on a given number of processors, when more processors are deployed, the synchronization between the tasks can become more complex and the task set could become* unschedulable *due to increased synchronization overhead.*

4.2 Synchronization-Cognizant Utilization Bound

Next, focusing on the WFD mapping heuristic, we develop the utilization bound for multiprocessor real-time systems with shared resources. From previous discussion, the synchronization overhead of tasks rather relies on a specific task-to-processor mapping. Therefore, to find the upper-bounds for such synchronization overhead and for the ease of presentation, we define the following notations:

- p^{min}: the minimum period of tasks under consideration;
- $n^{max,cs}$: the maximum number of critical sections in a task for the task system under consideration;
- $c^{max,cs}$: the size of the largest critical section for all tasks in the task system under consideration;
- BW^{ub}: the upper-bound of the overall global waiting time that can be experienced by any task to access all its resources. From MSRP and Property 1, we can safely have $BW^{ub} = n^{max,cs} \cdot ((M-1) \cdot c^{max,cs})$;
- B^{ub}: the upper-bound of the local blocking time that can be experienced by any task; similarly, from Property 2, we can safely have $B^{ub} = c^{max,cs} + (M-1) \cdot c^{max,cs} = M \cdot c^{max,cs}$;
- α: the maximum synchronization-cognizant utilization of tasks, which is defined as $\alpha = \max\{\frac{c_i + BW^{ub}}{p_i} | \forall \tau_i \in \Psi\}$;

- γ: the upper-bound for *utilization loss* on any processor due to local blocking time, which is defined as $\gamma = \frac{B^{ub}}{p^{min}}$;
- β: the minimum number of tasks that can feasibly fit into one processor under EDF when synchronization overhead is considered. From Proposition 1 and Equations (2) and (3), we have $\beta = \lfloor \frac{1-\gamma}{\alpha} \rfloor$.

From the definition of β, we can easily obtain the following lemma regarding to the number of tasks and system schedulability with the similar reasonings as in [32]:

Lemma 1. *For a set of N periodic real-time tasks running on a M-processor system with shared resources, the task set is schedulable under partitioned-EDF with MSRP if $N \leq \beta \cdot M$.*

In what follows, we focus on the cases with $N > \beta \cdot M$. Note that, with the definition of B^{ub} and BW^{ub}, for any given task-to-processor mapping $\Pi = \{\Psi_1, \cdots, \Psi_M\}$, the following equation holds for every processor $\mathcal{P}_k$ $(k = 1, \cdots, M)$:

$$L_k^{sc}(\Pi) \leq \max \left\{ \frac{B^{ub}}{p^{min}} + \sum_{\forall \tau_j \in \Psi_k}^{p_j \leq p_i} \frac{c_j + BW^{ub}}{p_j} | \forall \tau_i \in \Psi_k \right\}$$

Hence, from Proposition 1, we can get the following lemma.

Lemma 2. *For a set Ψ of periodic real-time tasks running on a M-processor system with shared resources that are scheduled under the partitioned-EDF with MSRP, a given task-to-processor mapping $\Pi = \{\Psi_1, \cdots, \Psi_M\}$ is feasible, if for every processor $\mathcal{P}_k$ $(k = 1, \cdots, M)$, there is:*

$$\max \left\{ \frac{B^{ub}}{p^{min}} + \sum_{\forall \tau_j \in \Psi_k}^{p_j \leq p_i} \frac{c_j + BW^{ub}}{p_j} | \forall \tau_i \in \Psi_k \right\} \leq 1 \quad (6)$$

Note that, compared to Equation (2), Equation (6) represents a much relaxed sufficient condition regarding to tasks' schedulability. In what follows, we derive the *synchronization-cognizant utilization bound ($U^{sc,bound}$)* for systems with shared resources based on Equation (6). That is, as long as the system utilization of a task set is no more than $U^{sc,bound}$, the result task-to-processor mapping under WFD guarantees that Equation (6) will hold, which further implies that the task set is schedulable under partitioned-EDF and MSRP. Define $\sigma = \max\{\gamma, \frac{BW^{ub}}{p^{min}}\}$ as the *synchronization overhead factor*.

Theorem 1. *For a set of N periodic real-time tasks running on a M-processor system with shared resources where the number of tasks $N > \beta \cdot M$, the synchronization-cognizant utilization bound ($U^{sc,bound}$) for partitioned-EDF and MSRP under the WFD mapping heuristic can be found as:*

$$U^{sc,bound} = \min\{U^{b1}, U^{b2}\} \quad (7)$$

where

$$U^{b1} = \frac{\beta \cdot M + 1}{1 + \beta} \cdot (1 - \sigma) - (\beta \cdot M + 1) \cdot \sigma \quad (8)$$

$$U^{b2} = \frac{M \cdot N}{M + N - 1} \cdot (1 - \sigma) - N \cdot \sigma \quad (9)$$

Proof: With the focus on WFD, we assume that tasks are sorted in non-increasing order of their utilizations. That is, for tasks τ_i and τ_j where $1 \leq i < j \leq N$, there is $u_i \geq u_j$.

Suppose that task τ_n is the first task that would *fail* the sufficient condition on every processor represented by Equation (6) should it be allocated to that processor. That is, for every processor $\mathcal{P}_k$ $(k = 1, \cdots, M)$, there is:

$$\frac{c_n + BW^{ub}}{p_n} + \frac{B^{ub}}{p^{min}} + \sum_{\forall \tau_j \in \Psi_k} \frac{c_j + BW^{ub}}{p_j} > 1$$

where Ψ_k contains the subset of tasks on processor $\mathcal{P}_k$ after allocating the first $(n-1)$ tasks. Note that $p^{min} \leq p_i$ $(i = 1, \cdots, n)$. From the definition of σ, the above inequality can be transformed as follows on every processor $\mathcal{P}_k$:

$$u_n + 2\sigma + \sum_{\forall \tau_j \in \Psi_k} u_j + |\Psi_k| \cdot \sigma > 1$$

where $|\Psi_k|$ represents the number of tasks in the subset Ψ_k. Adding all these M inequalities together, we can get:

$$(M-1) \cdot u_n + \sum_{j=1}^{n} u_j + (2M + n - 1) \cdot \sigma > M$$

From the assumption that tasks are ordered in non-increasing order of their utilizations, there is $u_n \leq \frac{\sum_{j=1}^{n} u_j}{n}$. Thus, the above inequality can be further transformed as:

$$\left(\frac{M-1}{n} + 1 \right) \sum_{j=1}^{n} u_j + (2M + n - 1) \cdot \sigma > M$$

Hence, we have:

$$\sum_{j=1}^{n} u_j > \frac{n}{M + n - 1} \cdot (M - (2M + n - 1) \cdot \sigma)$$

Therefore, considering that the system utilization of the whole task set $U \geq \sum_{j=1}^{n} u_j$, we can further have:

$$\begin{aligned} U &> \frac{n}{M + n - 1} \cdot (M - (2M + n - 1) \cdot \sigma) \\ &= \frac{M \cdot n}{M + n - 1}(1 - \sigma) - n \cdot \sigma = f(n) \quad (10) \end{aligned}$$

where $f(n)$ is a function of n. Note that $\beta \cdot M < n \leq N$. To obtain a positive (and meaningful) utilization bound, we need to have $f(n) > 0$. That is,

$$\frac{1 - \sigma}{\sigma} > \frac{M + n - 1}{M} \geq \frac{M + (\beta \cdot M + 1) - 1}{M} = 1 + \beta$$

Therefore, we need to have $\sigma < \frac{1}{\beta + 2} < 1$.

Since σ is not related to n, we can get the second derivative of $f(n)$ with respect to n as

$$f''(n) = -\frac{2M \cdot (M-1) \cdot (1-\sigma)}{(M + n - 1)^3} < 0$$

Therefore, the function $f(n)$ is a concave function with its minimum value can be found when either $n = \beta \cdot M + 1$ or $n = N$. Note that,

$$f(\beta \cdot M + 1) = \frac{\beta \cdot M + 1}{1 + \beta} \cdot (1 - \sigma) - (\beta \cdot M + 1) \cdot \sigma = U^{b1}$$

$$f(N) = \frac{M \cdot N}{M + N - 1} \cdot (1 - \sigma) - N \cdot \sigma = U^{b2}$$

Hence, the utilization bound $U^{sc,bound}$ will be the minimum value of $f(n)$ and $U^{sc,bound} = \min\{U^{b1}, U^{b2}\}$. Therefore, if the system utilization of a task set is no more than $U^{sc,bound}$, WFD guarantees to generate a task-to-processor mapping satisfying Equation (6), which concludes the proof.

□

As we mentioned earlier, by exploiting the upper-bounds of synchronization overhead, the simplified schedulability condition given in Equation (6) is very loose. Therefore, the synchronization-cognizant utilization bound derived from such sufficient condition as shown in Equation (7) is rather pessimistic. Specifically, for systems with large synchronization overhead (such as B^{ub}, BW^{ub} and σ), the value of $U^{sc,bound}$ can be very small, which limits its applicability.

On the other hand, when no task needs to access any resource (or there is no shared resource in the system), we will have $\sigma = 0$. For the function $f(n)$ defined in Equation (10), we can get its first derivative as $f'(n) = \frac{M\cdot(M-1)}{(M+n-1)^2} > 0$. Therefore, the minimum value of $f(n)$ can be found as $\frac{\beta\cdot M+1}{1+\beta}$ when $n = \beta\cdot M+1$, which actually reduces to be the utilization bound for systems without shared resources under WFD [32].

4.3 Non-Monotonicity of the Bound $U^{sc,bound}$

Different from the utilization bounds for systems without shared resources (which increase *monotonically* when the number of available processors increases [32]), the utilization bound $U^{sc,bound}$ for systems with shared resources can become lower when more processors are deployed. From Equations (7) to (9), we can see that, in addition to the number of deployed processors, $U^{sc,bound}$ depends heavily on the synchronization overhead of tasks. However, for a given set of real-time tasks, the bounds for such overhead (i.e., B^{ub} and BW^{ub}) actually become larger as the number of deployed processors increases. In what follows, we formally analyze such *non-monotonicity* of the utilization bound $U^{sc,bound}$.

Note that, from Equation (8), $U^{sc,bound}$ also depends on β, the minimum number of tasks that can feasibly fit into one processor under EDF when synchronization overhead is considered. Recall that $\beta = \lfloor\frac{1-\gamma}{\alpha}\rfloor$, where $\gamma = \frac{c^{max,cs}\cdot M}{p^{min}}$ and $\alpha = \max\{\frac{c_i+n^{max,cs}\cdot c^{max,cs}\cdot(M-1)}{p_i}|\forall\tau_i\in\Psi\}$. That is, β relies on M as well. However, due to the nature of the floor operation, we can have the following lemma regarding to the *invariance* property of β when M changes.

Lemma 3. *For a given set Ψ of real-time tasks that access shared resources, we have β to be a* constant *I (an integer) when $L(I) < M \leq H(I)$, where*

$$L(I) = \frac{p_k + (n^{max,cs}\cdot c^{max,cs} - c_k)\cdot(I+1)}{n^{max,cs}\cdot c^{max,cs}\cdot(I+1) + p_k\cdot c^{max,cs}/p^{min}} \quad (11)$$

$$H(I) = \frac{p_k + (n^{max,cs}\cdot c^{max,cs} - c_k)\cdot I}{n^{max,cs}\cdot c^{max,cs}\cdot I + p_k\cdot c^{max,cs}/p^{min}} \quad (12)$$

Here, task τ_k's maximum synchronization-cognizant utilization *is assumed to be* $\frac{c_k+n^{max,cs}\cdot c^{max,cs}\cdot(M-1)}{p_k} = \alpha$.

Proof: Note that, for a given integer I, from the definition of β, we have $\beta = I$ when $I \leq \frac{1-\gamma}{\alpha} < (I+1)$. Substitute α and γ in the above inequalities, we can get

$$I \leq \frac{1 - c^{max,cs}\cdot M/p^{min}}{(c_k + n^{max,cs}\cdot c^{max,cs}\cdot(M-1))/p_k} < I+1$$

With some transformations, we can obtain that, when $L(I) < M \leq H(I)$, there is $\beta = I$, which concludes the proof.

□

Therefore, from the above lemma, we can consider β to be a constant I when M changes within the range of $(L(I), H(I)]$, which further leads to the following theorem regarding to the non-monotonicity of the utilization bound $U^{sc,bound}$.

Theorem 2. *For a given set Ψ of real-time tasks that access shared resources, the synchronization-cognizant utilization bound under partitioned-EDF and MSRP as represented in Equation (7) can* decrease *as the number of deployed processors increases when there is $\frac{1}{3+\beta} < \sigma < \frac{1}{2+\beta}$.*

Proof: Define $\varsigma = \frac{n^{max,cs}\cdot c^{max,cs}}{p^{min}}$. Suppose that $BW^{ub} \geq B^{ub}$. We can have $\sigma = \frac{BW^{ub}}{p^{min}} = \frac{(M-1)\cdot n^{max,cs}\cdot c^{max,cs}}{p^{min}} = (M-1)\cdot\varsigma$. We further define $g(M) = U^{b1} = \frac{\beta\cdot M+1}{1+\beta}\cdot(1-\sigma) - (\beta\cdot M+1)\cdot\sigma$ and $h(M) = U^{b2} = \frac{M\cdot N}{M+N-1}\cdot(1-\sigma) - N\cdot\sigma$.

For $g(M)$, we can get its first derivative with respect to M as (note that β can be considered as a constant according to Lemma 3):

$$g'(M) = \beta\cdot\frac{1-\sigma}{1+\beta} - \varsigma\cdot\frac{\beta\cdot M+1}{1+\beta} - (\beta\cdot M+1)\cdot\varsigma - \beta\cdot\sigma$$

Substitute $\varsigma = \frac{\sigma}{M-1}$ into the above equation, we can get

$$g'(M) = \beta\cdot\frac{1-\sigma}{1+\beta} - \beta\cdot\sigma - \frac{\beta\cdot M+1}{M-1}\cdot\left(\frac{\sigma}{1+\beta} + \sigma\right)$$

Since $\frac{\beta\cdot M+1}{M-1} > \beta$, the above equation can be transformed as

$$g'(M) < \frac{\beta\cdot(1 - 4\sigma - 2\beta\cdot\sigma)}{1+\beta}$$

Here, we know that $g'(M) < 0$ when there is $\frac{1}{2\cdot(2+\beta)} < \sigma$. Moreover, from the proof of Theorem 1, we have $\sigma < \frac{1}{2+\beta}$. Therefore, when there is $\frac{1}{2\cdot(2+\beta)} < \sigma < \frac{1}{2+\beta}$, we can get that $g(M)$ decreases as M increases.

For $h(M)$, its first derivative with respect to M is:

$$h'(M) = N\cdot\left(\frac{(N-1)\cdot(1-\sigma)}{(M+N-1)^2} - \frac{2M+N-1}{M+N-1}\cdot\varsigma\right)$$

Substitute $\varsigma = \frac{\sigma}{M-1}$ into the above equation, we can have

$$\begin{aligned} h'(M) &= N\left(\frac{(N-1)\cdot(1-\sigma)}{(M+N-1)^2} - \frac{2M+N-1}{M+N-1}\cdot\frac{\sigma}{M-1}\right) \\ &< \frac{N}{M+N-1}\cdot\left((1-\sigma) - 2\cdot\sigma - \frac{(N-1)\sigma}{M-1}\right) \end{aligned}$$

Since $N \geq \beta\cdot M+1$, the above inequality can be further transformed as

$$h'(M) < \frac{N}{M+N-1}\cdot(1 - 3\cdot\sigma - \beta\cdot\sigma)$$

Therefore, we have $h'(M) < 0$ when there is $\frac{1}{3+\beta} < \sigma$. Note that there is $\frac{1}{2\cdot(2+\beta)} < \frac{1}{3+\beta}$. Hence, when there is $\frac{1}{3+\beta} <$

$\sigma < \frac{1}{2+\beta}$, it is possible to have both $g(M)$ and $h(M)$ (thus the utilization bound $U^{sc,bound}$) decrease as the number of deployed processors M increases.

For cases where $BW^{ub} < B^{ub}$, by re-defining $\varsigma = \frac{c^{max,cs}}{p^{min}}$, we have $\sigma = M \cdot \varsigma$. Following the similar steps, we can also get that both $g(M)$ and $h(M)$ decrease as the number of deployed processors increases when there is $\frac{1}{3+\beta} < \sigma < \frac{1}{2+\beta}$, which concludes the proof. □

5 Tightened Synchronization Overhead

Based on the upper-bounds of tasks' synchronization overhead, we developed the utilization bound $U^{sc,bound}$, which can be exploited for efficient schedulability test for the traditional WFD that relies on tasks' original utilizations. However, the pessimism nature of $U^{sc,bound}$ greatly limits its applicability. Moreover, without taking the synchronization overhead of tasks into consideration, it is very likely for the traditional WFD to fail to obtain feasible task-to-processor mappings for task sets with higher system utilizations [24].

On the other hand, the existing approach to calculating synchronization overhead is still *loose*, especially for the total global waiting time in Equation (5). Here, before a task τ_i accessing any resource for *every* of its critical sections, it *always* assumes the worst-case interference from tasks on other processors [20]. Such simplified calculation can result in unnecessarily larger values for tasks' total global waiting time, which in turn can falsely reject a feasible task-to-processor mapping based on Proposition 1.

5.1 Limits on Interference among Tasks

Recall that we consider synchronous periodic tasks where the first job of each task arrives at time 0. Therefore, from [11], we can directly get the following proposition regarding to the interference among tasks under partitioned-EDF scheduling.

Proposition 2. *For any two tasks τ_i and τ_j in a synchronous periodic task set scheduled under partitioned-EDF, the maximum number of τ_j's jobs that can interfere with the execution of any job of τ_i due to accessing shared resources is:*

$$\theta_{i,j} = \begin{cases} 1 & , \ p_i < p_j \wedge mod(p_j, p_i) = 0 \\ \frac{p_i}{p_j} & , \ p_i \geq p_j \wedge mod(p_i, p_j) = 0 \\ \left\lceil \frac{p_i}{p_j} \right\rceil + 1 & , \ otherwise \end{cases} \tag{13}$$

where $mod(x, y)$ returns the remainder of dividing x by y.

Note that a task may access one resource multiple times in its different critical sections. Define the set of task τ_i's critical sections where τ_i accesses resource $\mathcal{R}_a$ as $\mathcal{S}_{i,a} = \{s_{i,j} | r_{i,j} = a; j = 1, \cdots, n_i\}$. Then, from Proposition 2 and the MSRP protocol [20], we can obtain the proposition below.

Proposition 3. *For a given task-to-processor mapping Π, the number of interference (i.e., global waiting) that can be experienced by any job of task τ_i ($\in \Psi_k$) because of accessing resource $\mathcal{R}_a$ has the following limitations:*

- *$\theta_{i,j}$: caused by any task τ_j where $\tau_j \notin \Psi_k$;*
- *$|\mathcal{S}_{i,a}|$: caused by tasks on any processor $\mathcal{P}_m$ ($m \neq k$);*

where $|\mathcal{S}_{i,a}|$ denotes the number of critical sections in $\mathcal{S}_{i,a}$.

Algorithm 1 : Calculate waiting time $BW_i(\mathcal{R}_a)$

Input: Ψ, Π ($= \{\Psi_m\}$), $\tau_i \in \Psi_k$ and $\mathcal{R}_a$ ($\in \Re_i$);
Output: $BW_i(\mathcal{R}_a)$;

```
1: $BW_i(\mathcal{R}_a) = 0$; $limit[m] = |\mathcal{S}_{i,a}|$ $(m = 1, \cdots, M)$;
2: for ($s_{j,y} \in \mathcal{S}_a(\Psi)$ where $\tau_j \in \Psi_m \wedge m \neq k$) do
3:     $count = \min\{limit[m], \theta_{i,j}\}$;
4:     $BW_i(\mathcal{R}_a) += count \cdot c_{j,y}$; $limit[m] -= count$;
5: end for
```

5.2 Resource-Oriented Global Waiting Time

Next, we study a *resource-oriented* approach to *tightening* the calculation of tasks' total global wait time by exploiting the interference limitation among tasks. Such tightened overhead can in turn improve the acceptance test of given task-to-processor mappings. For such a purpose, we further define $\mathcal{S}_a(\Psi)$ as the set of critical sections of all tasks in Ψ where resource $\mathcal{R}_a$ is accessed; that is, $\mathcal{S}_a(\Psi) = \cup_{\forall \tau_i \in \Psi} \mathcal{S}_{i,a}$. Here, we assume that the critical sections in $\mathcal{S}_a(\Psi)$ are in descending order of their sizes.

For a given task-to-processor mapping Π where task $\tau_i \in \Psi_k$, Algorithm 1 summarizes the steps to calculate the maximum total global waiting time $BW_i(\mathcal{R}_a)$ that can be experienced by any job of task τ_i due to accesses of resource $\mathcal{R}_a$ ($\in \Re_i$). Here, based on Proposition 3, the limits on number of interference from processors are first initialized (line 1). Then, in descending order of their sizes, the critical sections in $\mathcal{S}_a(\Psi)$ are processed one at a time. Note that, only if a critical section is from a task on a processor other than $\mathcal{P}_k$, can it possibly interfere with task τ_i (line 2). Next, the number of interference that a critical section $s_{j,y}$ can put on task τ_i is subject to the limit of its task τ_j as well as the remaining limit from τ_j's processor (line 3). Finally, the global waiting time $BW_i(\mathcal{R}_a)$ cumulates and the processor's limit is updated properly (line 4).

Considering all resources that are accessed by task τ_i, its maximum total global waiting time can be given as:

$$BW_i = \sum_{\mathcal{R}_a \in \Re_i} BW_i(\mathcal{R}_a) \tag{14}$$

From Algorithm 1, we can see that, by incorporating the interference limits, the longest critical section (with the size of $tp_m^{max}(\mathcal{R}_a)$) of tasks accessing $\mathcal{R}_a$ on a processor $\mathcal{P}_m$ ($m \neq k$) may not be able to interfere with task τ_i when every time it accesses $\mathcal{R}_a$. Therefore, the maximum total global waiting time BW_i of task τ_i obtained from such a resource-oriented approach and given by Equation (14) is no more than that of Equation (5). Note that, the local blocking time given by Equation (1) rather relies on individual critical sections, which cannot be further reduced. However, with the tightened global waiting time, the resource-oriented approach can obtain smaller values for the synchronization-cognizant processor load given in Equation (2), which can improve schedulability as shown in Section 7.

6 Synchronization-Cognizant Mapping

For partitioned scheduling, there are two main issues when allocating tasks to processors: a) the order (i.e., priority) of

Algorithm 2 : Outline of SC-TMA

```
Input: $\Psi$ (the task set) and $M$ (number of processors);
Output: A feasible mapping $\Pi$ or FAIL;
1: $\Pi = \emptyset$; $L^{sc,min} = \infty$;
2: for ($K : \lceil U \rceil \to M$) do
3:     $\Phi = \Psi$; $\Psi_k = \emptyset$ ($k = 1, \cdots, K$);
4:     while ($\Phi \neq \emptyset$) do
5:         $\tau_i$ =MaxPriority($\Phi, \{\Psi_1, \cdots, \Psi_K\}$);//Section 6.2
6:         $k$ =FindProcessor($\tau_i, \{\Psi_1, \cdots, \Psi_K\}$);//Section 6.3
7:         $\Psi_k = \Psi_k \cup \{\tau_i\}$; $\Phi = \Phi - \{\tau_i\}$;
8:     end while
9:     $\Pi^{tmp} = \{\Psi_1, \cdots, \Psi_K\}$;
10:    Calculate $L^{sc}(\Pi^{tmp})$;//from Equations (1-4) and (14)
11:    if ($L^{sc}(\Pi^{tmp}) \leq 1$ and $L^{sc}(\Pi^{tmp}) < L^{sc,min}$) then
12:        $\Pi = \Pi^{tmp}$; $L^{sc,min} = L^{sc}(\Pi^{tmp})$;
13:    end if
14: end for
15: Return ($\Pi \neq \emptyset$ ? $\Pi$: FAIL);
```

tasks being allocated; and b) the selection of an appropriate target processor for the next task to be allocated. With different objectives (e.g., to minimize the number of processors deployed or to balance workload among processors), various heuristics (such as BFD and WFD [32]) have been studied for ordering the tasks and selecting the target processors.

In [24], we have studied a *synchronization-aware WFD (SA-WFD)* task mapping scheme that considers synchronization overhead when ordering and allocating tasks to processors. However, the order (i.e., priority) of tasks relies on the *fixed* estimation of their maximum synchronization overhead and the selection of target processor is based on a simple metrics of resource similarity (defined as the number of same resources accessed by tasks). Therefore, SA-WFD can still fail to generate feasible mappings for many task sets [24].

In this work, based on the tightened resource-oriented global waiting time of tasks, we propose the *synchronization-cognizant task mapping algorithms (SC-TMA)*. The objective is to obtain feasible and load-balanced task-to-processor mappings for more task sets with shared resources and thus improve the schedulability. Specifically, SC-TMA prioritizes and allocates tasks based on *iteratively* updated synchronization overhead of tasks that considers the constantly changing limits on interference among tasks during the mapping process.

6.1 Overview of SC-TMA

From Section 4, we know that a task set with shared resources may be schedulable on fewer processors but not on more processors. That is, not all the available M processors may be utilized in a feasible task-to-processor mapping. Moreover, for a task set Ψ with system utilization U, the minimum number of processors needed to successfully schedule the tasks is $\lceil U \rceil$. Therefore, in order to find the best mapping with the minimum system load and an appropriate number of processors, Algorithm 2 gives the outline of SC-TMA.

Here, we search over all possible number of processors (line 2). For each case, we select iteratively the highest priority task one at a time, where tasks' priorities are updated based on current partial mapping information (line 5), and its corresponding processor (line 6). Once all tasks are allocated, we can calculate the synchronization-cognizant system load of the current mapping based on Equations (1-4) and (14). If the current mapping is feasible and better than the best mapping obtained so far (line 11), the best mapping will be updated (line 12). In the end, SC-TMA either fails to find a feasible mapping or returns the best feasible one (line 15).

Algorithm 3 : *CalBWMax*(τ_i, $\mathcal{R}_a$, Φ, $\{\Psi_1, \cdots, \Psi_K\}$)

```
Input: $\tau_i \in \Phi$, $\mathcal{R}_a$ ($\in \Re_i$) and $\{\Psi_1, \cdots, \Psi_K\}$;
Output: $BW_i^{max}(\mathcal{R}_a)$;
1: $BW_i^{max}(\mathcal{R}_a) = 0$; $limit[m] = |\mathcal{S}_{i,a}|$ ($m = 1, \cdots, K$);
2: $limit^{total} = (K-1) \cdot |\mathcal{S}_{i,a}|$;
3: for ($s_{j,y} \in \mathcal{S}_a(\Psi)$) do
4:     if ($\tau_j \in \Psi_m$) then
5:         $count = \min\{limit^{total}, \theta_{i,j}, limit[m]\}$;
6:         $limit[m] -= count$;
7:     else
8:         $count = \min\{limit^{total}, \theta_{i,j}, |\mathcal{S}_{i,a}|\}$;
9:     end if
10:    $BW_i^{max}(\mathcal{R}_a) += count \cdot c_{j,y}$; $limit^{total} -= count$;
11: end for
```

6.2 Prioritization of Unmapped Tasks

From Equation (2), we can see that the load of a processor depends heavily on tasks' synchronization overhead, especially the total global waiting time of each task. Following the same idea of SA-WFD [24], SC-TMA also prioritizes unmapped tasks based on their *estimated synchronization-cognizant* utilizations, which is defined as:

$$u_i^{sc,estimate} = \frac{c_i + BW_i^{max}}{p_i} \tag{15}$$

where BW_i^{max} denotes the estimated maximum total global waiting time of task τ_i. However, different from SA-WFD that utilizes the fixed value of BW_i^{max} obtained from Equation (5) (i.e., fixed priority of tasks) [24], SC-TMA relies on a more accurate resource-oriented Equation (14) to calculate BW_i^{max} and adjusts it constantly based on the current partial mapping information. Such considerations, as shown in Section 7, can significantly improve the result task-to-processor mappings.

The detailed steps to estimate the maximum resource-oriented global waiting time $BW_i^{max}(\mathcal{R}_a)$ based on the current partial mapping information are given in Algorithm 3. Here, similar to Algorithm 1, the limitation on the interference among tasks is also incorporated. Note that, not all tasks have been mapped to processors in the partial mapping. Therefore, when K ($\leq M$) processors are considered, there is an additional limit on the total number of interference that can be experienced by a job of task τ_i due to accesses of resource $\mathcal{R}_a$ (line 2). Moreover, the interference limits from mapped and unmapped tasks are differentiated (lines 4 to 9). Once $u_i^{sc,estimate}$ for every unmapped task τ_i is obtained according to Equations (14) and (15), the highest priority task to be mapped next will be the one with the largest $u_i^{sc,estimate}$.

6.3 Selection of A Target Processor

To select an appropriate target processor for the highest priority task τ_i to be mapped next, we adopt the *worst-fit* principle as it can normally result in a workload-balanced mapping [32]. That is, the task τ_i should be mapped to a processor $\mathcal{P}_m$ such that $L^{sc}(\{\Psi_1, \cdots, \Psi_m \cup \{\tau_i\}, \cdots, \Psi_K\})$ $(m = 1, \cdots, K)$ is minimized. When there exist more than one processors that lead to the same minimum synchronization-cognizant system load, to provide more space for future tasks and increase the chance of obtaining a feasible mapping, the one that can minimize the lowest processor load should be selected. If there is still a tie, any of the processors can be chosen.

6.3.1 Probe-Based Processor Selection

From Section 3.2, the allocation of task τ_i to a processor $\mathcal{P}_m$ can affect not only the tasks on $\mathcal{P}_m$ due to changes in local blocking time but also tasks on other processors with possibly increased global waiting time. Therefore, for the case of allocating τ_i to $\mathcal{P}_m$, the loads on all processors (and thus the system load) need to be updated according to Equations (1-4) and (14) as well as Algorithm 1.

Note that, there are K possible allocations for task τ_i. Based on the above mentioned criteria (i.e., minimize system load and then the lowest processor load), the most appropriate processor $\mathcal{P}_m$ can thus be selected. This scheme is called SC-TMA with probe-based processor selection (denoted as *SC-TMA-Probe*). Here, by re-calculating the loads on *all* processors for every possible allocation of τ_i, SC-TMA-Probe has rather high complexity.

6.3.2 Quick Processor Selection

As a more efficient scheme with reduced complexity, we introduce another SC-TMA with quick processor selection, which is denoted as *SC-TMA-Quick*. Essentially, it focuses on only one processor at a time. That is, for any processor $\mathcal{P}_m$, there are only two possibilities when allocating task τ_i: either τ_i is mapped to $\mathcal{P}_m$ or not. Then, the new processor load on $\mathcal{P}_m$ can be estimated for both possibilities as follows.

When τ_i is assumed to be mapped to $\mathcal{P}_m$, its local blocking time and global waiting time can be updated accordingly. Then, for the existing tasks in Ψ_m, their global waiting time will remain unchanged. Moreover, for the local blocking time, adjustments are only needed for these tasks in Ψ_m that have periods less than that of τ_i. Therefore, for the case of τ_i being mapped to $\mathcal{P}_m$, based on Equation (2), $\mathcal{P}_m$'s new processor load $L^{sc}_m(\{\cdots, \Psi_m \cup \{\tau_i\}, \cdots\})$ can be calculated.

When τ_i is assumed to be mapped to a processor other than $\mathcal{P}_m$, for the existing tasks in Ψ_m, their global waiting times can be estimated by assuming that τ_i always bring in the maximum number of interference on them (similar to line 3 in Algorithm 3) since it is not known exactly which processor τ_i will be on. Then, their local blocking times can be adjusted accordingly. At the end, $\mathcal{P}_m$'s new processor load $L^{sc}_m(\{\cdots, \Psi_m, \cdots\})$ can be re-calculated.

Once we get two arrays of new loads for all K processors, two candidates $\mathcal{P}_x$ and $\mathcal{P}_y$ will be identified. Here, $L^{sc}_x(\{\cdots, \Psi_x \cup \{\tau_i\}, \cdots\})$ has the minimum value and $L^{sc}_y(\{\cdots, \Psi_y, \cdots\})$ has the maximum value within their respective arrays. Following the above mentioned principles, the task τ_i will be allocated to processor $\mathcal{P}_y$ only if the following two conditions are satisfied:

$$L^{sc}_x(\{\cdots, \Psi_x, \cdots\}) < L^{sc}_x(\{\cdots, \Psi_x \cup \{\tau_i\}, \cdots\})$$
$$\max\{L^{sc}_m(\{\cdots, \Psi_m \cup \{\tau_i\}, \cdots\})\} \leq L^{sc}_y(\{\cdots, \Psi_y, \cdots\})$$

Otherwise, τ_i will be allocated to processor $\mathcal{P}_x$.

7 Evaluations and Discussions

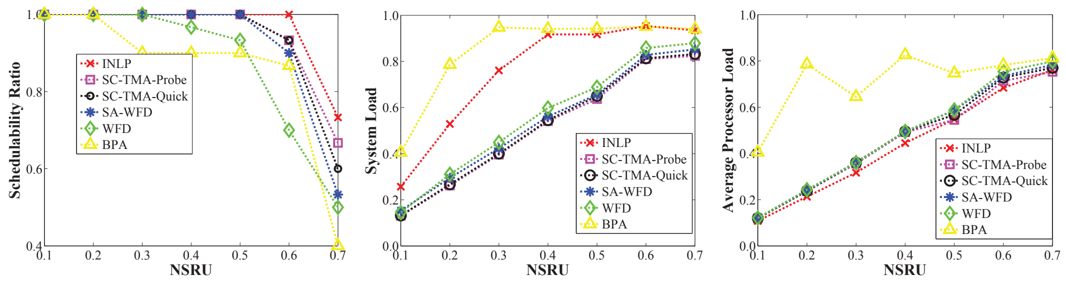
In this section, we evaluate the performance of the proposed *synchronization-cognizant task mapping algorithm (SC-TMA)* through extensive simulations. For comparison, we implemented the conventional WFD (which allocates tasks solely based on their utilizations without synchronization overhead being considered [32]), our previous *synchronization-aware WFD (SA-WFD)* [24] and the macrotask-based BPA [34]. Here, as BPA adopts a different synchronization protocol, the calculation of processor loads conforms to the principles as presented in [25].

We compare these task mapping schemes based on the following performance metrics: a) *Schedulability Ratio*, which is defined as the ratio of the number of schedulable task sets over the total number of task sets considered; b) *System Load ($L^{sc}()$)* as defined in Equation (3), which incorporate synchronization overhead and essentially indicates the quality of resulting task-to-processor mappings in terms of how close they meet the schedulability sufficient condition stated in Proposition 1; and c) *Average Processor Load* $\frac{\sum_{k=1}^{K} L^{sc}_k()}{K}$, where K refers to the number of deployed processors in the resulting mapping and $L^{sc}_k()$ is the processor load as defined in Equation (2), it indicates how well the synchronization-cognizant workload is distributed among the processors. Here, the system load and average processor load only account for the schedulable task sets.

7.1 Simulation Settings

There are many factors in the system that can affect the performance of the mapping algorithms under consideration. In this work, we vary the following parameters as summarized in Table 1. The number of available processors (M) and the number of shared resources in the system (R). The *normalized system raw utilization* without considering synchronization cost $NSRU = \frac{U}{M}$, where U is the system utilization as defined in Section 3. For tasks, we consider the number of tasks N, the number of critical sections in a task and the critical section ratio CSR (defined as the total length of critical sections of a task over its WCET). Here, the degree of resource contention among the tasks in the systems is determined by R, CSR as well as the number of critical sections per task n^{cs}_i: smaller R, larger CSR and number of critical sections mean higher resource contention and more complex synchronization requirements among tasks; and *vise versa*.

For a given set of M, $NSRU$, N, CSR and number of critical sections per task, following the similar steps as in [24], the synthetic task sets are generated as follows. The



(a) Ratio of schedulable task sets (b) Result system load (c) Average processor load

Fig. 2: Performance of the algorithms with varying $NSRU$ for small task sets;

utilization of a task τ_i is set as $u_i = \frac{NSRU \cdot M}{N}$. Then, the period of τ_i is randomly selected from one of the three types of periods in Table 1. Next, the task's WCET c_i is obtained uniformly in the range of $[0.2 \cdot p_i \cdot u_i, 1.8 \cdot p_i \cdot u_i]$. The number of critical sections in τ_i is obtained within the range of $[1, 8]$, followed by randomly selected resource for each critical section. The execution time of a critical section is generated randomly within $[\frac{0.2 \cdot c_i \cdot CSR}{n_i^{cs}}, \frac{1.8 \cdot c_i \cdot CSR}{n_i^{cs}}]$. In the end, the execution time of non-critical sections are obtained with their relative locations being randomly assigned.

TABLE 1: System parameters for the simulations

Parameters	Values/ranges
Number of processors (M)	2, 4, 8, 16
Normalized system raw utilization ($NSRU$)	$[0.1, 0.7]$
Number of resources (R)	[1,10]
Number of tasks (N)	[8,20], [40, 120]
Period of tasks	[50, 200], [200, 500], [500, 2000]
Critical sections per task	[1, 8]
Critical section ratio (CSR)	$0.003 - 0.03$

7.2 Evaluations for Small Scale Problems

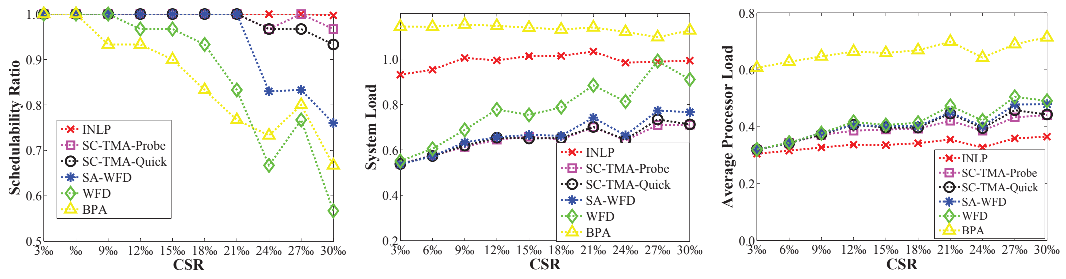
First, we conducted simulations for small scale problems with upto 20 ($N = [8, 20]$) tasks and 4 processors ($M = 4$). The goal is to see how close the task mapping schemes (i.e., BPA, WFD, SA-WFD, SC-TMA-Quick and SC-TMA-Probe) can perform in terms of generating feasible task-to-processor mappings when compared to that of an INLP (Integer Non-Linear Programming) based task mapping approach. Here, the INLP is formulated similar to the one in [24] and implemented using the Lingo tool [1]. Since it is difficult to incorporate the tasks' tightened resource-oriented waiting time (as calculated in Algorithm 1) into the constraints of the INLP formulation, the one in Equation (5) is used. Here, to obtain results of INLP in reasonable amount of time, 200 task sets are generated for each setting and the average results are reported.

By varying the normalized system raw utilization $NSRU$ (recall that it does not include synchronization cost), Figure 2 shows the performance of the mapping algorithms under consideration. In general, the schedulability ratios of these algorithms decrease with increasing $NSRU$ due to larger workload (Figure 2(a)). Not surprisingly, by searching through all possible combinations, INLP obtains the best schedulability result for all cases.

The results also show that, by effectively incorporating the iteratively tightened synchronization overhead into the task mapping process, the SC-TMA based schemes obtain much better schedulability results compared other partitioning heuristics (e.g., WFD, SA-WFD and BPA). Moreover, with the more precise estimation of the synchronization-cognizant load on processors, SC-TMA-Probe performs more close to INLP when compared to that of SC-TMA-Quick, but with higher time complexity. Note that, by exploring the macrotasks that cannot directly incorporate synchronization overhead, BPA results in the worst performance in terms of schedulability. When $NSRU$ becomes higher than 0.6, BPA fails to generate a feasible mapping for almost all task sets.

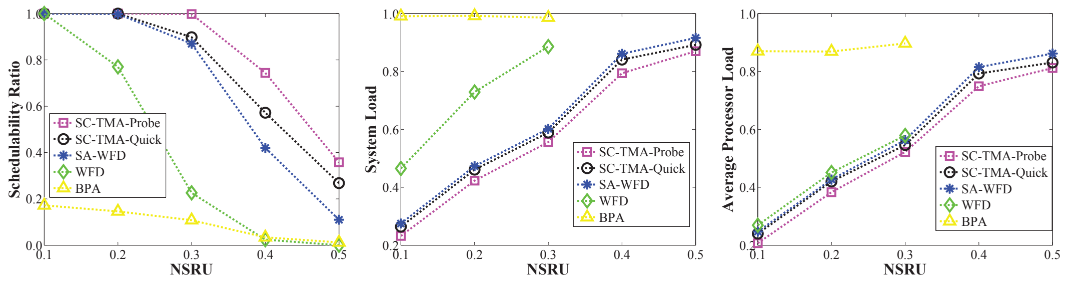
Moreover, for the task sets where the mapping schemes obtain feasible task-to-processor mappings, Figures 2(b) and 2(c) further shows the result synchronization-cognizant system load and average processor load, respectively. Here, as $NSRU$ increases, there are more workload and the critical sections of tasks become larger. Therefore, the higher system loads and average processor loads, which incorporate synchronization overhead, increase as well. Although INLP obtains better schedulability ratio, it cannot incorporate the tightened resource-oriented global waiting time during the mapping process and thus results in higher system load for task sets with feasible mappings. Moreover, with the principles of minimizing the maximum and minimum estimated processor loads being considered during the task mapping process, SC-TMA based schemes perform better than INLP and can obtain lower system load as well as average processor load with more workload-balanced mappings. For BPA, with the adopted Best-Fit (BF) heuristic, it obtains much higher and relatively steady system load.

Figure 3 further illustrates the effects of CSR on the performance of these schemes with $NSRU = 0.25$. As mentioned earlier, larger values of CSR indicate more resource contention among tasks and thus higher synchronization overhead. Therefore, the schedulability ratios generally decrease with increasing $NSRU$ (Figure 3(a)). As CSR becomes larger, BPA shows better schedulability performance by clustering tasks that share the same resources into macrotasks. Again, SC-TMA based schemes perform better on both schedulability and system/processor loads than other existing mapping heuristics due to similar reasoning as discussed above.



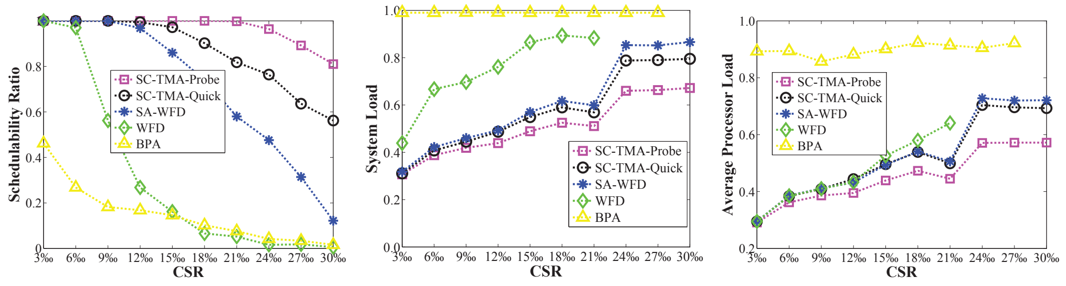
(a) Ratio of schedulable task sets (b) Result system load (c) Average processor load

Fig. 3: Performance of the algorithms with varying CSR for small task sets with $NSRU = 0.25$;



(a) Ratio of schedulable task sets (b) Result system load (c) Average processor load

Fig. 4: Performance of the algorithms with varying $NSRU$ for large task sets.



(a) Ratio of schedulable task sets (b) Result system load (c) Average processor load

Fig. 5: Performance of the algorithms with varying CSR for large task sets.

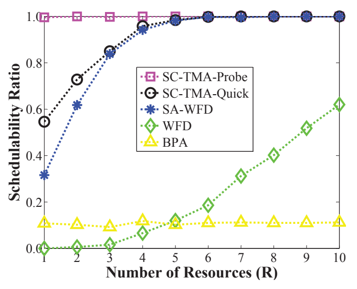
7.3 Evaluation for Large Scale Problems

Next, we consider large scale problems with upto 120 (i.e., $N \in [40, 120]$) tasks in a task set. Without specified otherwise, the default values for other parameters are: $M = 16$, $NSRU = 0.25$ and $CSR = 0.009$. Moreover, for the results reported below, each data point corresponds to the average of $10,000$ task sets.

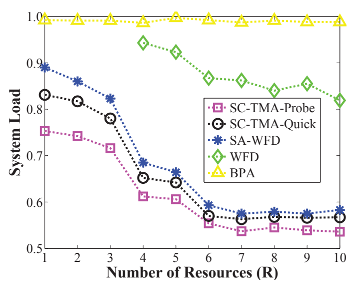
Figures 4 and 5 show the impacts of varying $NSRU$ and CSR on the performance of the mapping schemes under consideration, respectively. Compared to the results for small scale problems (as shown in Figures 2 and 3), similar trends can be observed. However, it is noteworthy that, when there are more tasks and processors, the synchronization requirements among tasks become more complex with much higher synchronization overhead. Therefore, the schedulability ratios of all schemes become much smaller as it is more likely for the considered mapping schemes to fail to obtain a feasible mapping for a given task set (note that the range of $NSRU \in [0.1, 0.5]$ in Figure 4 is smaller than that of Figure 2), especially for larger values of CSR (Figure 5(a)).

Note that, when $NSRU \geq 0.3$, BPA and WFD fail to obtain feasible mappings for almost all generated task sets. Therefore, no result is shown for them in Figures 4(b) and 4(c). Here, with the same reasoning, our SC-TMA based schemes can get better task-to-processor mappings with more balanced workload among processors. For the case of varying CSR, similar results are obtained as in Figures 5(b) and 5(c).

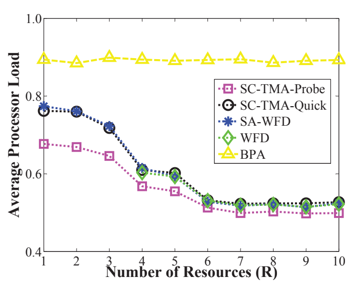
Figure 6 further shows the impacts of the number of shared resources on the performance of these mapping schemes. Note that, with other parameters (such as the number of critical sections per task and CSR) are fixed, having fewer number of shared resources means higher probability of more than one tasks accessing the same resource and thus stronger synchronization requirements. When the task synchronization requirements are strong (e.g., $R \leq 4$), SC-TMA-Probe can obtain the best schedulability performance compared to the others as explained earlier. Again, due to their synchronization-cognizant nature that targets at load-balanced mappings, SC-TMA based schemes can obtain lower system load and average processor load as shown in Figures 6(b) and 6(c), respectively.



(a) Ratio of schedulable task sets

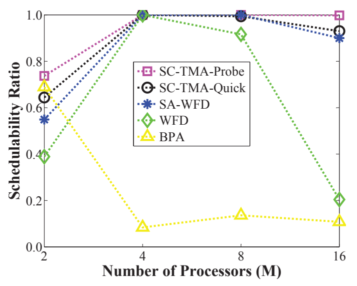


(b) System utilization

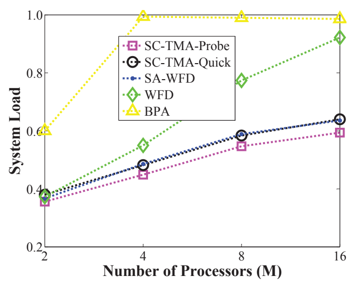


(c) Average utilization

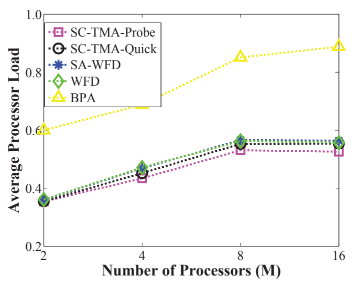
Fig. 6: Performance of the algorithms with varying R.



(a) Ratio of schedulable task sets



(b) System utilization



(c) Average utilization

Fig. 7: Performance of the algorithms with varying M.

Finally, we evaluate the performances of the mapping schemes with varying number of available processors in the system and the results are shown in Figure 7. The results show that our SC-TMA based schemes perform pretty consistently in terms schedulability ratio (i.e., obtaining feasible mappings for task sets with shared resources), when compared to other existing mapping heuristics. Without taking synchronization overhead into consideration, the schedulability ratio of WFD degrades dramatically for systems with more processors (i.e., $M \geq 8$), where stronger synchronization requirements among tasks are expected.

8 Conclusions

Focusing on multiprocessor real-time systems with shared resources, we study the utilization bound and efficient task mapping schemes for the partitioned-EDF scheduling with MSRP resource access protocol. After illustrating the phenomenon of *scheduling anomaly* due to synchronization requirements among tasks, we develop the first *synchronization-cognizant utilization bound* for such systems under partitioned-EDF with MSRP. After showing the mapping problem of tasks with shared resources is NP-hard, we propose two efficient *synchronization-cognizant task mapping algorithms (SC-TMA)* that rely on the tightened synchronization overhead and aim at obtaining workload-balanced mappings and thus better schedulability. The simulation results show that, the schedulability ratio and (average) system load under SC-TMA based schemes are close to that of an INLP (Integer Non-Linear Programming) based solution for small task systems. When compared to the existing task mapping algorithms, SC-TMA can obtain much better schedulability and lower/balanced load on the deployed processors.

References

[1] http://www.lindo.com/.

[2] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *Proc. of The 22th IEEE Real-Time Systems Symposium*, pages 193–202, Dec. 2001.

[3] B. Andersson and J. Jonsson. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In *Proc. of the 15th Euromicro Conference on Real-Time Systems*, pages 33–40, 2003.

[4] B. Andersson and E. Tovar. Multiprocessor scheduling with few preemptions. In *Proc. of the IEEE Int'l Conference on Embedded and Real-Time Computing Systems and Applications*, pages 322–334, 2006.

[5] T. P. Baker. Stack-based scheduling for realtime processes. *Real-Time Syst.*, 3(1):67–99, April 1991.

[6] T.P. Baker. Multiprocessor edf and deadline monotonic schedulability analysis. In *Proc. of The 24th IEEE Real-Time Systems Symposium*, pages 120–129, Dec. 2003.

[7] S. Baruah. Techniques for multiprocessor global schedulability analysis. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 119–128, 2007.

[8] S. Baruah and T. Baker. Schedulability analysis of global edf. *Real-Time Systems*, 38(3):223–235, 2008.

[9] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, 1996.

[10] A. Block, H. Leontyev, B.-B. Brandenburg, and J.-H. Anderson. A flexible real-time locking protocol for multiprocessors. In *Proc. of the 13th IEEE Int'l Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 47–56, 2007.

[11] B.-B. Brandenburg and J.-H. Anderson. Optimality results for multiprocessor real-time locking. In *Proc. of the 31st IEEE Real-Time Systems Symposium (RTSS)*, pages 49–60, 2010.

[12] J. Brodt. Revving up with automotive multicore, 2008. available at http://www.edn.com/article/ca6528579.html.

[13] H. Cho, B. Ravindran, and E.G. Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 101–110, 2006.

[14] R. Davis and A. Burns. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 398–409, 2009.

[15] R.-I. Davis and A. Burns. A survey of hard real-time scheduling for

multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, October 2011.

[16] M. L. Dertouzos and A. K. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Trans. On Software Engineering*, 15(12):1497–1505, 1989.

[17] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operation Research*, 26(1):127–140, 1978.

[18] A. Easwaran and B. Andersson. Resource sharing in global fixed-priority preemptive multiprocessor scheduling. In *Proc. of the 30th IEEE Real-Time Systems Symposium (RTSS)*, pages 377–386, 2009.

[19] K. Funaoka, S. Kato, and N. Yamasaki. Work-conserving optimal real-time scheduling on multiprocessors. In *Proc. of the Euromicro Conference on Real-Time Systems*, pages 13–22, 2008.

[20] P. Gai, G. Lipari, and M.D. Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *Proc. of the 22nd IEEE Real-Time Systems Symposium*, pages 73–82, Washington, DC, USA, 2001. IEEE Computer Society.

[21] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems*, 25(2-3):187–205, 2003.

[22] N. Guan, M. Stigge, W. Yi, and G. Yu. New response time bounds for fixed priority multiprocessor scheduling. In *Proc. of the IEEE Real-Time Systems Symposium*, pages 387–397, 2009.

[23] N. Guan, M. Stigge, W. Yi, and G. Yu. Fixed-priority multiprocessor scheduling with liu & layland's utilization bound. In *Proc. of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 165–174, 2010.

[24] J.-J. Han, X. Wu, D. Zhu, H. Jin, L. T. Yang, and J.-L. Gaudiot. Synchronization-aware energy management for vfi-based multicore real-time systems. *IEEE Trans. Comput.*, 61(12):1682–1696, 2012.

[25] P.-C. Hsiu, D.-N. Lee, and T.-W. Kuo. Task synchronization and allocation for many-core real-time systems. In *Proc. of the 9th ACM international Conference on Embedded software (EMSOFT)*, pages 79–88, 2011.

[26] S. Kato, K. Funaoka, and N. Yamasaki. Semi-partitioned scheduling of sporadic task systems on multiprocessors. In *Proc. of the 21th Euromicro Conference on Real-Time Systems*, pages 249–258, 2009.

[27] S. Kato and N. Yamasaki. Portioned edf-based scheduling on multiprocessors. In *Proc. of the 8th ACM Int'l conference on Embedded software*, pages 139–148, 2008.

[28] K. Lakshmanan, D.-de Niz, and R. Rajkumar. Coordinated task scheduling, allocation and synchronization on multiprocessors. In *Proc. of the 30th IEEE Real-Time Systems Symposium (RTSS)*, pages 469–478, 2009.

[29] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt. Dp-fair: A simple model for understanding optimal multiprocessor scheduling. In *Proc. of the Euromicro Conference on Real-Time Systems*, pages 3–13, 2010.

[30] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of ACM*, 20(1):46–61, 1973.

[31] J.M. Lopez, J.L. Diaz, and D.F. Garcia. Minimum and maximum utilization bounds for multiprocessor rm scheduling. In *Proc. of the 13th Euromicro Conference on Real-Time Systems*, pages 67–75, 2001.

[32] J.M. Lopez, J.L. Diaz, and D.F. Garcia. Utilization bound for edf scheduling on real-time multiprocessor systems. *Real-Time Systems*, 28(1):39–68, 2004.

[33] G. Martin. Overview of the mpsoc design challenge. In *Proc. of the 43rd Annual Design Automation Conference (DAC)*, pages 274–279, 2006.

[34] F. Nemati, T. Nolte, and M. Behnam. Partitioning real-time systems on multiprocessors with shared resources. In *Proc. of the 14th international conference on Principles of distributed systems*, OPODIS, pages 253–269, 2010.

[35] D.I. Oh and T.P. Baker. Utilization bound for n-processor rate monotone scheduling with stable processor assignment. *Real-Time Systems*, 15(2):183–193, 1998.

[36] K. Olukotun, B.-A. Nayfeh, L. Hammond, K. Wilson, and K. Chang. The case for a single-chip multiprocessor. In *Proc. of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 2–11, 1996.

[37] X. Qi, D. Zhu, and H. Aydin. Global scheduling based reliability-aware power management for multiprocessor real-time systems. *Real-Time Systems: The International Journal of Time-Critical Computing Systems*, 47(2):253–284, 2011.

[38] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the 9th IEEE Real-Time Systems Symposium (RTSS '88)*, pages 259–269, Washington, DC, USA, 1988. IEEE Computer Society.

[39] P. Regnier, G. Lima, E. Massa, G. Levin, and S. Brandt. Run: Optimal multiprocessor real-time scheduling via reduction to uniprocessor. In *Proc. of the 32nd IEEE Real-Time Systems Symposium*, pages 104–115, 2011.

[40] L. Sha, T. Abdelzaher, K.-E. En, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A.-K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Syst.*, 28(2-3):101–155, November 2004.

[41] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Comput.*, 39(9):1175–1185, September 1990.

[42] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Proc. of the 20th Euromicro Conference on Real-Time Systems*, pages 181–190, Jul. 2008.

[43] D. Zhu, D. Mossé, and R. Melhem. Periodic multiple resource scheduling problem: how much fairness is necessary. In *Proc. of The 24^{th} IEEE Real-Time Systems Symposium*, pages 142–151, Dec. 2003.

Jian-Jun Han received the PhD degree in computer science and engineering from Huazhong University of Science and Technology (HUST), in 2005. He is now an Associate Professor at the School of Computer Science and Technology in HUST. He worked at the University of California, Irvine as a visiting scholar between 2008 and 2009, and at the Seoul National University between 2009 and 2010. His research interests include real-time system and parallel computing. He is currently a member of IEEE and ACM.

Dakai Zhu received the PhD degree in Computer Science from University of Pittsburgh in 2004. He is currently an Associate Professor in the Department of Computer Science at the University of Texas at San Antonio. His research interests include real-time systems, power aware computing and fault-tolerant systems. He was a recipient of the US National Science Foundation (NSF) Faculty Early Career Development (CAREER) Award in 2010. He is a member of the IEEE and the IEEE Computer Society.

Xiaodong Wu is now a PhD Candidate at the School of Computer Science and Technology in Huazhong University of Science and Technology (HUST). His research interests include real-time scheduling algorithm and embedded system.

Laurence T. Yang is with the Department of Computer Science, St. Francis Xavier University, Antigonish, NS, Canada and holds adjunct position at HUST. His current research interests include high performance computing and networking, embedded systems, ubiquitous/pervasive computing, and intelligence. His research is supported by National Sciences and Engineering Research Council, Canada and Canada Foundation for Innovation.

Hai Jin is a Professor and serves as Dean of the School of Computer Science and Technology at HUST. Jin received his PhD in computer engineering from HUST in 1994. Jin worked at the University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He is a senior member of the IEEE and a member of the ACM. He has coauthored 15 books and published over 400 papers. His research interests include computer architecture, virtualization technology, cluster computing and peer-to-peer computing.

Appendix

Estimating Processor Load for SC-TMA-Quick

Without loss of generality, we assume that task τ_i is chosen to be assigned (i.e., τ_i has the maximum $u_i^{sc,estimate}$ in set Φ). For each processor, task τ_i has two options: whether τ_i is allocated to it or not. Due to the changes of allocated tasks' synchronization overheads, for an allocated task τ_j, we denote its *estimated global waiting time* by BW_j^{est} and *estimated local blocking time* by B_j^{est}. Initially, we have $BW_j^{est} = BW_j$ and $B_j^{est} = B_j$. We can define BW_i^{est} and B_i^{est} as well with initial values being BW_i^{max} (see Algorithm 3) and 0, respectively. Moreover, similar to Equation (2), the *estimated load* of processor $\mathcal{P}_k$ is defined as

$$L_k^{sc,est} = \max\left\{\frac{B_a^{est}}{p_a} + \sum_{\forall \tau_b \in \Psi_k}^{p_b \leq p_a} \frac{c_b + BW_b^{est}}{p_b} | \forall \tau_a \in \Psi_k\right\} \tag{16}$$

Further, the *estimated system load* is defined as $L_{max}^{sc,est} = \max\{L_m^{sc,est}(\Pi)|\forall \mathcal{P}_m\}$ and the minimum estimated load of processors is defined as $L_{min}^{sc,est} = \min\{L_m^{sc,est}(\Pi)|\forall \mathcal{P}_m\}$.

The objective of our task assignment heuristics is to find a processor for task τ_i that can result in minimum $L_{max}^{sc,est}$ first (for more balanced workload) and then minimum $L_{min}^{sc,est}$ (for better schedulability possibility for those un-assigned tasks).

The estimated synchronization overheads of allocated tasks and task τ_i are determined as follows:

- **C1:** Assume that task τ_i will not be mapped onto processor $\mathcal{P}_k$ and an allocated task τ_j ($\in \Psi_k$) needs to access the same resource(s) as τ_i. Recall that $tt_i^{max}(\mathcal{R}_a)$ refers to the maximum execution time for τ_i to access $\mathcal{R}_a$ once.
  - For each resource $\mathcal{R}_a$ that is accessed by both τ_i and τ_j (i.e., $\mathcal{R}_a \in (\Re_i \cap \Re_j)$), BW_j will be increased by at most $tt_i^{max}(\mathcal{R}_a) \cdot |\mathcal{S}_{j,a}|$:

    $$BW_j^{est} = \min\{BW_j^{max}, BW_j^{'} + tt_i^{max}(\mathcal{R}_a) \cdot |\mathcal{S}_{j,a}|\} \tag{17}$$

    where BW_j^{max} records the newest upper-bound of BW_j before τ_j is assigned (see Section 5.2).
  - Similarly, for τ_j's each critical section $s_{j,x}$ ($r_{j,x} = a$) that needs to access resource $\mathcal{R}_a$, its waiting time for $\mathcal{R}_a$ is increased by at most $tt_i^{max}(\mathcal{R}_a)$:

    $$BW_{j,x}^{est} = \min\{BW_{j,x}^{max}, BW_{j,x} + tt_i^{max}(\mathcal{R}_a)\} \tag{18}$$

    where $BW_{i,x}^{max}$ can be obtained from Algorithm 3: here each $limit[m]$ (Line 1) is set to 1 and $limit^{total}$ is set to $(K-1)$ (Line 2) to account for the global waiting time for executing only one critical section.
  - For a task τ_m ($\tau_m \in \Psi_k \wedge p_m < p_j$), similar to Equation (1), its B_m^{est} can be upper bounded by:

    $$\begin{aligned} B_m^{est} = \; & \max\{B_m^{est}, \max\{BW_{j,x}^{est} + c_{j,x}|\forall s_{j,x} : \\ & \tau_m \in \Psi_k \wedge p_m < p_j \wedge \mathcal{R}_{r_{j,x}} \in \Re_i\}\} \end{aligned} \tag{19}$$

  Note that, only one task is allocated at a time under our partitioned algorithm. The waiting times for an assigned task τ_j to access its resources can vary only when it accesses the same resource(s) as τ_i. Thus, we only need to consider the impact of such variations on B_m^{est} as shown in above equation.
- **C2:** Assume that task τ_i will be mapped onto processor $\mathcal{P}_k$. B_j^{est} ($\tau_j \in \Psi_k \wedge p_j < p_i$) can be calculated similar to Equation (1), that is,

  $$B_j^{est} = \max\{B_j, \max\{BW_{i,x}^{max} + c_{i,x}|\forall s_{i,x} : r_{i,x} \neq 0\}\} \tag{20}$$
- **C3:** For the same case (i.e., τ_i is assumed to be allocated to $\mathcal{P}_k$), we can obtain B_i^{est} directly from Equation (1).

We denote the estimated load of a processor $\mathcal{P}_k$ due to whether task τ_i is assigned to this processor or not as $L_k^{sc}(\cdots, \Psi_k \cup \{\tau_i\}, \cdots)$ ($L_k^{sc,est}(\Psi_k \cup \{\tau_i\}$ for abbreviation) and $L_k^{sc}(\cdots, \Psi_k, \cdots)$ ($L_k^{sc,est}(\Psi_k)$ for abbreviation), respectively. Initially, there is $L_k^{sc,est}(\Psi_k \cup \{\tau_i\}) = L_k^{sc,est}(\Psi_k) = L_k^{sc}$, where L_k^{sc} can be obtained from Line 10 in Algorithm 2.

Following above analysis (see C1, C2 and C3), the basic steps for computing each $L_k^{sc,est}(\Psi_k)$ and $L_k^{sc,est}(\Psi_k \cup \{\tau_i\})$ are respectively given in Algorithm 4 and Algorithm 5. Here, tasks on each processor are sorted in order of non-decreasing relative deadlines (which is called *normal order*).

Algorithm 4 Calculating $L_k^{sc,est}(\Psi_k)$

```
Input: $\Psi_k$;
Output: $L_k^{sc,est}(\Psi_k)$;
1:  $\varphi = 0$; $\omega = 0$; $L_k^{sc,est}(\Psi_k) = L_k^{sc}$;
2:  for (each $\tau_j$ in set $\Psi_k$ with normal order) do
3:      for (each $z_{j,x}$ ($\mathcal{R}_{r_{j,x}} \in \Re_i$)) do
4:          Calculate $BW_{j,x}^{est}$ from Equation (18);
5:      end for
6:      for (each $\mathcal{R}_a \in (\Re_i \cap \Re_j)$) do
7:          Calculate $BW_j^{est}$ from Equation (17);
8:      end for
9:  end for
10: for (each $\tau_j$ in set $\Psi_k$ with inverse (non-increasing relative
    deadline) order) do
11:     $B_j^{est} = \max\{B_j^{est}, \varphi\}$;
12:     if ($\Re_j \cap \Re_i \neq \emptyset$) then
13:         $\varphi = \max\{\varphi, \max\{BW_{j,x}^{est} + c_{j,x}|\forall s_{j,x} : \mathcal{R}_{r_{j,x}} \in \Re_i\}\}$; //see Equation (19)
14:     end if
15: end for
16: for (each $\tau_j$ in set $\Psi_k$ by normal order) do
17:     $\omega += \frac{c_j + BW_j^{est}}{p_j}$;
18:     $L_k^{sc,est}(\Psi_k) = \max\{L_k^{sc,est}(\Psi_k), \omega + \frac{B_j^{est}}{p_j}\}$;
19: end for
```

In Algorithm 4, Lines 2 to 9 (the first out-most for-loop) estimate the global waiting times for tasks on processor $\mathcal{P}_k$ to access resources if they access the same resource(s) as τ_i. Consequently, due to such variations, the estimated blocking times for tasks on $\mathcal{P}_k$ are modified accordingly (the second out-most for-loop). In the end, the estimated load of processor $\mathcal{P}_k$ can be directly calculated from Equation (16) as shown in Lines 16 to 19 (the third out-most for-loop).

In Algorithm 5, Line 5 calculates B_j^{est} ($p_j < p_i$) (see C2), while Lines 8 to 10 estimate the blocking time for task τ_i (see C3). Lines 6, 11 and 14 compute the estimated load of $\mathcal{P}_k$ from Equation (16) assuming that τ_i is allocated to $\mathcal{P}_k$.

Algorithm 5 Calculating $L_k^{sc,est}(\Psi_k \cup \{\tau_i\})$

```
Input: $\Psi_k$;
Output: $L_k^{sc,est}(\Psi_k \cup \{\tau_i\})$;
1: $\varphi = 0$; $\omega = \frac{c_i+BW_i^{est}}{p_i}$; $\delta = 0$; $L_k^{sc,est}(\Psi_k \cup \{\tau_i\}) = L_k^{sc}$;
2: for (each $\tau_j$ in set $\Psi_k$ with normal order) do
3:     $\varphi + = \frac{c_j+BW_j^{est}}{p_j}$;
4:     if ($p_j < p_i$) then
5:         Calculate $B_j^{est}$ from Equation (20); $\delta = \varphi$;
6:         $L_k^{sc,est}(\Psi_k \cup \{\tau_i\}) = \max\{L_k^{sc,est}(\Psi_k \cup \{\tau_i\}), \varphi + \frac{B_j^{est}}{p_j}\}$;
7:     else
8:         if ($p_j > p_i$) then
9:             $B_i^{est} = \max\{B_i^{est}, \max\{BW_{j,x} + c_{j,x} | \forall s_{j,x} : r_{j,x} \neq 0\}\}$;
10:        end if
11:        $L_k^{sc,est}(\Psi_k \cup \{\tau_i\}) = \max\{L_k^{sc,est}(\Psi_k \cup \{\tau_i\}), \varphi + \omega + \frac{B_j^{est}}{p_j}\}$;
12:    end if
13: end for
14: $L_k^{sc,est}(\Psi_k \cup \{\tau_i\}) = \max\{L_k^{sc,est}(\Psi_k \cup \{\tau_i\}), \delta + \omega + \frac{B_i^{est}}{p_i}\}$;
```

We assume that processor $\mathcal{P}_x$ has the minimum $L_k^{sc,est}(\Psi_k \cup \{\tau_i\})$ from Algorithm 5 and tie is broken in favor of the processor with the maximum $L_k^{sc,est}(\Psi_k)$ from Algorithm 4. That is,

$$\begin{cases} L_x^{sc,est}(\Psi_x \cup \{\tau_i\}) = \min\{L_k^{sc,est}(\Psi_k \cup \{\tau_i\}) | \forall \mathcal{P}_k\} & \text{(21a)} \\ L_x^{sc,est}(\Psi_x) = \max\{L_k^{sc,est}(\Psi_k) | \forall \mathcal{P}_k : L_k^{sc,est}(\Psi_k \cup \{\tau_i\}) \\ = L_x^{sc,est}(\Psi_x \cup \{\tau_i\})\} & \text{(21b)} \end{cases}$$

If there are still more than one processor satisfying Equation (21), $\mathcal{P}_x$ is set as the processor having the smallest index.

Moreover, we find processor $\mathcal{P}_y$ that has the maximum $L_k^{sc,est}(\Psi_k)$ from Algorithm 4 and tie-breaking favors the processor with the minimum $L_k^{sc,est}(\Psi_k \cup \{\tau_i\})$ from Algorithm 5. That is,

$$\begin{cases} L_y^{sc,est}(\Psi_y) = \max\{L_k^{sc,est}(\Psi_k) | \forall \mathcal{P}_k\} & \text{(22a)} \\ L_y^{sc,est}(\Psi_y \cup \{\tau_i\}) = \min\{L_k^{sc,est}(\Psi_k \cup \{\tau_i\}) | \forall \mathcal{P}_k : \\ L_k^{sc,est}(\Psi_k) = L_y^{sc,est}(\Psi_y)\} & \text{(22b)} \end{cases}$$

If there exists multiple processors satisfying Equation (22), the processor with the smallest index is chosen.

At the end, determining the processor for a task by SC-TMA-Quick is subject to the following criteria.

- Task τ_i is allocated to processor $\mathcal{P}_y$ only if the following two conditions are satisfied:

$$\begin{aligned} L_x^{sc,est}(\Psi_x) < L_x^{sc,est}(\Psi_x \cup \{\tau_i\}) \\ \max\{L_z^{sc,est}(\Psi_z \cup \{\tau_i\}) | \forall \mathcal{P}_z\} \leq L_y^{sc,est}(\Psi_y) \end{aligned} \tag{23}$$

- Otherwise, τ_i is assigned to processor $\mathcal{P}_x$.

Theorem 3. *Our approach to determining the target processor for a task guarantees that first $L_{max}^{sc,est}$ and then $L_{min}^{sc,est}$ are minimized.*

Proof: All processors can be categorized into three subsets: $\mathbb{A}$, $\mathbb{B}$ and $\mathbb{C}$. Any processor $\mathcal{P}_k$ in $\mathbb{A}$ has the minimum $L_k^{sc,est}(\Psi_k \cup \{\tau_i\})$ of all processors (see Equation (21a)), while any processor $\mathcal{P}_k$ in set $\mathbb{B}$ has the maximum $L_k^{sc,est}(\Psi_k)$ of all processors (see Equation (22a)). The other processors form a processor set $\mathbb{C}$. Note that, there may be a processor pertaining to both set $\mathbb{A}$ and set $\mathbb{B}$ (i.e., $\mathbb{A} \cap \mathbb{B} \neq \emptyset$).

Case 1: First, we consider processor $\mathcal{P}_x$ and a processor $\mathcal{P}_k$ ($\mathcal{P}_k \in \mathbb{A} \wedge k \neq x$). If $\mathcal{P}_k = \emptyset$, processor $\mathcal{P}_x$ is preferable.

If τ_i is allocated to $\mathcal{P}_x$, we have that the estimated load of $\mathcal{P}_x$ is $L_x^{sc,est}(\Psi_x \cup \{\tau_i\})$, that of $\mathcal{P}_k$ is $L_k^{sc,est}(\Psi_k)$ ($L_k^{sc,est}(\Psi_k) \leq L_x^{sc,est}(\Psi_x) \leq L_y^{sc,est}(\Psi_y)$ from Equations (21b) and (22a)), and the maximum estimated load of other processors is $L_y^{sc,est}(\Psi_y)$. Thus, we have

$$L_{max}^{sc,est} = \max\{L_x^{sc,est}(\Psi_x \cup \{\tau_i\}), L_y^{sc,est}(\Psi_y)\} \tag{24}$$

$$\begin{aligned} L_{min}^{sc,est} = &\min\{L_x^{sc,est}(\Psi_x \cup \{\tau_i\}), L_k^{sc,est}(\Psi_k), \\ &\min\{L_m^{sc,est}(\Psi_m) | \forall m : m \neq x \wedge m \neq k\}\} \end{aligned} \tag{25}$$

Otherwise, the estimated load of $\mathcal{P}_k$ is $L_k^{sc,est}(\Psi_k \cup \{\tau_i\}) = L_x^{sc,est}(\Psi_x \cup \{\tau_i\})$ from the definition of $\mathcal{A}$, that of $\mathcal{P}_x$ is $L_x^{sc,est}(\Psi_x) \leq L_y^{sc,est}(\Psi_y)$ from Equation (22a), and the maximum estimated load of other processors is $L_y^{sc,est}(\Psi_y)$. Thus, it leads to the same $L_{max}^{sc,est}$ as Equation (24) and

$$\begin{aligned} L_{min}^{sc,est} = &\min\{L_x^{sc,est}(\Psi_x \cup \{\tau_i\}), L_x^{sc,est}(\Psi_x), \\ &\min\{L_m^{sc,est}(\Psi_m) | \forall m : m \neq x \wedge m \neq k\}\} \end{aligned} \tag{26}$$

As $L_k^{sc,est}(\Psi_k) \leq L_x^{sc,est}(\Psi_x)$ holds true from Equation (21b), $\mathcal{P}_x$ is chosen due to lower or the same $L_{min}^{sc,est}$ from Equations (25) and (26).

Case 2: Second, we consider processor $\mathcal{P}_y$ and a processor $\mathcal{P}_k$ ($\mathcal{P}_k \in \mathbb{B} \wedge k \neq y$). If $\mathcal{P}_k = \emptyset$, processor $\mathcal{P}_y$ is preferable.

If τ_i is allocated to $\mathcal{P}_y$, we can have that the estimated load of $\mathcal{P}_y$ is $L_y^{sc,est}(\Psi_y \cup \{\tau_i\})$, that of $\mathcal{P}_k$ is $L_k^{sc,est}(\Psi_k)$ (i.e., $L_y^{sc,est}(\Psi_y)$ from the definition of $\mathbb{B}$) and the maximum estimated load of other processors is no higher than $L_y^{sc,est}(\Psi_y)$ from Equation (22a). Thus, we have

$$L_{max}^{sc,est} = \max\{L_y^{sc,est}(\Psi_y \cup \{\tau_i\}), L_y^{sc,est}(\Psi_y)\} \tag{27}$$

$$\begin{aligned} L_{min}^{sc,est} = &\min\{L_y^{sc,est}(\Psi_y \cup \{\tau_i\}), L_y^{sc,est}(\Psi_y), \\ &\min\{L_m^{sc,est}(\Psi_m) | \forall m : m \neq y \wedge m \neq k\}\} \\ = &\min\{L_y^{sc,est}(\Psi_y \cup \{\tau_i\}), \min\{L_m^{sc,est}(\Psi_m) | \\ &\forall m : m \neq y \wedge m \neq k\}\} \end{aligned} \tag{28}$$

Otherwise, the estimated load of $\mathcal{P}_k$ is $L_k^{sc,est}(\Psi_k \cup \{\tau_i\})$, that of $\mathcal{P}_y$ is $L_y^{sc,est}(\Psi_y)$, and the maximum estimated load of other processors is no higher than $L_y^{sc,est}(\Psi_y)$. Note that, $L_y^{sc,est}(\Psi_y \cup \{\tau_i\}) \leq L_k^{sc,est}(\Psi_k \cup \{\tau_i\})$ holds true from Equation (22b). Thus, we can obtain

$$\begin{aligned} L_{max}^{sc,est} = &\max\{L_k^{sc,est}(\Psi_k \cup \{\tau_i\}), L_y^{sc,est}(\Psi_y)\} \\ \geq &\max\{L_y^{sc,est}(\Psi_y \cup \{\tau_i\}), L_y^{sc,est}(\Psi_y)\} \end{aligned} \tag{29}$$

$$\begin{aligned}L_{min}^{sc,est} =& \min\{L_k^{sc,est}(\Psi_k \cup \{\tau_i\}), L_y^{sc,est}(\Psi_y),\\ &\min\{L_m^{sc,est}(\Psi_m)|\forall m: m \neq y \wedge m \neq k\}\}\\ =& \min\{L_k^{sc,est}(\Psi_k \cup \{\tau_i\}), \min\{L_m^{sc,est}(\Psi_m)|\\ &\forall m: m \neq y \wedge m \neq k\}\}\\ \geq& \min\{L_y^{sc,est}(\Psi_y \cup \{\tau_i\}), \min\{L_m^{sc,est}(\Psi_m)|\\ &\forall m: m \neq y \wedge m \neq k\}\}\end{aligned} \tag{30}$$

Therefore, the processor $\mathcal{P}_y$ is chosen from Equations (27) to (30).

Case 3: Third, we only consider processors $\mathcal{P}_x$ and $\mathcal{P}_y$. If $y = x$ or $\mathcal{P}_y \in \mathbb{A}$, processor $\mathcal{P}_x$ is preferable subject to Case 1. Otherwise, if $L_x^{sc,est}(\Psi_x \cup \{\tau_i\}) > L_y^{sc,est}(\Psi_y)$, $\mathcal{P}_x$ is chosen due to lower $L_{max}^{sc,est}$ from Equations (24), (27) and the definition of set $\mathbb{A}$.

Otherwise, when $L_x^{sc,est}(\Psi_x \cup \{\tau_i\}) \leq L_y^{sc,est}(\Psi_y)$, the allocation of τ_i to $\mathcal{P}_x$ can result in the same or lower $L_{max}^{sc,est}$ as the allocation of τ_i to $\mathcal{P}_y$ from Equations (24) and (27). More specifically, when $L_x^{sc,est}(\Psi_x \cup \{\tau_i\}) \leq \max\{L_z^{sc,est}(\Psi_z \cup \{\tau_i\})|\forall \mathcal{P}_z\} \leq L_y^{sc,est}(\Psi_y)$, the same $L_{max}^{sc,est}$ (i.e., $L_y^{sc,est}(\Psi_y)$) can be obtained from Equations (24) and (27).

In this case, if τ_i is allocated to processor $\mathcal{P}_x$, we can have

$$\begin{aligned}L_{min}^{sc,est} =& \min\{L_x^{sc,est}(\Psi_x \cup \{\tau_i\}), L_y^{sc,est}(\Psi_y),\\ &\min\{L_m^{sc,est}(\Psi_m)|\forall m: m \neq x \wedge m \neq y\}\}\\ =& \min\{L_x^{sc,est}(\Psi_x \cup \{\tau_i\}), \min\{L_m^{sc,est}(\Psi_m)|\\ &\forall m: m \neq x \wedge m \neq y\}\}\end{aligned} \tag{31}$$

Similarly, if τ_i is allocated to processor $\mathcal{P}_y$, there is

$$\begin{aligned}L_{min}^{sc,est} =& \min\{L_y^{sc,est}(\Psi_y \cup \{\tau_i\}), L_x^{sc,est}(\Psi_x),\\ &\min\{L_m^{sc,est}(\Psi_m)|\forall m: m \neq x \wedge m \neq y\}\}\end{aligned} \tag{32}$$

- If $L_x^{sc,est}(\Psi_x \cup \{\tau_i\}) \leq L_x^{sc,est}(\Psi_x)$, as $L_x^{sc,est}(\Psi_x \cup \{\tau_i\}) < L_y^{sc,est}(\Psi_y \cup \{\tau_i\})$ holds from Equation (21a), processor $\mathcal{P}_x$ is chosen due to lower or the same $L_{min}^{sc,est}$ from Equations (31) and (32).
- Otherwise, when $L_x^{sc,est}(\Psi_x) < L_x^{sc,est}(\Psi_x \cup \{\tau_i\}) \leq \max\{L_z^{sc,est}(\Psi_z \cup \{\tau_i\})|\forall \mathcal{P}_z\} \leq L_y^{sc,est}(\Psi_y)$ (see Equation (23)), processor $\mathcal{P}_y$ is preferable from Equations (31) and (32).

Case 4: Finally, we consider tasks in set $\mathbb{C}$. If $\mathbb{C} = \emptyset$, $\mathcal{P}_x$ or $\mathcal{P}_y$ is chosen from above analysis. Otherwise, when τ_i is assigned onto a processor $\mathcal{P}_z$ ($\in \mathbb{C}$), the estimated load of $\mathcal{P}_z$ is $L_z^{sc,est}(\Psi_z \cup \{\tau_i\})$. The maximum estimated load of other processors is $L_y^{sc,est}(\Psi_y)$. Then, we can obtain

$$L_{max}^{sc,est} = \max\{L_z^{sc,est}(\Psi_z \cup \{\tau_i\}), L_y^{sc,est}(\Psi_y)\} \tag{33}$$

$$\begin{aligned}L_{min}^{sc,est} =& \min\{L_z^{sc,est}(\Psi_z \cup \{\tau_i\}), L_x^{sc,est}(\Psi_x), L_y^{sc,est}(\Psi_y),\\ &\min\{L_m^{sc,est}(\Psi_m)|\forall m: m \neq z \wedge m \neq x \wedge m \neq y\}\}\\ =& \min\{L_z^{sc,est}(\Psi_z \cup \{\tau_i\}), L_x^{sc,est}(\Psi_x),\\ &\min\{L_m^{sc,est}(\Psi_m)|\forall m: m \neq z \wedge m \neq x \wedge m \neq y\}\}\\ \geq& \min\{L_z^{sc,est}(\Psi_z \cup \{\tau_i\}), L_x^{sc,est}(\Psi_x),\\ &\min\{L_m^{sc,est}(\Psi_m)|\forall m: m \neq x \wedge m \neq y\}\}\end{aligned} \tag{34}$$

If $L_x^{sc,est}(\Psi_x \cup \{\tau_i\}) > L_y^{sc,est}(\Psi_y)$, processor $\mathcal{P}_x$ is chosen due to lower $L_{max}^{sc,est}$ from Equations (24), (33) and (21a). Otherwise, when $L_x^{sc,est}(\Psi_x \cup \{\tau_i\}) \leq L_y^{sc,est}(\Psi_y)$, the allocation of τ_i to $\mathcal{P}_x$ can result in the same or lower $L_{max}^{sc,est}$ as the allocation of τ_i to $\mathcal{P}_z$. More specifically, when $L_x^{sc,est}(\Psi_x \cup \{\tau_i\}) \leq \max\{L_m^{sc,est}(\Psi_m \cup \{\tau_i\})|\forall \mathcal{P}_m\} \leq L_y^{sc,est}(\Psi_y)$, the same $L_{max}^{sc,est}$ (i.e., $L_y^{sc,est}(\Psi_y)$) can be obtained from Equations (24) and (33).

In this case, as $L_z^{sc,est}(\Psi_z \cup \{\tau_i\}) > L_x^{sc,est}(\Psi_x \cup \{\tau_i\})$ from the definition of set $\mathbb{A}$, Equation (34) can be transformed as

$$\begin{aligned}L_{min}^{sc,est} \geq& \min\{L_x^{sc,est}(\Psi_x \cup \{\tau_i\}), L_x^{sc,est}(\Psi_x),\\ &\min\{L_m^{sc,est}(\Psi_m)|\forall m: m \neq x \wedge m \neq y\}\}\end{aligned} \tag{35}$$

- If $L_x^{sc,est}(\Psi_x \cup \{\tau_i\}) \leq L_x^{sc,est}(\Psi_x)$, processor $\mathcal{P}_x$ is chosen due to lower or the same $L_{min}^{sc,est}$ from Equations (31) and (35).
- Otherwise, when $L_x^{sc,est}(\Psi_x) < L_x^{sc,est}(\Psi_x \cup \{\tau_i\}) \leq \max\{L_m^{sc,est}(\Psi_m \cup \{\tau_i\})|\forall \mathcal{P}_m\} \leq L_y^{sc,est}(\Psi_y)$ (i.e., Equation (23)), Equation (35) can be transformed as

$$\begin{aligned}L_{min}^{sc,est} \geq& \min\{L_x^{sc,est}(\Psi_x), \min\{L_m^{sc,est}(\Psi_m)|\\ &\forall m: m \neq x \wedge m \neq y\}\}\end{aligned} \tag{36}$$

Moreover, as $L_x^{sc,est}(\Psi_x \cup \{\tau_i\} \leq L_y^{sc,est}(\Psi_y \cup \{\tau_i\})$ from the definition of set $\mathbb{A}$, Equation (32) can be transformed as

$$\begin{aligned}L_{min}^{sc,est} =& \min\{L_x^{sc,est}(\Psi_x), \min\{L_m^{sc,est}(\Psi_m)|\\ &\forall m: m \neq x \wedge m \neq y\}\}\end{aligned} \tag{37}$$

Thus, processor $\mathcal{P}_y$ is preferable from Equations (36) and (37).

In summary, the claim holds true. □

Time Complexity of SC-TMA

Recall that N is the number of tasks and M is the number of available processors in the system. Assume that the number of critical sections per task is taken to be a constant [24], [25] and that $M \leq N$.

From Proposition 2 and Algorithm 3, finding out the estimated maximum total waiting time for an un-mapped task to access its one resource can be done in $O(N)$ time. Thus, adjusting the estimated synchronization overhead for all unallocated tasks can be done in $O(N^2)$ time, and the complexity of finding the highest priority task to be mapped next can be found as $O(N^2)$ (see Line 5 in Algorithm 2).

Obtaining $U_k^{sc,est}(\Psi_k \cup \{\tau_i\})$ for each processor $\mathcal{P}_k$ requires at most $|\Psi_k|$ time from Algorithm 5. As at most $(N-1)$ tasks have been assigned when mapping a task to a processor, performing Algorithm 5 for all processors requires $O(N)$ time. Also, the complexity of repeating Algorithm 4 for all processors can be done in $O(N)$ time. Moreover, finding out the appropriate processor for a task can be done in $O(M)$ time from Equations (21) to (23). Thus, a task allocation (see Line 6 in Algorithm 2) can be done in $O(N)$ time. Similarly, updating the synchronization overhead for all allocated tasks can be done in $O(N^2)$ time (see Line 10 Algorithm 2).

Therefore, the complexity of SC-TMA-Quick can be determined as $O(M \cdot N^3)$ that approximates to that of SA-WFD ($O(\max\{M, log(N)\} \cdot N^2)$) [24]. Similarly, the complexity of SC-TMA with probe-based mapping policy (i.e., SC-TMA-Probe) can be found as $O(M^2 \cdot N^3)$.

TABLE 2: The timing parameters of tasks in the example

	c_i	p_i	$\Re_i$	$\{c_{i,x} \mid \forall r_{i,x} \neq 0\}$	$\mathcal{S}_{i,1}$	$\mathcal{S}_{i,2}$	$BW_i^{max}(\mathcal{R}_1)$	$BW_i^{max}(\mathcal{R}_2)$	BW_i^{max}	$u_i^{sc,estimate}$
τ_1	1	10	$\{\mathcal{R}_1\}$	{0.5}	$\{s_{1,2}\}$	$\emptyset$	4	0	4	0.5
τ_2	1	10	$\{\mathcal{R}_2\}$	{0.5}	$\emptyset$	$\{s_{2,1}\}$	0	3.5	3.5	0.45
τ_3	3	20	$\{\mathcal{R}_1, \mathcal{R}_2\}$	{1,1}	$\{s_{3,2}\}$	$\{s_{3,4}\}$	4	3.5	7.5	0.525
τ_4	9	30	$\{\mathcal{R}_1\}$	{1,1,1.5,2}	$\{s_{4,2}, s_{4,4}, s_{4,6}, s_{4,8}\}$	$\emptyset$	7.5	0	7.5	0.55
τ_5	10	30	$\{\mathcal{R}_1, \mathcal{R}_2\}$	{1,1.5,2,2}	$\{s_{5,2}, z_{5,8}\}$	$\{s_{5,4}, s_{5,6}\}$	5.5	3	8.5	0.617

TABLE 3: The varying synchronization overhead during mapping process

(a) The allocation of task τ_5 to processor $\mathcal{P}_1$

	$BW_i^{max}(\mathcal{R}_1)$	$BW_i^{max}(\mathcal{R}_2)$	BW_i^{max}	$u_i^{sc,estimate}$	BW_i	B_i
τ_1	4	0	4	0.5	-	-
τ_2	0	3	3	0.4	-	-
τ_3	4	2.5	6.5	0.475	-	-
τ_4	7.5	0	7.5	0.55	-	-
τ_5	-	-	-	-	0	0

(b) The allocation of task τ_4 to processor $\mathcal{P}_2$

	$BW_i^{max}(\mathcal{R}_1)$	$BW_i^{max}(\mathcal{R}_2)$	BW_i^{max}	$u_i^{sc,estimate}$	BW_i	B_i
τ_1	4	0	4	0.5	-	-
τ_2	0	3	3	0.4	-	-
τ_3	4	2.5	6.5	0.475	-	-
τ_4	-	-	-	-	3	0
τ_5	-	-	-	-	3.5	0

(c) The allocation of task τ_1 to processor $\mathcal{P}_3$

	$BW_i^{max}(\mathcal{R}_1)$	$BW_i^{max}(\mathcal{R}_2)$	BW_i^{max}	$u_i^{sc,estimate}$	BW_i	B_i
τ_1	-	-	-	-	4	0
τ_2	0	3	3	0.4	-	-
τ_3	4	2.5	6.5	0.475	-	-
τ_4	-	-	-	-	4.5	0
τ_5	-	-	-	-	4.5	0

(d) The allocation of task τ_3 to processor $\mathcal{P}_2$

	$BW_i^{max}(\mathcal{R}_1)$	$BW_i^{max}(\mathcal{R}_2)$	BW_i^{max}	$u_i^{sc,estimate}$	BW_i	B_i
τ_1	-	-	-	-	4	0
τ_2	0	3	3	0.4	-	-
τ_3	-	-	-	-	4.5	4.5
τ_4	-	-	-	-	4.5	0
τ_5	-	-	-	-	6.5	0

(e) The allocation of task τ_2 to processor $\mathcal{P}_3$

	$BW_i^{max}(\mathcal{R}_1)$	$BW_i^{max}(\mathcal{R}_2)$	BW_i^{max}	$u_i^{sc,estimate}$	BW_i	B_i
τ_1	-	-	-	-	4	0
τ_2	-	-	-	-	3	0
τ_3	-	-	-	-	5	4.5
τ_4	-	-	-	-	4.5	0
τ_5	-	-	-	-	7.5	0

An Example for SC-TMA

Consider the example in Section 3.1 with task and resource patterns shown in Figure 1, we further illustrate how our SC-TMA-Quick works. Initially, there is no task on any processor ($\Psi_k = \emptyset, k = 1, 2, 3$) and the timing parameters of tasks related to synchronization are given in Table 2. We have $\Phi = \{\tau_5, \tau_4, \tau_3, \tau_1, \tau_2\}$ in non-increasing order of estimated synchronization-cognizant utilizations.

First, task τ_5 is allocated to processor $\mathcal{P}_1$ (i.e., $\Psi_1 = \{\tau_5\}$) and the estimated maximum global waiting times of unassigned tasks for resources that are obtained from Algorithm 3 are shown in Table 3(a). Moreover, the synchronization overheads of assigned tasks (i.e., τ_5) are recalculated as listed in Table 3(a) as well and we have $\Phi = \{\tau_4, \tau_3, \tau_1, \tau_2\}$.

Then, task τ_4 is chosen to be allocated. We can obtain $B_5^{est} = 0$ from Equation (20) and then $L_1^{sc,est}\{\Psi_1 \cup \{\tau_4\}\} = \max\{\frac{10}{30}, \{\frac{10}{30} + \frac{9+7.5}{30}\}\} = 0.883$ from Algorithm 5; $BW_5^{est} = 4$ from Equation (17) and then $L_1^{sc,est}\{\Psi_1\} = \frac{10+4}{30} = 0.467$ from Algorithm 4. Similarly, we have $L_2^{sc,est}\{\Psi_2 \cup \{\tau_4\}\} = L_3^{sc,est}\{\Psi_3 \cup \{\tau_4\}\} = 0.55$ and $L_2^{sc,est}\{\Psi_2\} = L_3^{sc,est}\{\Psi_3\} = 0$. We can obtain that $\mathcal{P}_x = \mathcal{P}_2$ from Equation (21) and $\mathcal{P}_y = \mathcal{P}_1$ from Equation (22). Therefore, task τ_4 will be allocated to processor $\mathcal{P}_2$ and the adjustment of tasks' synchronization overhead is given in Table 3(b).

Instead of task τ_3 as the original candidate, task τ_1 is chosen to be mapped next due to τ_3's reduced estimated maximum global waiting time (from initial 7.5 to 6.5). Following the same steps (see Tables 3(c) to 3(e)), the final task-to-processor mapping can be obtained as: $\Psi_1 = \{\tau_5\}$, $\Psi_2 = \{\tau_4, \tau_3\}$ and $\Psi_3 = \{\tau_1, \tau_2\}$. Then, from Equations (1-4) and (14), we can get the loads on the processors as $L_1^{sc} = 0.5833$, $L_2^{sc} = 0.85$ and $L_3^{sc} = 0.9$, respectively. Finally, the system load can be determined as $L^{sc} = \max_{j=1}^{3}\{L_j^{sc}\} = 0.9$.

Next, we consider reducing the number of processors to two, the result mapping of tasks to processors is $\Psi_1 = \{\tau_1, \tau_3, \tau_4, \tau_5, \tau_2\}$ and $\Psi_2 = \emptyset$ (the details are omitted here), and thus we can have $L^{sc} = 0.9833$. When the number of processors is reduced to one, we can get $L^{sc} = 0.9833$ as well. Therefore, the result L^{sc} is 0.9 under SC-TMA-Quick.

Furthermore, assuming that three processors are used, the task partitioning for SC-TMA-Probe is $\Psi_1 = \{\tau_5, \tau_3, \tau_2\}$ and $\Psi_2 = \{\tau_4, \tau_1\}$, which results in $L_1^{sc} = 0.8$, $L_2^{sc} = 0.7667$ and $L^{sc} = 0.8$. The same result can be obtained when only two processors are considered. If only one processor is used, the result system load is 0.9883. Finally, we can have $L^{sc} = 0.8$ under SC-TMA-Probe.

For comparison, the final partitioning of tasks to processors subject to conventional WFD is $\Psi_1 = \{\tau_5\}$, $\Psi_2 = \{\tau_4\}$, $\Psi_3 = \{\tau_1, \tau_2, \tau_3\}$ and thus we have $L^{sc}(\Pi) = 1.3$, which results in un-schedulable task mapping; while that of SA-WFD [24] is $\Psi_1 = \{\tau_5\}$, $\Psi_2 = \{\tau_4, \tau_2\}$ and $\Psi_3 = \{\tau_1, \tau_3\}$ where the result L^{sc} is 1. Moreover, the result mapping of tasks to processors under BPA [34] is $\Psi_1 = \{\tau_1, \tau_3, \tau_4, \tau_5, \tau_2\}$ and $\Psi_2 = \emptyset$ and thus we have $L^{sc} = 0.9833$. Here, SC-TMA obtains better task partitioning when comparing with WFD (due to schedulability of tasks), SA-WFD and BPA (due to lower system load). Further, in Section 7, we can see that SC-TMA shows better performance compared to these mapping algorithms, with respect to schedulability ratio of tasks and (average) system load.