Overhead-Aware Temporal Partitioning on Multicore Processors

Risat Mahmud Pathan, Per Stenström  
Chalmers University of Technology, Sweden  
Email: {risat, per.stenstrom}@chalmers.se

Lars-Göran Green, Torbjörn Hult, Patrik Sandin  
RUAG Space Sweden AB, Göteborg, Sweden  
{lars-goran.green, torbjorn.hult, patrik.sandin}@ruag.com

Abstract—Many previously proposed interface models for composability analysis of hierarchical scheduling are overhead-unaware, which is unsafe for real systems. This paper proposes an overhead-aware schedulability analysis to guarantee temporal partitioning among real-time applications/components hosted on a multicore platform.

First, a new interface model and the method to generate an interface for a given component is proposed. Each interface has a tunable parameter $d$ (degree) that can balance between abstraction and accuracy in capturing each component’s task-level timing constraints. Second, the problem of constructing an overhead-aware system-level schedule of all the components is addressed. The system designer has the flexibility to select parameters (e.g., allocated processor bandwidth) for each component such that overhead (e.g., preemptions across partitions) is minimized. Third, a slack distribution algorithm to reduce various overhead is proposed and its effectiveness is evaluated using randomly generated interface sets and also using workload of a real space-borne application provided by RUAG Space Sweden AB.

I. INTRODUCTION

Multicore processors is one of the major enablers for many safety-critical to consider composing multiple applications/components on a single, powerful computing platform to reduce cost. The design of such integrated systems often needs to follow certain certification standard, e.g., ARINC-653 for integrated modular avionics (IMA) and AUTOSAR framework for automotive. Recently, there has been growing interest to also investigate ARINC-653 applied to space-borne systems. The work in this paper is done in joint collaboration between RUAG Space AB and Chalmers University of Technology in Sweden to study the potential of hierarchical real-time scheduling on multicores for space applications based on ARINC-653.

The concept of temporal partitioning is at the heart of scheduling principle of ARINC-653. Temporal partitioning ensures that multiple applications hosted on the same platform must not interfere with one another in the time domain. The hardware support to implement temporal partitioning is operational in different architecture, e.g., ARM MPCore. Temporal partitioning can be implemented on such architecture based on multiprocessor hierarchical scheduling (MHS) where each application has a partition server with a (dedicated) fraction of processors’ capacity. The servers of different applications are scheduled by a system-level scheduler while the tasks of each application are scheduled by a task-level scheduler — but only when its server is active to ensure temporal partitioning.

Context of this research. Consider a set of $N$ applications to be scheduled on a multicore processor having $m$ cores. Each application has $n'$ partition servers that will be allocated to $n'$ cores. Each of the servers has an execution capacity and a replenishment period on the processor core to which it will be allocated. The period and capacity of each server respectively specifies how much and how frequently the capacity allocated by some core is given to the corresponding application.

The tasks of a component are scheduled based on preemptive global fixed-priority (G-FP) scheduling on $n'$ servers. The allocated servers in each core are used to construct the system-level schedule, which is a static cyclic schedule of the servers in that core. Although G-FP is assumed as the task-level scheduler, our approach can be applied for any work-conserving global scheduler, for example, global EDF. The choice of fixed-priority (FP) scheduling at the task level and static cyclic scheduling at the system level are compliant with the OS executive of ARINC-653.

Hierarchical scheduling has the advantage that the timing constraints can be analyzed by the supplier of each component without any concern regarding the analysis of other components. The supplier can simply deliver the component to the system designer as a black box labeled with an interface that abstracts/hides the internal (task-level) details of the component. The system designer can then compose all the $N$ components only based on their interfaces. The interface of a component provides “separation of concern” between the component’s supplier and system designer. Such separation of concern between the component suppliers and system designer is vital in the industries, e.g., automotive, aviation, and space.

Related works. Many of the existing MHS techniques have proposed virtual-platform-based interface model to abstract each component. A virtual platform (VP) specifies a component’s required processing capacity allocated in at most $n'$ virtual processors. The VPs are mapped later to physical processors by the system designer. Different interesting VP-based interface models have been proposed to restrict the parallelism of the tasks inside a component to at most $n'$ (e.g., multiprocessor periodic resource (MPR) model, multi-supply functions (MSF), parallel-supply function (PSF) (1), bounded-delay multipartition (BDM), deferrable server (DS) (22)). We argue below that the existing VP-based interface

1The terms “application” and “component” are used interchangeably.
models do not provide “good” separation of concerns between the supplier and system designer with respect to all the following properties:

- There must be a way to generate the interface for a given component. The works in [3], [4], [22] do not address the problem of generating an interface.
- An interface generation should not depend on any composition-related parameter which needs to be provided by the system designer. This is because the value of that parameter may not be known to the system designer without knowing the interfaces of all the components (circular-dependency problem). The works in [20], [7], [17] assume that a composition-dependent parameter, i.e., period parameter (P) used in [20], [7] and the delay parameter (Δ) used in [17], is provided by the system designer for generating the MPR and BDM interface, respectively.
- An interface should be flexible in the sense that it allows the system designer to choose parameters (e.g., capacity, period) for the servers, for example, to minimize overall processing bandwidth. While the BDM interface [17] provides such flexibility during allocation of VPs to physical processors, the MRP interface [20], [7] is quite rigid and does not provide such flexibility.
- There must be a way to generate and analyze the system-level scheduling for a given set of components’ interfaces. Such analysis is not explicitly addressed in [5], [4], [16], [22]. Moreover, the overheads (e.g., partition preemptions) must be taken into account during the system-level schedulability analysis. Although one can rely on the works in [8], [12] for task-level overhead-aware schedulability analysis, none of the MHS works in [20], [5], [4], [17], [16], [22] addressed system-level overhead-aware schedulability analysis.

Contributions. We have the following contributions. First, a new interface to model the resource requirement of a component is proposed. The interface captures the period-capacity relationship of m′ servers as a collection of d points in the Cartesian plane. The tunable parameter d, called the degree of the interface, can be used to balance the level of abstraction and accuracy. The method to generate an interface for a given component is proposed (called, the folding operation) and does not require any composition-dependent parameter to be provided by the system designer. The interface is flexible: given some capacity, the upper bound on server’s period can be determined; and given some period, the lower bound on capacity can be computed (called, interface unfolding operation).

Second, the construction of system-level static schedule for a set of N components based on their interfaces is presented. We applied the first-fit (FF) heuristic to allocate the servers to physical cores. A system-level overhead-aware schedulability analysis by considering various run-time overhead is proposed. A metric, called Normalized OVErhead (NOVE), is proposed to compute the system-level scheduling overhead with respect to the length of the static schedule. The value of NOVE is used to compare different possible system-level schedules and to choose the one that minimizes NOVE.

Third, a slack-distribution algorithm to reduce overhead (i.e., NOVE) is proposed. The principle behind the design of this algorithm is that rather than wasting processing bandwidth in managing overhead, we can allocate additional capacity to adjust the periods of the servers (thanks to the flexibility of interface) for reducing the run-time overhead to the extent so that overall bandwidth is saved.

Finally, we have conducted extensive simulations based on randomly generated interface sets. We observed that (i) overhead-unaware schedulability analysis is unsafe, (ii) selecting the right periods of the servers can reduce overhead, and (iii) the slack-distribution algorithm reduces the run-time overhead and can save processing bandwidth significantly. We have also applied our proposed technique to a real-world workload of space application provided by RUAG Space Sweden AB. The proposed slack-distribution algorithm can find the minimum number of physical processors required for allocating the servers of the space applications.

Organization. The remainder of this paper is organized as follows: Section II proposes the system model and notations. Section III presents the folding operation and unfolding operation of an interface. The method to generate the system-level scheduling without considering overhead is presented in Section IV. Then, Section V presents the overhead-aware analysis for scheduling partition servers at the system-level. Section VI presents the slack-distribution algorithm used to reduce system-level overheads. Experimental results and the real-world case study are presented in Section VII and Section VIII. Finally, we conclude the paper in Section IX.

II. System Model and Notations

The applications A_1, ..., A_N are to be scheduled on a multicore platform. Each application A_k is a collection of n_k constrained-deadline sporadic tasks τ_1, ..., τ_n_k. Each task τ_i ∈ A_k is characterized by a triple (C_i, D_i, T_i), where C_i represents the worst-case execution time (WCET), D_i is the relative deadline, and T_i is the minimum inter-arrival time of the jobs of task τ_i such that T_i ≥ D_i ≥ C_i. The set of higher priority tasks of τ_i ∈ A_k is denoted by hp(i).

The interface of application A_k is denoted as I_k = (m_k′, P_1, P_2, ..., P_d) where m_k′ is the number of servers that will be allocated in m_k′ cores. Each P_ν = (x_ν, y_ν) in I_k is a point in the first quadrant of the Cartesian plane that can represent many different period-capacity relationship of the m_k′ servers of A_k. The degree d of an interface I_k is d = |{P_1, P_2, ..., P_d}|. The interface I_k is generated by the component supplier while the system designer uses interface I_k to determine the
processing capacity and the period for each of the $m'_k$ servers. The $m'_k$ servers of component $A_k$ is collectively denoted as partition $S_k$. We abuse the notation $S_k$ also to say “server $S_k$” to denote the server of partition $S_k$ when the allocation of that server in some core is known.

If a server that is allocated on some core has capacity $\xi_k$ and period $\Delta_k$, then $(\Delta_k \cdot \xi_k)$ time units is allocated to partition $S_k$ in every period $\Delta_k$ in that core. An upper bound on the delay in allocating capacity $\xi_k$ to server $S_k$ allocated in some core is at most $(1 - \xi_k) \cdot \Delta_k$. The delay not more than $(1 - \xi_k) \cdot \Delta_k$ is ensured in system-level scheduling by allocating the capacity in the core the same way in every period (as is shown in Figure II) based on distance-constrained scheduling [13].

![Figure 1. The capacity in different cores may be allocated differently. However, the capacity $\xi$ in a given core is allocated the same way for all the periods of the server. This guarantees that given any $t$, the amount of execution time available during $[t, t + \Delta]$ in that core is $(\xi - \Delta)$.](image)

Each application $A_k$ executes its tasks using G-FP scheduling when partition $S_k$ in some core is active (i.e., a server of $S_k$ is scheduled in that core by the system-level scheduler). When a server $S_k$ in some core becomes inactive, the task of $A_k$ running on that core is preempted and can migrate to a different (active) server that is allocated to a different core or may resume execution later on the same core when the server is reactivated.

### III. INTERFACE GENERATION

In this section, we first present the G-FP schedulability analysis of an application $A_k$ in order to determine the relationship between the capacity $\xi_k$ and period $\Delta_k$ by assuming that each of the $m'_k$ servers for application $A_k$ has capacity $\xi_k$ and period $\Delta_k$. Then, the method to generate the interface of a given component $A_k$ (the folding operation) that abstracts the internal details of this relationship is presented. Finally, we describe how the interface can be used (unfolding operation) by the system designer to select the server’s capacity and period. Moreover, we relax the assumption that each of the $m'_k$ servers has the same capacity and period in Section [VI] where the slack-distribution algorithm is proposed.

#### A. The Relationship between Capacity and Period

There exist many G-FP schedulability tests considering $m$ (dedicated) identical speed-1 processors (a recent survey can be found in [10]). One such schedulability test, called the $DA-LC$ test, is proposed by Davis and Burns [11] that works as follows: given a task $\tau_i \in A_k$, an upper bound on total workload, denoted by $W_i$, of the tasks in $A_k$ in an interval of length $D_i$ is computed. And, the upper bound on interference due to the tasks in $\text{hp}(i)$ on any job of task $\tau_i$ is $W_i / m$. We denote $L_i$ as $L_i = W_i / m + C_i$. The details of computing $L_i$ using the $DA-LC$ test, considering $m$ speed-1 processors, can be found in [11] and is not critical to understand the rest of this paper.

The $DA-LC$ test [11] essentially checks for each task $\tau_i$ whether $L_i \leq D_i$ holds or not. The $DA-LC$ test runs in polynomial time, and therefore, the value of minimum $m$, denoted by $m'_k$, for which $L_i = (W_i / m + C_i) \leq D_i$ can be determined very efficiently. It follows directly from the $DA-LC$ test that application $A_k$ is G-FP schedulable on $m'_k$ dedicated speed-$\xi_k$ processors if

$$\max_{i=1\ldots n_k} \left\{ \frac{L_i \cdot \Delta_k}{D_i} \right\} = \max_{i=1\ldots n_k} \left\{ \frac{W_i / m'_k + C_i}{D_i} \right\} \leq \xi_k \quad (1)$$

where $m'_k$ is the minimum number of processors that satisfies $L_i = (W_i / m'_k + C_i) \leq D_i$.

**Theorem 1.** Application $A_k$ is G-FP schedulable in partition $S_k$ having $m'_k$ servers, each with capacity $\xi_k$ and period $\Delta_k$, if the following two conditions are satisfied:

(C1) Capacity is allocated in distance-constrained manner

$$\frac{1}{\xi_k \cdot (1 - \xi_k)} \min_{j=1\ldots n_k} \left\{ \xi_k \cdot \Delta_j - L_j \right\} \geq \Delta_k \quad (2)$$

where $L_j$ is computed using the $DA-LC$ test considering $m'_k$ speed-1 processors.

**Proof:** It follows from Eq. (2) of condition C2 that

$$\min_{j=1\ldots n_k} \left\{ D_j - L_j / \xi_k \right\} \geq (1 - \xi_k) \cdot \Delta_k$$

And, this implies that $L_i / (D_i - (1 - \xi_k) \cdot \Delta_k) \leq \xi_k$ for any task $\tau_i \in A_k$. Consequently, it is guaranteed based on Eq. (1) that each job of $\tau_i$ completes $(1 - \xi_k) \cdot \Delta_k$ time units earlier than its deadline on a dedicated $m'_k$ speed-$\xi_k$ processor platform.

Consider that a job of task $\tau_i$ released at time $r$. Since distance-constrained scheduling [15] is assumed for each of the $m'_k$ servers with capacity $\xi_k$ and period $\Delta_k$ (condition C1), the maximum delay in allocating capacity to partition $S_k$ by each of the servers is $(1 - \xi_k) \cdot \Delta_k$. The capacity allocated to application $A_k$ in $[r, r + D_i]$ is minimized if each processor allocates no capacity to $S_k$ in $[r, r + (1 - \xi_k) \cdot \Delta_k)$ and allocates $(\xi_k \cdot \Delta_k)$ time units in every interval of length $\Delta_k$ starting from time $r + (1 - \xi_k) \cdot \Delta_k$. However, the capacity under such worst-case is allocated as quickly as possible to partition $S_k$ from time instant $r + \Delta_k \cdot (1 - \xi_k)$ to satisfy the distance-constrained property. Therefore, the execution of $A_k$’s tasks is no worse than that of on a dedicated $m'_k$ speed-$\xi_k$ processor platform in the interval $[r + (1 - \xi_k) \cdot \Delta_k, D_i]$. Since $L_i / (D_i - (1 - \xi_k) \cdot \Delta_k) \leq \xi_k$ holds from condition C2, task $\tau_i$ meets its deadline in partition $S_k$ where each of the $m'_k$ servers has capacity $\xi_k$ and period $\Delta_k$. ■

This relationship between $\xi_k$ and $\Delta_k$ in Eq. (2) is used
to generate $A_k$’s interface. To avoid the case $\Delta_k \leq 0$, the minimum capacity, denoted by $\xi^\text{min}_k$, that can guarantee schedulability of $A_k$, is given (based on Eq. (1)) as follows:
\[
\xi^\text{min}_k = \max_{j=1,\ldots,n_k} \left\{ \frac{y_j \cdot L_j}{D_j} \right\} + \epsilon
\]
where $\epsilon > 0$ is a very small constant, for example $\epsilon = 0.001$. Moreover, to avoid the case $\Delta_k = \infty$, we assume $\xi_k \leq (1 - \epsilon)$. In summary, the range for the schedulable capacity of server $S_k$ is $[\xi^\text{min}_k, 1]$ and the corresponding range of period is $(0, \Delta_k]$ where $\Delta_k$ satisfies Eq. (2).

B. Interface Generation: Folding and Unfolding

In this subsection, we first present the folding operation, i.e., how the interface of a component is generated. Then, we present unfolding operation, i.e., how the system designer can retrieve the capacity and period relationship from the interface of a component.

**Definition 1** (Interface of $A_k$). An interface $\mathcal{I}_k$ of component $A_k$ is a 2-tuple given as follows
\[
\mathcal{I}_k = (m_k', \{P_1, P_2, \ldots P_{d_k}\}) = (m_k', \{P_i \equiv (x_i, y_i) \mid i = 1, \ldots, d_k\})
\]
where $m_k'$ is the number of servers required for partition $S_k$, $d$ is the degree of the interface where $d \geq 2$ and each $P_i$ is a point in the first quadrant of the Cartesian plane.

**Folding Operation.** Our objective is to compact Eq. (2) that guarantees that application $A_k$ is G-FP schedulable on $m_k'$ servers each with capacity $\xi_k$ and period $\Delta_k$. The challenge is to hide (abstract) all the task-related parameters (i.e., $D_j$ and $L_j$) that are on the left-hand side of Eq. (2).

The interface $\mathcal{I}_k$ with maximum degree, denoted by $d_{\text{max}}$, contains those points that are also the end points of the line segments that tightly lower bound the following equation:
\[
y = \min_{j=1,\ldots,n_k} \left\{ x \cdot D_j - L_j \right\}
\]
where $\xi_{\text{min}} \leq x \leq 1$. And, the line segments that tightly lower bound Eq. (4) are also the tight lower bound of the term $\min_{j=1,\ldots,n_k} \{\xi_k \cdot D_j - L_j\}$ which is present in Eq. (2).

If the line joining points $P_a = (x_a, y_a)$ and $P_b = (x_b, y_b)$ tightly lower bounds Eq. (4) where $x_a \leq x \leq x_b$, then $P_a$ and $P_b$ are in $\mathcal{I}_k$. By including all such points in $\mathcal{I}_k$, the interface of component $A_k$ with maximum degree $d_{\text{max}}$ is generated. Such end points are all the intersection points of the following linear equations in the xy-coordinate system where $\xi_{\text{min}} \leq x \leq 1$:
\[
x = \xi_{\text{min}} \\
y = x \cdot D_j - L_j \quad \text{for} \quad j = 1, \ldots, n_k \\
x = 1
\]
To find the points in the interface, at most $O((n_k)^2)$ pairs of linear equations in Eq. (5) have to be solved. Let $\mathcal{PT}$ denote the set of all intersection points of the lines given in Eq. (5). Not every line segment connecting points $(x_a, y_a)$ and $(x_b, y_b)$ in $\mathcal{PT}$ forms a lower bound of Eq. (4). We apply the following rules to filter the invalid points from $\mathcal{PT}$:

**Rule 1:** Since $\xi^\text{min}_k \leq x \leq 1$ for Eq. (4), point $P_a = (x_a, y_a) \in \mathcal{PT}$ is removed if either $x_a < \xi^\text{min}_k$ or $x_a > 1$. After applying this rule, each point $P_a \in \mathcal{PT}$ satisfies $\xi^\text{min}_k \leq x_a \leq 1$ and since $\xi_k^\text{min}$ satisfies Eq. (5), we also have $y_a > 0$.

**Rule 2:** $P_a \in \mathcal{PT}$ is removed if $y_a > (x_a \cdot D_j - L_j)$ for any $j = 1, 2, \ldots n_k$. This is because such a point $P_a = (x_a, y_a)$ cannot be on the line segment that forms the lower bound of Eq. (4). After applying this rule, the $\mathcal{PT}$ contains all points that are the end points of the line segments that tightly lower bound Eq. (4). We sort the points in $\mathcal{PT}$ in order of increasing $x$ values.

The points $P_1, P_2, \ldots P_{d_{\text{max}}}$ in $\mathcal{PT}$ are in interface $\mathcal{I}_k$ having maximum degree $d_{\text{max}}$. Note that $x_1 = \xi^\text{min}_k$ and $x_{d_{\text{max}}} = 1$. Since application $A_k$ can be assigned capacity within $[\xi^\text{min}_k, 1]$, an interface with degree $d < d_{\text{max}}$ contains at least two points $P_1$ and $P_{d_{\text{max}}}$. And, the remaining points are any of the $(d - 2)$ different points selected from set $(\mathcal{PT} - \{P_1, P_{d_{\text{max}}}\})$. For example, we can select $(d - 2)$ points from $(\mathcal{PT} - \{P_1, P_{d_{\text{max}}}\})$ having the smallest $x$ values to generate interface $\mathcal{I}_k = (m_k', \{P_1, P_2, \ldots P_{d_{\text{max}} - 1}, P_{d_{\text{max}}}\})$ with degree $d$. The value of $m_k'$ is the minimum number of dedicated speed-1 processor platform on which $A_k$ is G-FP schedulable based on the $\mathcal{DA-LC}$ test. Note that the line segments connecting consecutive points in $\mathcal{I}_k$ also forms the lower bound (may not be tight when $d < d_{\text{max}}$) of Eq. (4), and hence also lower bound for $\xi_k = \xi^\text{min}_k$ in Eq. (2). The following properties hold for any points $P_a, P_b, P_c$ at $\mathcal{I}_k$: 

**Property 1:** $x_a < x_b$ if and only if $y_a < y_b$ for $P_a, P_b \in \mathcal{I}_k$.

**Property 2:** $x_a < x_b < x_c$ if and only if $m_{a,b} > m_{b,c}$, where $m_{a,b}$ and $m_{b,c}$ are the slopes of the lines connecting points $\{P_a, P_b\}$ and $\{P_b, P_c\}$.

**Unfolding Operation.** Now we describe how a component’s interface $\mathcal{I}_k$ of degree $d$ can be unfolded by the system designer to determine the capacity $\xi_k$ and period $\Delta_k$ relationship. The minimum capacity $\xi^\text{min}_k$ for which a component with interface $\mathcal{I}_k$ is G-FP schedulable is $x_1$ where $(x_1, y_1) = P_1 \in \mathcal{I}_k$. Since $P_1 = (x_1, y_1)$ is the end point of the line segment that lower bounds $\min_{j=1,\ldots,n_k} \{\xi_k \cdot D_j - L_j\}$ in Eq. (2) for $\xi_k = x_1 = \xi^\text{min}_k$, the upper bound on period for given capacity $x_1$ is given, according to the G-FP schedulability condition in Eq. (2), as follows:
\[
\frac{1}{\xi_k \cdot (1 - \xi_k)} \cdot y_1 = \frac{y_1}{x_1 \cdot (1 - x_1)} \geq \Delta_k
\]
For example, if $(0.551, 0.2ms) = P_1 \in \mathcal{I}_k$, then $0.2ms / (0.551 \cdot (1 - 0.551)) = 808.4\mu s \geq \Delta_k$.

Now we demonstrate that given some capacity $\xi_k$ such that $x_1 = \xi^\text{min}_k < \xi_k \leq 1$, how the upper bound on period $\Delta_k$ can be determined from the interface. Given capacity $\xi_k$, the two points $P_a = (x_a, y_a)$ and $P_b = (x_b, y_b)$ at $\mathcal{I}_k$ such that $x_a \leq \xi_k \leq x_b$ are determined. Such two points
can always be found in \( I_k \) since \( x_1 = \xi_k^{\text{min}} \leq \xi_k \leq x_d = 1 \).
The line connecting points \( \{P_a, P_b\} \) is
\[ y = (y_b - y_a)/(x_b - x_a) \cdot (x - x_a) + y_a \tag{7} \]
This line in Eq. (7) is a lower bound of \( \min_{j=1 \ldots n_k} \{ \xi_k \cdot D_j - I_j \} \) in Eq. (2) whenever \( x_a \leq \xi_k \leq x_b \). By setting \( x = \xi_k \) in Eq. (2), the upper bound on the period \( \Delta_k \), according to the G-FP schedulability condition in Eq. (2), is given as follows:
\[ \frac{1}{\xi_k \cdot (1 - \xi_k)} \cdot [(y_b - y_a)/(x_b - x_a) \cdot (\xi_k - x_a) + y_a] \geq \Delta_k \tag{8} \]
Now we demonstrate that given some period \( \Delta_k \), how the lower bound on capacity \( \xi_k \) can be determined. Given period \( \Delta_k \), we find \( P_a \) and \( P_b \) in \( I_k \) such that \( x_a = \frac{y_a}{x_a \cdot (1 - x_a)} \leq \Delta_k \leq \frac{y_b}{x_b \cdot (1 - x_b)} \).
If such two points cannot be found in \( I_k \), then period \( \Delta_k \) is too large that some deadline of component \( A_k \) may be missed (i.e., schedulability condition in Eq. (2) is not satisfied). Then, the system designer has to consider a smaller period. Otherwise, when such two points are found in \( I_k \), the line segment connecting points \( \{P_a, P_b\} \) is given in Eq. (7). Now let \( \xi_s \) is the lower bound on capacity for the given period \( \Delta_k \). Then, to satisfy G-FP schedulability in Eq. (2) we must have
\[ \frac{1}{\xi_s \cdot (1 - \xi_s)} \cdot [(y_b - y_a)/(x_b - x_a) \cdot (\xi_s - x_a) + y_a] \geq \Delta_k \tag{9} \]
where \( m_{a,b} = (y_b - y_a)/(x_b - x_a) \). Eq. (9) is a quadratic inequality with unknown \( \xi_s \) which can be solved based on standard technique to solve quadratic equation as follows:
\[ \xi_s \geq \frac{-m_{a,b} - \Delta_k + \sqrt{(m_{a,b} - \Delta_k)^2 - 4 \cdot \Delta_k \cdot (y_b - m_{a,b} \cdot x_a)}}{2 \cdot \Delta_k} \]
Easwaran et al. in [14] also considered the space of (period, capacity) relationship based on the use of demand-bound function which is a step function along with the linear supply-bound function. In contrast, we use a collection of linear equations to exploit the \( (y, capacity) \) relationship where \( period = y/(capacity(1 - capacity)). \) Therefore, properties 1 and 2 may not hold for the (period, capacity) relationship as is explored in [14].

IV. CONSTRUCTING THE SYSTEM LEVEL SCHEDULE

This section solves the following problem: given a set of interfaces \( \mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_N \) for \( N \) components, how to generate the system-level distance-constrained static schedule for a given multicore platform. The steps to compute the static schedule are the following:

Step 1 (Find the minimum capacity): For an interface \( I_k = \{m'_k, (x_1 = e_k^{\text{min}}, y_1), (x_2, y_2), \ldots, (x_d = 1, y_d)\} \), the minimum capacity that guarantees G-FP schedulability is \( x_1 = \xi_k^{\text{min}} \). Therefore, the set of partition parameters that minimizes the total capacity requirement for \( N \) partitions is \( \{m'_1, e_1^{\text{min}}, \Delta_1\}, \{m'_2, e_2^{\text{min}}, \Delta_2\}, \ldots, \{m'_N, e_N^{\text{min}}, \Delta_N\}\) where \( \Delta_k \) is maximum period computed for each interface \( I_k \) using Eq. (6). The triple \( (m'_k, e_k^{\text{min}}, \Delta_k) \) represents the \( m'_k \) servers of partition \( S_k \) for \( k = 1, \ldots, N \). We denote \( \Delta_{\text{min}} \) the minimum period of \( N \) partitions as follows:
\[ \Delta_{\text{min}} = \min_{k=1}^{N} \{\Delta_k\} \tag{10} \]

Step 2 (Mapping periods to harmonics): We have to transform the periods \( \{\Delta_1, \Delta_2, \ldots, \Delta_N\} \) of the partitions to a harmonic set since otherwise a distance-constraint schedule may not be generated if the periods are not harmonic [13].

Two different techniques to transform the set of periods \( \{\Delta_1, \Delta_2, \ldots, \Delta_N\} \) to a set of harmonic periods, denoted by \( \{\pi_1, \pi_2, \ldots, \pi_N\} \), are proposed, called the naive transformation and robust transformation.

Naïve Transformation: This approach is based on the principle that if component \( A_k \) is schedulable in partition \( (m'_k, \xi_k, \Delta_k) \), then \( A_k \) is also schedulable in partition \( (m'_k, \xi_k, \Delta_{\text{min}}) \) where \( \Delta_{\text{min}} \leq \Delta_k \). Each of the periods \( \Delta_k \) for \( k = 1, 2, \ldots, k \) is transformed to \( \Delta_{\text{min}} \) where \( \Delta_{\text{min}} \) is defined in Eq. (10).

Therefore, the set of \( N \) partitions parameters’ is \( \{m'_1, \xi_1^{\text{min}}, \pi_1\}, \{m'_2, \xi_2^{\text{min}}, \pi_2\}, \ldots, \{m'_N, \xi_N^{\text{min}}, \pi_N\}\) where \( \pi_k = \Delta_{\text{min}} \) for \( k = 1, 2, \ldots, N \).

Robust Transformation: The robust approach is based on a technique proposed by Chan and Chin [2], called \( S_a \) specialization. The \( S_a \) specialization with respect to \( \Delta_{\text{min}} \) is performed as follows:
\[ \pi_k = \Delta_{\text{min}} \cdot 2^{(\log_2(\Delta_k/\Delta_{\text{min}})} \tag{11} \]

Example 1. The periods \( \{\Delta_1 = 2, \Delta_2 = 5\} \) are transformed to harmonic periods \( \{\pi_1 = 2, \pi_2 = 2\} \) in naive approach and to \( \{\pi_1 = 2, \pi_2 = 4\} \) in robust approach.

Note that for both naïve and robust approaches \( \pi_k \leq \Delta_k \) for all \( k = 1, 2, \ldots, N \) and \( \pi_k \) divides \( \pi_{k'} \) whenever \( \pi_k \neq \pi_{k'} \).

Step 3 (Allocation of the servers): Given a multicore platform having \( m \) physical cores, we have to allocate the \( m'_k \) servers, each having capacity \( \xi_k^{\text{min}} \) and period \( \pi_k \), for all \( k = 1, 2, \ldots, N \), on \( m \) cores. A total of \( \sum_{k=1}^{N} m'_k \) servers for all the \( N \) partitions have to be allocated on \( m \) cores.

We use first-fit bin packing heuristic [13] to allocate the servers on \( m \) physical cores. During allocation we use the following uniprocessor schedulability condition: a set of servers on a core are guaranteed to schedulable using Rate-Monotonic (RM) scheduling policy if and only if the sum of their capacities is not larger than 1 (follows directly from harmonicity of the periods). Therefore, when assigning a server to a core, we need to verify that the sum of the capacities in that core is not larger than 1 after the server is allocated to that core. If there is no processor on which a server can be allocated, we declare “failure”. Otherwise, if every server is allocated to some core, we declare “success”.

Without loss of generality, assume that one server of the partitions \( S_1, S_2, \ldots, S_z \) are successfully allocated to some core where \( S_k \equiv (m'_k, \xi_k^{\text{min}}, \pi_k) \) for \( k = 1, 2, \ldots, z \). We define the set \( \mathcal{S} \) as follows:
\[ \mathcal{S} = \{(x_k, \pi_k)e_k = \xi_k^{\text{min}} \cdot \pi_k \text{ where } k = 1, 2, \ldots, z\} \tag{12} \]
If integer timing parameters are assumed, then we consider period $\Delta_k$ before transforming it to harmonic period $\pi_k$ and $\epsilon_k = [\epsilon_k^{\min} \cdot \pi_k]$. Each $(\epsilon_k, \pi_k) \in S$ represents one of the $m_k$ servers of partition $S_k$ with execution time $\epsilon_k$ and period $\pi_k$. The uniprocessor RM schedule in a hyperperiod of the servers in $S$ can be generated in pseudo-polynomial time. Due to the harmonicity of the periods, $\epsilon_k$ execution time units are allocated the same way in each period $\pi_k$, which guarantees the distance-constrained property.

Transforming periods to harmonics does not introduce processor utilization overhead since the minimum capacity requirement $\xi_k^{\min}$ of a partition server is expressed in terms of percentage of processor capacity; not in terms of absolute CPU time units. And, capacity remains unchanged after the periods are transformed to harmonics. For example, allocating 50% capacity in every period of 15 ms or 8 ms are equivalent in terms of total CPU time allocated over some lifetime of the system — total 6 hours of CPU time over a lifetime of 12 hours are allocated when the capacity is 50% regardless whether the period is 15 ms or 8 ms.

System-level Table-Driven scheduling. We consider that the system-level static table in each core stores a set of triples (start time, end time, partition name) in order of increasing start time of the partitions scheduled in one hyperperiod. The scheduler can be implemented based on timer interrupt, i.e., a timer is set to interrupt the processor when it is time to start the next partition — determine by consulting the stored static table in each core.

Consider that there is an interrupt at time $t$ in some core. At this instant, if there is a server of partition $S_{next}$ that is also allocated to that core and has start time $t_{next}$ where $t = t_{next}$ (whether such a partition exists can be decided using the static table), then the timer is set equal to the difference between the end-time of $S_{next}$ and $t$. And, the control is handed over to the scheduler to start execution of partition $S_{next}$ on that core. If $t < t_{next}$, then the timer is set equal to a duration of $(t_{next} - t)$ and the core is idle between $[t, t_{next})$ and interrupted again at time $t_{next}$.

V. OVERHEAD AWARE SCHEDULING

Recently, Phan et al. [19] proposed interesting techniques to generate task-level overhead-aware interface considering hierarchical scheduling on uniprocessor. However, system-level overhead-aware schedulability analysis is not addressed in [19]. In this section, we close this gap for multicores by performing system-level overhead-aware schedulability analysis by assuming that each component’s interface is task-level (not system-level) overhead-aware. We consider the following overheads at the system-level:

Partition release overhead ($O^{rel}$): The maximum time needed to execute the timer’s interrupt service routine (ISR).

Partition scheduling overhead ($O^{sch}$): The maximum time needed to start the execution of a partition. This overhead is incurred when timer ISR transfers control to the scheduler.

Partition context-switch overhead ($O^{ctxs}$): The maximum time a core needs to switch from one partition to another. The overhead is due to the switching of execution stack and processor registers.

Partition preemption overhead ($O^{pre}$): The maximum time a core needs to restore the cache blocks of the task preempted earlier due to partition’s context switching.

We do not experimentally measure the value of these overheads for any particular platform; rather we show how these overhead need to be accounted during the system-level overhead-aware schedulability analysis.

Accounting Overheads in Analysis. Consider a static schedule in some core generated by preemptive uniprocessor RM scheduling of the servers in $S = \{(\epsilon_k, \pi_k)_{k=1}^{N}\}$ given in Eq. (12). We denote $\chi_k$ the number of preemptions that each instance of server $S_k$ suffers on that core.

Since each instance of server $S_k$ is scheduled exactly the same way in each period due to the harmonicity of the periods in that core, there are $(\chi_k + 1)$ different chunks of each instance of server $S_k$ in that core. The system-level overhead-aware schedulability analysis is based on accounting overhead for each of these $(\chi_k + 1)$ chunks in every period of each server $S_k \in S$, where $k = 1, 2, \ldots z$.

Since the periods are harmonic, the execution of each chunk of server $S_k$ in a core is immediately preceded by the execution of a (higher priority) server $S_{k-1}$ for $k = 2, 3, \ldots z$. Therefore, the start of the execution of each such chunk suffers one release overhead and one scheduling overhead. This is because each chunk has an entry in the static table and the release timer triggers the ISR when start-time of the chunk is equal to the current time. And, the ISR in turns call the scheduler to start the execution of that chunk. In addition, the context of the chunk is restored back from stack and the cache blocks lost due to the execution of (high priority) server $S_{k-1}$ are restored. Moreover, at the end of the execution of this chunk (i.e., when the next release timer interrupts), the context of the current task is stored back in stack for later retrieval. In summary, each of the $(\chi_k + 1)$ chunks of servers $S_2, S_3, \ldots S_z$ suffers one release, one scheduling, one context-switch\footnote{In fact two context-switch overheads: one for retrieval and one for storing the context; we consider that $O^{ctxs}$ includes both.} and one preemption overhead.

We now consider the chunks of $S_1$. There is exactly one chunk of server $S_1$ in each period, i.e., $\chi_1 = 0$ since it is never preempted. There is at least one instance of server $S_1$ whose execution is preceded by the execution of server $S_2$, for example, the instance of $S_1$ that executes after the end of level-$N$ busy period in the RM schedule of $S$. Such an instance of $S_1$ suffers one scheduling, one context-switch and one preemption overhead. Since the same capacity is allocated for all the instances of $S_1$ in partitioned system, we have to consider that there is a scheduling, context-switch
and preemption overhead for every instance of server $S_1$. In contrast to the chunks of servers $S_k$ for $k \geq 2$, an instance of server $S_1$ may suffer from more than one release overhead, due to the so called ISR amortization problem \cite{19}, which can be explained as follows.

Consider that there is a timer interrupt at time $t$ and the next partition starts at time $t_{next}$ such that $t_{next} > t$. Consequently, the interval $[t, t_{next})$ is idle. Since $S_1$ has the smallest period and all the periods in a core are harmonics, a new instance of server $S_1$ is eligible for execution at time $t_{next}$. This instance of $S_1$ suffers one release overhead due to timer interrupt at time $t_{next}$. Moreover, if $(t_{next} - t) < O^{rel}$, then the execution of the ISR due to the timer interrupt at time $t$ is not complete by $t_{next}$ and continues execution even after $t_{next}$ for $(O^{rel} - t_{next} + t)$ time units. This is because interrupts are serviced as soon as they arrive \cite{3}. And, such execution of an earlier ISR can delay the execution of $S_1$ by $(O^{rel} - t_{next} + t)$ time units. If $(t_{next} - t)$ is very small in comparison to $O^{rel}$, then an instant of $S_1$ may suffer from two back-to-back release overheads — one at $t$ and another at $t_{next}$ when $(t_{next} - t) \approx 0$. To avoid such ISR amortization problem, we consider that each instance of $S_1$ suffers two release overheads. In summary, each of the instances of $S_1$ suffers two release, one scheduling, one context-switch and one preemption overhead.

If $e_k$ is the task-level overhead-aware execution time, then its system-level overhead-aware (inflated) execution time, denoted by $\hat{e}_k$, is

$$
\hat{e}_k = \begin{cases} 
    e_k + (2 \cdot O^{rel} + O^{sch} + O^{cxs} + O^{pre}) & \text{if } k = 1 \\
    e_k + (O^{rel} + O^{sch} + O^{cxs} + O^{pre}) \cdot (\chi_k + 1) & \text{if } k > 1
\end{cases}
$$

(13)

where $\chi_k$ is the number of preemptions that each instance of server $S_k$ for $k \geq 2$ suffers in that given core. The overhead for each instance of server $S_k$ is $(\hat{e}_k - e_k)$. The total overhead of system-level scheduling in one hyperperiod of length $H$ for a given core is $\sum_{k=1}^{z} H(\hat{e}_k - e_k)$. To compare different possible static schedules in a core, we define a metric called, hyperperiod-based normalized overhead (NOVE), as follows:

$$
\text{NOVE} = \left( \sum_{k=1}^{z} H \left( \frac{\hat{e}_k - e_k}{\pi_k} \right) \right) / H = \sum_{k=1}^{N} \left( \frac{\hat{e}_k - e_k}{\pi_k} \right)
$$

(14)

The system-level overhead-aware schedulability test when allocating servers to some physical core is given as follows: all the servers allocated to the core is schedulable if and only if $\sum_{k=1}^{z} \frac{\hat{e}_k}{\pi_k} \leq 1$. To compute $\hat{e}_k$, we have to find the value of $\chi_k$ for each server $S_k$ by simulating the RM schedule in each core as follows.

We start executing the servers in set $S$ using execution time $e_k = c_k$ for server $S_k$ where $e_k$ is overhead-unaware execution time. We initialize $\chi_k = 0$. If server of partition

3By measuring the length of all such idle intervals $[t, t_{next})$ in the hyperperiod of the system-level schedule in a core, the maximum value of $O^{rel} - (t_{next} - t)$ can be determined and more precise release overhead for each instance of $S_1$ can be computed.

$S_k$ is preempted, then $\chi_k$ is incremented by one and $\hat{e}_k$ for this new $\chi_k$ is computed using Eq. (13). This process continues until the first instance of server $S_k$ completes total $\hat{e}_k$ time units. When the process stops, $\chi_k$ is the number of preemptions of each instance of server $S_k$. Computing $\chi_k$ for all the servers has pseudo-polynomial time complexity.

VI. SLACK ALLOCATION

The system-level scheduling overhead can be reduced by intelligently selecting the capacity and period for the server in each core so that NOVE is reduced. We have assumed so far that the capacity and period for all the $m'$ servers of partition $S_k$ are same. We can relax this assumption, i.e., the capacity and period of the $m'$ servers can be different in different cores as long as they satisfy the capacity-period relationship in Eq. (2). This is because a server's capacity needs to be increased to satisfy Eq. (2) when the period increases, which guarantees that between the release time and deadline of a job of any task $\tau_i$, enough capacity is provided so that deadline of $\tau_i$ is not missed. In other words, when Eq. (2) is satisfied by each of the $m'_k$ servers having different capacity-period pairs, it is guaranteed that enough capacity is allocated to each task of each application $A_k$. Based on this observation, we propose a slack-distribution (SD) algorithm that can be used to distribute the slack capacity when allocating the servers in a core so that overall processor bandwidth is saved.

Consider the allocation of the servers $S = \{(e_k, \pi_k)\}_{k=1}^{z}$ in a core where set $S$ is defined in Eq. (12) based on task-level (not system-level) overhead-aware interfaces $I_1, I_2, \ldots, I_z$. Algorithm SD tries to lengthen the harmonic period of each server $S$ based on unfolding operation of the interfaces starting from the server having the smallest period. The objective is to make all the periods equal so that we have a large hyperperiod with no preemption.

Algorithm SD is designed based on the following heuristic: Consider two servers of partitions $S_i$ and $S_j$ in set $S$ with parameters $(e_i, \pi_i)$ and $(e_j, \pi_j)$ such that $\pi_i < \pi_j$. If the smaller period of server $S_i$ is made equal to the larger period of server $S_j$, then server $S_j$ suffers no preemption by server $S_i$. However, increasing the period of server $S_i$ may require additional capacity to guarantee G-FP schedulability based on Eq. (2). Given the new (lengthened) harmonic period $\pi_i = \pi_j$ for server $S_i$, the lower bound on capacity, say $\xi_i$, to guarantee G-FP schedulability of the tasks of application $A_i$ can be derived by applying the unfolding operation of interface $I_i$ described in subsection III-B. The (new) execution time and period of server of partition $S_i$ is now $(e_i = \xi_i \cdot \pi_j, \pi_j)$. And, the extra capacity in addition to $\xi_k^{min}$ that is used to make server $S_i$’s period equal to server $S_j$’s period is $(\xi_i - \xi_k^{min})$.

Since increasing the capacity, hence, execution time of $S_i$ may cause additional preemptions on the lower priority partitions, we try to lengthen the (now equal) periods of both
servers $S_1$ and $S_j$ further in order to nullify such negative effect on lower priority partitions. We continue lengthening the smaller periods until we do not have sufficient slack capacity to lengthen any period. The pseudocode and detail description for algorithm SD is given in the appendix.

VII. EXPERIMENTAL ANALYSIS

In this section, we present the result of our experimental analysis to investigate our proposed system-level overhead-aware schedulability analysis. The empirical investigation into the following five ways to test the system-level schedulability in each core are evaluated:

- **UNT Test**: Checks whether all servers allocated to a core are schedulable where periods are transformed using *Naïve Transformation* considering system-level overhead-Unaware schedulability analysis.
- **URT Test**: Checks whether all servers allocated to a core are schedulable where periods are transformed using *Robust Transformation* considering system-level overhead-Unaware schedulability analysis.
- **ANT Test**: Same as UNT but considering system-level overhead-Aware schedulability analysis.
- **ART Test**: Same as URT but considering system-level overhead-Aware schedulability analysis.
- **ART-SD test**: Same as ART with *Slack Distribution*.

To compare these system-level schedulability tests, simulation experiment using randomly generated interface sets are conducted. An *interface set* is a set of $N$ interfaces that represent $N$ applications. The metric, called *acceptance ratio*, is used to evaluate the effectiveness of each schedulability test. The acceptance ratio of a schedulability test is the percentage of the randomly generated interface sets that are deemed schedulable by that test. Before presenting the results, the interface sets generation algorithm is presented next.

**Interface Set Generation**: An interface set with $N$ interfaces is characterized by $(m, d, \xi_{\text{min}})$ where $m$ is the number of physical processors, $d$ is the degree of each interface and $\xi_{\text{min}}$ is the *normalized minimum* capacity of all the $N$ partitions over $m$ processors. In other words, the total *minimum* capacity of all the $N$ partitions is $\xi_{\text{min}} \cdot m$. The value $\xi_{\text{min}} \cdot m$ is an upper bound on the total utilization of all the partitions, hence represents the load of the system. We consider consolidation of $N = 2m$ applications on a $m$ core platform. We also consider $m'_k = m$ for each randomly generated interface $I_k$ where $k = 1, 2, \ldots, N$. In other words, each partition $S_k$ has $m'_k = m$ servers.

The normalized capacity $\xi_{\text{min}}$ is distributed among the $N$ interfaces using UUnifast algorithm [6]. The UUnifast algorithm with parameters $N$ and $\xi_{\text{min}}$ returns $N$ different capacity values $\xi_1^\text{min}, \xi_2^\text{min}, \ldots, \xi_N^\text{min}$ such that $\xi_1^\text{min} + \ldots + \xi_N^\text{min} = \xi_{\text{min}}$. Remember that $\xi_k^\text{min}$ is the minimum capacity (given in Eq. (5)) that guarantees component $A_k$’s schedulability on $m'_k$ servers, where each of the $m'_k$ servers of partition $S_k$ has capacity $\xi_k^\text{min}$. Therefore, the value of $x_1$ for the point $P_1$ of interface $I_k$ is now known, i.e., $x_1 = \xi_k^\text{min}$. The other parameters of each interface $I_k$ with degree $d$ are generated based on Property 1 and Property 2 of an interface presented in subsection III-B. The detail procedure to generate the other parameters of an interface $I_k$ is given in the appendix.

**Overhead Values**. Based on the experimental results of Phan et al. [19], we consider $O^{\text{req}} = 13.72\, \mu s$, $O^{\text{sch}} = 36.56\, \mu s$, $O^{\text{cache}} = 86.91\, \mu s$, and $O^{\text{pre}} = 139.12\, \mu s$ where 139.12$\mu s$ is the cache-related preemption delay considered for each task. Note that when a server of partition $S_k$ allocated in some core is preempted, the cost of at most one task’s preemption needs to be considered since other cost of preemptions are already assumed to be included in the task-level overhead-aware interfaces. Although these overhead values are computed for a particular system in [19], we believe that our experimental result based on these overhead values would be able to highlight the effectiveness of our system-level overhead-aware schedulability analysis, usefulness of the new interface model and the bandwidth saving by algorithm SD.

**Experiments**. A total of 1000 interface sets with parameter $(m, d, \xi_{\text{min}})$ are generated at each of the 40 different minimum capacity values $\xi_{\text{min}} \in \{0.025, 0.5, 0.975, 1\}$. Each of the 1000 interface sets generated at a particular capacity level $\xi_{\text{min}}$ has $2m$ interfaces — each with degree $d$. We generate the 1000 interfaces at each $\xi_{\text{min}}$ such that each of the sets are successfully allocated in all $m$ cores using the FF heuristic [13]. Consequently, the acceptance ratios of UNT and URT tests are 100%. Our purpose is to show how overhead-aware analysis impact this 100% acceptance ratio of overhead-unaware UNT and URT tests. We ran our simulations on a Dell OptiPlex 990 Intel Dual Core i7-2600 processor, each with speed 3.40 GHz.

**A. Result Analysis**

A series of experiments for different pairs of $(m, d)$, where $m \in \{2, 4, 8\}$ and $d \in \{5, 10, 15, 20\}$ are conducted. The important trends based on these experiments are presented. Figure 2 and Figure 3 presents the acceptance ratios for experiments $(m = 4, d = 5)$ and $(m = 8, d = 5)$ where the x-axis represents the minimum capacity $(x_{\text{min}})$ and the y-axis represents the acceptance ratios.

![Figure 2](image_url)  
**Figure 2.** Acceptance ratio for $(m = 2, d = 5)$.

**Observation 1**: Overhead-unaware analysis is unsafe. None of the overhead-aware schedulability tests (i.e., ANT, ART,
Observation 2: Acceptance ratio of overhead-aware ANT, ART and ART-SD tests decreases as minimum capacity $\xi_{\text{min}}$ increases in Figure 2 and Figure 3. This is because slack capacity decreases with increasing $\xi_{\text{min}}$ and such diminishing slack capacity becomes insufficient to compensate overheads without missing some deadlines.

Observation 3: The acceptance ratios of ANT, ART and ART-SD tests in Figure 5 for $m = 4$ is lower than that of in Figure 2 for $m = 2$ at each capacity level $\xi_{\text{min}}$. Increasing $m$ from 2 to 4 means increasing the number of applications from $N = 2m = 4$ to $N = 2m = 8$ and increasing the number of servers from 16 to 32. With more servers, there is a higher possibility of increased number of partitions’ context switches and preemptions. Consequently, the overhead increases in relatively shorter hyperperiod and allocation of servers fails.

Observation 4: The performance of overhead-aware naive transformation (i.e., ANT test) is poor in both Figure 2 and Figure 3. This is because the hyperperiod of the static schedule based on naive transformation is set to the minimum period $\Delta_{\text{min}}$ of the servers. Therefore, capacity is more frequently allocated than it is necessary for many servers. Such frequent invocation of servers increases partition context switches and preemptions. Consequently, the overhead increases in relatively shorter hyperperiod and allocation of servers fails.

Robust approach (i.e., ART test) performs better than the ANT test since its hyperperiod is relatively larger due to the mapping of the periods using $Sa$ specialization. The robust transformation with slack-distribution, i.e., ART-SD test, significantly outperforms both ANT and ART tests in Figure 2 and Figure 3. This demonstrates the power of our proposed interface model and its unfolding operation applied to lengthen the period in order to reduce overhead (i.e., value of NOVE) using the slack distribution algorithm SD.

To further demonstrate the effectiveness of ART-SD test, we conducted another set of experiments to find the bandwidth saving of ART-SD test over ART test. We computed the average value of NOVE over all the processors only for the schedulable interface sets based on the both ART and ART-SD tests. The difference between the average NOVE of ART test and average NOVE of ART-SD test at each $\xi_{\text{min}}$ is the bandwidth saved by the ART-SD test. We consider $m = \{2, 4, 8\}$ and $d = 10$. The result of these experiments are given in Figure 4 where the x-axis represents capacity ($\xi_{\text{min}}$) and the y-axis represents the bandwidth saving by ART-SD test.

Observation 5: The bandwidth saving by ART-SD is significant for each selection of $m$. For example, when the capacity $\xi_{\text{min}}$ is $\leq 70\%$, the ART-SD requires around 10%, 6% and 2.5% less bandwidth on average in each processor in comparison to that of the ART test for $m = 8$, $m = 4$ and $m = 2$, respectively. This shows that the new interface model and its unfolding operation is very powerful to distribute slack capacity using algorithm SD for reducing system-level overhead. Such bandwidth saving can be used to integrate new applications.

VIII. CASE STUDY: SPACE APPLICATION

In this section, we present the analysis of our overhead-aware schedulability analysis for a real-world space application workload provided by RUAG Space AB in Sweden (the details of the workload is given in Table VIII in the appendix). There are seven applications with total utilization 2.492 in Table III. The tasks of some application are replicated (e.g., there are 3 instances of “MM File download” task of the “Data Handling” application). Our purpose is to determine the minimum number of processors to schedule all the seven applications based on ARINC-653 standard.

Since the DA-LC test runs in polynomial time and the folding operation (proposed in Subsection III-B) needs to solve at most $O((n_k)^2)$ pairs of linear equations, where $n_k$ is the number of tasks in application $A_k$, the interface of an application can be generated in polynomial time. The interfaces for all the seven applications is generated in 380 ms in our experimental setup. For example, the interface for “Instrument Control (IC)” application is

$$I_{IC} = (2, \{(0.551, 0.2ms), (0.259, 4.28ms), (1, 30ms)\})$$

where $m' = 2$ is the number of servers each with minimum capacity $\xi_{IC}^{\text{min}} = x_1 = 0.551$ and the corresponding upper bound on period is 0.2ms/0.551 = 0.36ms. The maximum degree of interface $I_{IC}$ is $d_{\text{max}} = 3$. The values of $\xi_k^{\text{min}}, d_{\text{max}}$ and $m'_k$ for all the seven applications are given in Table IV.
There are total 9 servers of the 7 applications. The servers are allocated using two different bin-packing heuristics — first-fit decreasing period (FF-DP) and first-fit decreasing capacity (FF-DC) — assuming an unlimited number of cores. The minimum number of cores required to successfully allocate all the 9 servers based on each of the five tests (i.e., UNT, URT, ANT, ART and ART-SD) is determined. We found that the minimum number of processors required for both FF-DP and FF-CP heuristics using the overhead-unaware tests (i.e., UNT and URT tests) is \( m = 5 \).

For overhead-aware analysis, we consider a platform such that \( O^{rel} = \kappa \times 13.727\mu s \), \( O^{sch} = \kappa \times 30.565\mu s \), \( O^{exx} = \kappa \times 86,917\mu s \) and \( O^{pre} = \kappa \times 139.12\mu s \), where \( \kappa > 0 \) is a control parameter used to investigate the impact of various degrees of overhead in different hardware platforms.

The minimum number of cores required for each allocation heuristic using each of the overhead-aware ANT, ART and ART-SD tests for each \( \kappa \) is determined. The results are shown in Table III which also presents the average NOVE over all the processors required for successfully allocating the 9 servers for each allocation heuristic and each value of \( \kappa \).

**Table I**

**Table II**

**Observation:** While each of the overhead-unaware UNT and URT tests requires \( m = 5 \) processors, each of the overhead-aware ANT and ART tests requires \( m = 6 \) processors (notice that the value of \( m \) in the fourth (ANT) and fifth (ART) columns in Table III is 6). It can be observed from the last (ART-SD) column of Table III that the minimum number of processors required is \( m = 5 \) using overhead-aware ART-SD test even for high-overhead (i.e., \( \kappa = 2 \)) platform. On the other hand, each of the overhead-aware ANT and ART tests requires at least \( m = 6 \) processors even for low-overhead (i.e., \( \kappa = 1 \)) platform.

It is easy to see from the minimum capacity requirement \( \xi_{k}^{\min} \) given for each application in Table III that the minimum number of processors required for any possible allocation of the servers is \( m = 5 \) for platform having overhead. This is because the two servers of “AOCS”, the two servers of “Instrument Control” and the server of “FDIR” application require to be allocated in at least five different physical processors. The slack-distribution algorithm SD thus finds the optimal number of processors for the given spaceborne applications in Table III. If the number of physical processors has to be reduced further, then improved task-level schedulability analysis for each application has to be applied, for example, using the approach proposed in [18].

The FF-DP heuristic is better than the FF-DC heuristic considering the ART-SD test when \( \kappa = 1 \). This is because both FF-DP and FF-DC allocation heuristics require \( m = 5 \) processors but FF-DP saves \((6.4\% - 5.5\%) = 0.9\%\) average bandwidth in all the \( m = 5 \) processors. On the other hand, the FF-DC heuristic is better than FF-DP heuristic considering the ART-SD test for \( \kappa = 2 \) since the former heuristic requires \( m = 5 \) processors while the latter requires \( m = 6 \) processors. Therefore, bin-packing heuristic plays an important role, depending on the overhead of the target platform, in minimizing bandwidth and number of processors.

### State-of-the-art and our approach

In addition to the bin-packing heuristic, the G-FP schedulability test used to generate the interfaces has high impact on the number of processors required to successfully allocate all the servers.

The (state-of-the-art) bounded-delay multipartition (BDM) interface model proposed by Lipari and Bini [17] generalizes or mitigates the limitations of previously proposed interface models (e.g., MPR [20], MSF [5], and PSF [4] models). However, the generation of a BDMA interface in [17] is based on a relatively more pessimistic global FP schedulability test in comparison to the DA–LC test [11], which we have used in this paper. We demonstrate this pessimism of BDMA interface using the “Spacecraft Control (SC)” application in Table III.

The BDMA interface for application SC requires minimum capacity of 0.015 while our approach requires capacity of 0.011 in one server. This is because the schedulability test for BDMA interface considers the higher priority “Battery Management” task as a carry-in task while the DA–LC test considers this task as a non-carry-in task (please see [11] for the definition of carry-in task). As a result, the lower priority “Thermal control” task is considered to suffer 100 ms of interference from the higher priority “Battery Management” (carry-in) task when generating the BDMA interface while in our case the interference is only 50 ms. Therefore, our approach is not more pessimistic than the state-of-the-art approach in generating the interfaces of an application.

The minimum capacity requirement of an application can be reduced further by considering relatively less pessimistic global FP schedulability analysis, for example, using hybrid-
priority-assignment policy [18], or using Audsley’s optimal priority assignment (OPA) algorithm combined with the DA–LC test [11]. We applied this combination of OPA and DA–LC test to generate the interfaces of the applications in Table III. We found that each application needs only one server and \( m = 4 \) cores are required to successfully allocate all the servers using all the three tests for both FF–DC and FF–DP heuristics and for both \( \kappa = 1 \) and \( \kappa = 2 \). And, if there is no overhead, then \( m = 3 \) cores are enough to allocate all the servers. Note that the interfaces in Table III are generated considering some (random) priority assignment given in Table III and require minimum \( m = 5 \) cores to allocate all the servers. Finally, based on the concept of robust priority assignment introduced by Davis and Burns, the minimum capacity requirement for each interface can be optimized further and the number of processors for allocating a collection of servers can be reduced. Due to brevity, such optimizations are not presented in this paper.

IX. Conclusion

In this paper, a two-level ARINC-653 compliant hierarchical scheduling and its overhead-aware schedulability analysis is proposed. A new way to model interface with a tunable parameter (degree \( d \)) which balances level of abstraction and accuracy is proposed. This model while being concise is very powerful for performing overhead-aware system-level schedulability analysis. Due to the flexible nature of the interface the system designer can select the best set of capacity/period pairs for each application.

A method to generate an interface is proposed based on task-level schedulability analysis. And, the construction of an overhead-aware system-level schedule is presented. We also designed a slack distribution algorithm that allocates additional capacity to the servers to save the overall processing bandwidth. Experimental results show the usefulness of our proposed overhead-aware slack-distribution algorithm for real-world space applications.

REFERENCES


APPENDIX

SLACK DISTRIBUTION ALGORITHM (SD)

Given a set of \( z \) servers in set \( S = \{ (v_k, \pi_k) \}_{k=1}^{z} \) to be allocated in a core. Algorithm I computes set \( S \) in line 1 based on Eq. (12) using the interfaces \( I_1, \ldots, I_z \). The execution time of the servers in set \( S \) are only task-level overhead aware. The flag \( invalid[j] \) is set to \( false \) in line 2 to specify that each server \( S_j \)’s period has the potential to be increased by allocating additional capacity.

The while loop in line 3–19 iterates as long as there are two tasks having different periods in set \( S \) having their invalid flag set to \( false \). During each iteration of the while loop, two servers \( S_i \) and \( S_j \) from set \( S \) having two different smallest periods are selected in line 4–5 such that \( \pi_i < \pi_j \). If no such partitions can be found in \( S \) (checked in line 6), the algorithm returns set \( S \).

If two such servers of partitions \( S_i \) and \( S_j \), where \( \pi_i < \pi_j \), are found in \( S \) (i.e., condition in line 6 is false), the capacity required, say \( \xi_i \), for server \( S_i \) to make its period equal to the period of server \( S_j \) is determined in line 9 based on the unfolding operation of interface proposed in subsection III-B. We make a copy of the
Algorithm 1: Slack Distribution (SD)

Input: Intra-component overhead-aware interfaces $I_1, \ldots, I_z$

1. Compute $S = \{(e_i, \pi_i)\}^z_{k=1}$ based on Eq. (12)
2. $Valid[i] \leftarrow false \forall i = 1, 2, \ldots, z$

while true do

4. $S_i \leftarrow$ Partition having the smallest period in $S$ such that 
   $Invalid[i] = false$
5. $S_j \leftarrow$ Partition having the smallest period in $S - \{\pi_i\}$
   where $\pi_i \neq \pi_j$ and $Invalid[j] = false$

if two such $S_i$ and $S_j$ are not found in $S$ then

7. return $S$
else

8. Unfold interface $I_k$ to find minimum capacity $\xi_k$ for 
    given period $\pi_i$, based on the operation presented in 
    subsection III-B
9. $(e, p) \leftarrow (e_i, \pi_i)$
10. Remove $(e_i, \pi_i)$ from $S$ and add $\{(\xi_k \cdot \pi_j, \pi_j)\}$ to $S$
11. Compute $U \leftarrow \sum_{k=1}^z \frac{\xi_k}{\pi_k}$ where $\xi_k$ is in Eq. (13).

if $U > 1$ then

13. // Slack is not enough
14. $S \leftarrow S - \{(\xi_k \cdot \pi_j, \pi_j)\}$
15. $Valid[i] \leftarrow true$
16. $S \leftarrow S \cup \{(e, p)\}$

end

end

old execution time and period of server $S_i$ in line 10 in case we 
fail to allocate sufficient slack. We update set $S$ by removing old 
parameters $(e_i, \pi_i)$ of partition $S_i$ and adding the new parameters 
$(\xi \cdot \pi_j, \pi_j)$ in line 11. And, we compute in line 12 the total system-
level overhead-aware capacity requirement, denoted by variable $U$, 
based on the analysis in Section III-B

If $U > 1$, then period of $\pi_i$ of server $S_i$ cannot be increased to 
$S_j$’s period $\pi_j$ and element $(\xi \cdot \pi_j, \pi_j)$ of server $S_i$ is removed 
from set $S$ in line 14. Consequently, the invalid flag of $S_i$ is set to 
true (line 15) since $S_i$’s period cannot be increased. The old 
parameters of $S_i$ (that was copied in line line 10) is stored back in 
set $S$ in line 16 and the next iteration starts from line 3.

If $U \leq 1$ (i.e., condition in line 13 is false), then period of 
server $S_i$ is successfully increased to $S_j$’s period $\pi_j$. When the 
algorithm stops, the set $S$ is returned. Set $S$ accounts only the 
task-level overhead-aware execution time of the partitions and the 
new periods. We can now compute the value of NOVE considering 
set $S$ returned by algorithm SD.

If all the periods of the servers in set $S$ are same, say $\pi_{\text{common}}$, 
after $S$ is returned by algorithm SD then we can again try to 
lengthen all the periods further by $\gamma$ time units by allocating more 
slack capacity if available. Such an upper bound on $\gamma$ can be 
determined using binary search on $\gamma$ and the value of NOVE 
can be reduced further.

GENERATING THE PARAMETERS OF AN INTERFACE

Given that the value of $x_1$ is known, we demonstrate below how the values of $x_2, x_3, \ldots, x_d$ and $y_1, y_2, \ldots, y_d$ for 
the points in interface $I_d$ are generated.

- Since the degree of each interface is $d$, we set $x_d = 1 - \epsilon$ 
  where $\epsilon = 0.001$. We select the value of $x_i$, starting with 
  $i = 2$ to $i = (d - 1)$, from uniform distribution in the range 
  $(x_{i-1}, x_d)$. Now value of $x_i$ for each point $P_i$ of $I_d$ is known.
- According to Property 2 given in subsection III-B, 
  given of an interface $I_k = (m_{ik}, \{P_1, P_2, \ldots, P_d\})$, the line segments 
  connecting points $(P_i, P_{i+1})$ and $(P_{i+1}, P_{i+2})$ must satisfy 
  $m_{i+1} > m_{i+1, i+2}$ where $m_{i,j}$ is the slope of the line connecting 
  points $(P_i, P_j)$. The slope of a line connecting points $P_i$ and $P_j$ 
  is essentially a relative deadline of some task in application $A_k$. The tasks’ 
deadline of real avionics and space workloads are generally in the range of 
$[25\text{ms}, 1000\text{ms}]$. To this end, a total of $(d - 1)$ different random values from 
uniform distribution in the range $[25\text{ms}, 1000\text{ms}]$ are generated.

and, each such value, considering decreasing order, is set as the 
value of $m_{i+1} \geq i$ starting from $i = 1$ to $i = (d - 1)$.

- We now show how $y_1$ is generated. Since point $P_1 = (x_1, y_1)$ 
is on the line having slope $m_{i+1, i+2}$ and is also on the line 
segment that lower bounds Eq. (4), the value of $y_1$ is equal to 
$y_1 = (x_1 \cdot m_{1,2} - L_i)$ where $L_i$ is selected from uniform 
distribution between $0, x_1 \cdot m_{1,2}$ so that the line $y = (x \cdot m_{1,2} - L_i)$ forms the lower bound of 
Eq. (4). So, the value of $y_1$ is now known. And, the 
value of $y_{i+1}$ for $i = 1, 2, \ldots, (d - 1)$ are computed using 
$y_{i+1} = m_{i+1, i+2} \cdot (x_{i+1} - x_i) + y_i$ where the terms on the 
right-hand side are known. At this point all $x_i$ and $y_i$ values of each point $P_i$ 
are known for interface $I_k$ with degree $d$. And, $m_{ik}$ for interface $I_k$ is set equal to $m$.

REAL-WORLD WORKLOAD OF 7 SPACE APPLICATIONS

<table>
<thead>
<tr>
<th>Application (priority) Task name</th>
<th>$T_1$</th>
<th>$D_1$</th>
<th>$C_r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Handling</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2) Reception-routing</td>
<td>1000</td>
<td>1000</td>
<td>25</td>
</tr>
<tr>
<td>(5) Time tagged</td>
<td>1000</td>
<td>1000</td>
<td>25</td>
</tr>
<tr>
<td>(4) Position commanding</td>
<td>1000</td>
<td>1000</td>
<td>25</td>
</tr>
<tr>
<td>(0) HK-1M collection generation</td>
<td>100</td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>(1) TM routing</td>
<td>100</td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>(6) MM File upload</td>
<td>1000</td>
<td>1000</td>
<td>25</td>
</tr>
<tr>
<td>(7–9) MM File download (3 instances)</td>
<td>1000</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>(10–19) File packet management (10 instances)</td>
<td>1000</td>
<td>1000</td>
<td>3</td>
</tr>
<tr>
<td>Spacecraft control</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0) Battery management</td>
<td>10000</td>
<td>10000</td>
<td>50</td>
</tr>
<tr>
<td>(1) Thermal control</td>
<td>10000</td>
<td>10000</td>
<td>50</td>
</tr>
<tr>
<td>AOCS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4) Mode control</td>
<td>1000</td>
<td>1000</td>
<td>70</td>
</tr>
<tr>
<td>(0) Sensor pre-processing</td>
<td>100</td>
<td>100</td>
<td>7</td>
</tr>
<tr>
<td>(1) Control Algorithm</td>
<td>100</td>
<td>70</td>
<td>50</td>
</tr>
<tr>
<td>(2) Antenna pointing</td>
<td>100</td>
<td>70</td>
<td>7</td>
</tr>
<tr>
<td>(3) Solar panel control</td>
<td>100</td>
<td>70</td>
<td>7</td>
</tr>
<tr>
<td>Instrument control</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4) Mode management</td>
<td>1000</td>
<td>1000</td>
<td>5</td>
</tr>
<tr>
<td>(3) Mission data management</td>
<td>1000</td>
<td>1000</td>
<td>10</td>
</tr>
<tr>
<td>(2) Instrument FDIR</td>
<td>1000</td>
<td>1000</td>
<td>10</td>
</tr>
<tr>
<td>(1) Instrument configuration</td>
<td>200</td>
<td>60</td>
<td>30</td>
</tr>
<tr>
<td>(0) Instrument processing</td>
<td>200</td>
<td>200</td>
<td>110</td>
</tr>
<tr>
<td>FDIR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0) Monitoring</td>
<td>200</td>
<td>200</td>
<td>4</td>
</tr>
<tr>
<td>(1) Event reporting</td>
<td>200</td>
<td>200</td>
<td>4</td>
</tr>
<tr>
<td>(2) Event action</td>
<td>200</td>
<td>200</td>
<td>4</td>
</tr>
<tr>
<td>(4) SC configuration management</td>
<td>1000</td>
<td>1000</td>
<td>20</td>
</tr>
<tr>
<td>(3) On board procedures</td>
<td>200</td>
<td>200</td>
<td>4</td>
</tr>
<tr>
<td>IO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0) IO manager</td>
<td>20</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Star tracker</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0–2) Start tracker processing (3 instances)</td>
<td>500</td>
<td>500</td>
<td>83</td>
</tr>
</tbody>
</table>

Table III

SPACECRAFT WORKLOAD FROM RUAG SPACE AB. The period ($T_i$), 
deadline ($D_i$) and WCET ($C_r$) are in ms. Number in 
parentheses before each task’s name in second column is its 
priority (smaller number implies higher priority).