Mixed-Criticality Multicore Scheduling of Real-Time Gang Task Systems

Ashikahmed Bhuiyan[†,*], Kecheng Yang[‡,*], Samsil Arefin[¶], Abusayeed Saifullah[§], Nan Guan[♣], Zhishan Guo[†, **]
[†]*Department of Electrical and Computer Engineering, University of Central Florida*
[‡]*Department of Computer Science, Texas State University*
[¶]*Microsoft New England Research and Development Center*
[§]*Department of Computer Science, Wayne State University*
[♣]*Department of Computing, The Hong Kong Polytechnic University*

***Abstract*—Mixed-criticality (MC) scheduling of sequential tasks (with no intra-task parallelism) has been well-explored by the real-time systems community. However, till date, there has been little progress on MC scheduling of parallel tasks. MC scheduling of parallel tasks is highly challenging due to the requirement of various assurances under different criticality levels.In this work, we address the MC scheduling of parallel tasks of gang model that allows workloads to execute on multiple cores simultaneously.Such a workload model represents an efficient mode-based parallel processing scheme with many potential applications. To schedule such task sets, we propose a new technique GEDF-VD, which integrates Global Earliest Deadline First (GEDF) and Earliest Deadline First with Virtual Deadline (EDF-VD). We prove the correctness of GEDF-VD and provide a detailed quantitative evaluation in terms of speedup bound in both the MC and the non-MC cases. Specifically, we show that GEDF provides a speedup bound of 2 for non-MC gang tasks, while the speedup for GEDF-VD considering MC gang tasks is $\sqrt{5}+1$. Experiments on randomly generated gang task sets are conducted to validate our theoretical findings and to demonstrate the effectiveness of the proposed approach.**

***Index Terms*—Multi-core systems, Mixed Criticality, Real-time scheduling, Speedup bound.**

I. Introduction

Due to size, weight, and power considerations, there is a trend that multiple tasks with different criticality levels (that are subject to varying degrees of assurance/verification) share a computing platform [1]. This type of system is commonly known as a *mixed-criticality (MC) system*, where each task can be associated with various execution budgets. During normal operation, all tasks are scheduled according to their typical execution budget. However, some critical tasks may exceed their typical budget and need more resources to finish their execution. In these scenarios, if the available resources are not sufficient, the less critical task will be sacrificed to free up the resources for accommodating the additional computational requirements requested by the more critical ones.

Take an avionics software standard as an example, where the ground control subsystems are more safety-critical than ground communication and light controls. During the incident of emergency (e.g., an accident), it is more important to execute the safety-critical components rather than the other components. On the other hand, in normal condition, all these components are required to perform smoothly (for more details, refer to the Table 1.1 of [2], which demonstrates the RTCA DO-178B avionics software standard). MC scheduling has received considerable attention (refer to [3] for a thorough and updated survey) as it brings significant improvements in resource efficiency.

Note that safety-critical MC systems have tight correctness requirements. These requirements can be verified by two related but orthogonal perspectives: *a priori verification* and *run-time robustness* [4]. Before run-time, a priori Verification determines whether a system will behave correctly (or not) during execution, while run-time robustness deals with unexpected system behavior at run-time. There are some debates in terms of the applicability of MC into run-time robustness [5] [6]. However, no criticism is valid on applying Vestal model for a priori verification [4], which is the scope of our work.

Parallel Computing Workloads. Recent advances in parallel computing allow executing a single piece of code simultaneously on multiple computing units. Such design provides a much better capability of exploiting the benefits provided by modern platforms. As a result, there is an urgent need in handling workload models that allow intra-task parallelism (i.e., parallel tasks). Parallel computing systems perform a large number of computations and often need to interact with their surroundings under real-time constraints, e.g., arms system (RADAR). In these types of applications, a lot of processors co-operate with each other, and these communications are timing critical. It is necessary for a system to have both high performance and predictability; i.e., efficient control that minimizes the introduced overhead, while responding to external events (coming through sensors) in real-time. The gang task model is a practical, widely used, and representative workload model for intra-task parallelism [7], [8], [9] [1]. In gang scheduling, all threads of a task are grouped into a gang. While executing, the whole group is concurrently scheduled on distinct cores. Also, the gang task model is supported by some widely used parallel computing programming standard (e.g., OpenACC [10]), which is commonly used in the graphics processing unit (GPU).

∗Equal contribution.

** Corresponding author: zsguo@ucf.edu.

[1] [9] pointed out some fundamental flaws in [7].

Existing Work. The real-time systems and parallel computing communities have given considerable attention towards these two directions: *MC scheduling* and *scheduling of parallel tasks*. These two emerging trends bring in some critical and exciting problems, and there is an emerging need in integrating those two trends. There has been extensive research on the (a) MC scheduling of sequential (i.e., non-parallel) tasks (refer to the recent survey in [11], [3]) and (b) scheduling of parallel tasks with a single-criticality level [7], [8], [9], [12], [13], [14]. Till date, very few efforts [15], [16], [17] have been made towards the combined problem of MC scheduling of parallel tasks. To our knowledge, none of these efforts has considered mixed-criticality gang task scheduling on multi-core platforms.

Motivation Behind This Work. Multi-core platform enables applications that require better energy efficiency, higher performance, and real-time guarantees. The notion of MC systems with the intra-task parallelism stems from many current trends. For example, the number of cores fabricated on a chip is increasing rapidly. Besides, the computational demand for an individual task (with stringent timing requirements) is rising, which makes it essential to consider the intra-task parallelism. Furthermore, when safety-critical and non-safety-critical tasks share a common computational platform, there is an increasing demand to integrate functionality with different levels of criticality. Such demand promotes the idea of MC scheduling, i.e., combining various functionalities of varying criticality levels onto the same computing platform.

Challenges. In gang task model, a task cannot start execution until the number of available cores is no less than what is required by it (i.e., a task's *degree of parallelism*). This simple constraint adds a huge restriction on real-time schedulability and makes the problem highly challenging. We are aware of only one known correct schedulability analysis [9] under Global Earliest Deadline First (GEDF) for gang tasks. Besides, integrating MC in gang scheduling scheme adds additional challenges due to the dual notion of correctness. That is, in the normal mode a task may have a utilization less than 1, while in the critical mode the utilization could be much higher than 1 [9]—schedulers do not know the exact behavior of each task prior to run-time (non-clairvoyant). The scheduler must be able to detect the critical condition early enough so that it can allocate more resources to the more critical tasks to handle this drastic change and still be able to meet the deadlines.

This research. In this paper, we study the real-time scheduling of MC gang tasks on identical multi-core platforms. We propose the first scheduling algorithm GEDF-VD (GEDF with Virtual Deadline) for MC gang tasks. Our approach leverages the synthesis of uniprocessor scheduling techniques such as EDF-VD [18] as well as *GEDF* [9] that was designed for non-MC gang tasks. To our knowledge, this is the first work that studies the MC scheduling of the gang task model. Specifically, we make the following contributions:

- We generalize the gang task model to the MC context by incorporating required extensions and propose GEDF-VD for the generalized model. We also conduct a utilization based schedulability test and prove its correctness formally.
- We prove that the speedup bound [19] for GEDF to gang tasks in a non-MC platform is at most $\big(2 - 1/(M + 1 - \min_i\{m_i\})\big)$, where M denotes the total number of processor cores and m_i denotes the degree of parallelism of task τ_i. To our knowledge, this is the first speedup bound result for GEDF scheduling of gang tasks.
- With the result from the previous step, we then derive a speedup bound of $\sqrt{5}+1$ for GEDF-VD considering MC gang tasks.
- Extensive simulations under randomly generated task sets are conducted to demonstrate the real-time performance, and effectiveness of the proposed algorithm in terms of acceptance ratio which is defined as the ratio of the number of schedulable task sets over the total number of task sets.

Organization. The remainder of this paper is organized as follows. Section II describes the task model, notations, and preliminaries. Section III provides a detailed description of our scheduling algorithm and prove its correctness. Section IV derives the speedup bounds for the non-MC and MC platform, under GEDF and GEDF-VD scheduling algorithms, respectively. Simulation results are presented in Section V. Section VI discusses related prior work. Section VII concludes this paper and points out future research directions.

II. Dual-Criticality Gang Task Model

In this work, we consider the problem of scheduling a task set $\tau = \{\tau_1, \ldots, \tau_n\}$ of n independent *implicit deadline* (i.e., the period of a task is equal to its deadline) sporadic MC gang tasks on M identical cores. In this model, each task generates an infinite number of MC gang jobs (the j^{th} job of task τ_i is denoted as $\tau_{i,j}$). To describe the dual-criticality gang task model, first, we provide details on traditional *non-MC gang task model* and *MC sporadic sequential task model*. Then, by leveraging these two models, we generalize the gang task model to the MC context. We restrict our attention to dual-criticality because there are many unsolved issues for the dual-criticality model, specifically in the parallel computing domain. We consider this work as an important step towards the multi-criticality systems.

Non-MC gang task model. In traditional non-MC gang task model, each task τ_i is represented with a 4-tuple (m_i, c_i, T_i, D_i), where each job of τ_i requires access to m_i cores for at most c_i time units to complete its execution, T_i is the task period, and D_i is the relative deadline. The relative deadline D_i specifies that for each of the released jobs $\tau_{i,j}$ (of task τ_i), its deadline $d_{i,j} = r_{i,j} + D_i$, where $r_{i,j}$ denotes the release time of $\tau_{i,j}$ [9]. The *utilization* u_i of each task $\tau_i \in \tau$ is given by $u_i = (m_i c_i)/T_i$ and the overall system utilization is: $U_{sum} = \sum_{\tau_i \in \tau} u_i$. Note that, it is possible that the value of u_i is larger than one, which is different from the traditional sequential task model. Based on the scheduling flexibility, a gang task τ_i can be categorized into three groups. A task τ_i is said to be:

- *rigid*, if m_i is fixed a priori and does not change throughout the execution,
- *moldable*, if m_i is fixed during its activation and does not change throughout the execution,
- *malleable*, if m_i is not fixed and can be changed during its execution by the scheduler.

In this work, we focus on the *rigid* task model.

MC sporadic task model. In a dual-criticality systems, the criticality level of τ_i is represented by $\chi_i = \{\text{LO}, \text{HI}\}$. The worst case execution time (WCET) estimations of each task is also represented by a tuple $(c_i^{\text{LO}}, c_i^{\text{HI}})$ where c_i^{LO} and c_i^{HI} represent the LO and HI-criticality WCETs respectively. c_i^{HI} is measured by a more pessimistic tool by considering all possible scenarios, while c_i^{LO} is calculated using a less pessimistic yet realistic tool. Collection of all LO- and HI-criticality tasks in τ are denoted by τ_{LO} and τ_{HI} respectively. u_i^{LO} and u_i^{HI} denotes the utilization of τ_i in LO- and HI-criticality mode respectively, where $u_i^{\text{LO}} = c_i^{\text{LO}}/T_i$ and $u_i^{\text{HI}} = c_i^{\text{HI}}/T_i$.

MC gang task model. By leveraging the above two models, in our work, we consider a workload model of MC gang tasks, where each task τ_i is represented by a 6-tuple $(m_i, \chi_i, c_i^{\text{LO}}, c_i^{\text{HI}}, T_i, D_i)$, where

m_i = number of cores required for τ_i.
χ_i = criticality level of each task τ_i and $\chi_i \in \{\text{LO}, \text{HI}\}$.
$c_i^{\text{LO}}(c_i^{\text{HI}})$ = τ_i's WCET in LO(HI)-criticality mode.
T_i = minimum inter-arrival time between jobs.
D_i = relative deadline.

If $\forall \tau_i, m_i = 1$, i.e., degree of parallelism for each gang task is 1, our analysis (Section III and IV) will reduce to the existing MC scheduling method designed for the sporadic task model. We believe this is common for a restricted special case of a more complex and expressive model. For example, the directed acyclic graph (DAG) task model [13], [14] is popular to represent intra-task parallelism. Many of the existing schedulability analysis considering the DAG model would also reduce to prior study for ordinary sporadic tasks if the number of nodes of each DAG task is equal to 1.

Now, we generalize the utilization concepts to suit the MC gang task model, which are analogous to the above-mentioned concepts. Refer to the Example 1 for details.

$$U_{\text{LO}}^{\text{LO}} \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_{\text{LO}}} m_i \times c_i^{\text{LO}}/T_i,$$

$$U_{\text{HI}}^{\text{LO}} \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_{\text{HI}}} m_i \times c_i^{\text{LO}}/T_i,$$

$$U_{\text{HI}}^{\text{HI}} \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_{\text{HI}}} m_i \times c_i^{\text{HI}}/T_i$$

TABLE I: An MC gang task set with GEDF schedule shown in Figure 1.

Task ID	c_i^{LO}	c_i^{HI}	T_i	χ_i	m_i
τ_1	3	4	5	HI	3
τ_2	3	3	10	LO	2
τ_3	1	2	10	HI	2

Example 1. *Consider the task-set $\tau = (\tau_1, \tau_2, \tau_3)$ in Table I. For this task-set we derive the utilization as follows:* $U_{\text{LO}}^{\text{LO}} = \tau_2^{\text{LO}} \times m_2/T_2 = 0.6, U_{\text{HI}}^{\text{LO}} = \tau_1^{\text{LO}} \times m_1/T_1 + \tau_3^{\text{LO}} \times m_3/T_3 = 2,$ *and* $U_{\text{HI}}^{\text{HI}} = \tau_1^{\text{HI}} \times m_1/T_1 + \tau_3^{\text{HI}} \times m_3/T_3 = 2.8.$

Example 2. *Consider the MC gang task set in Table I to be scheduled in four cores. A GEDF schedule for this task set is shown in Figure 1. The system starts at* LO*-criticality mode, and all the tasks (τ_1, τ_2, τ_3) will execute up-to C_i^{LO}. At a mode switch ($t = 9$), all* LO*-criticality tasks (τ_2) are dropped, and all* HI*-criticality tasks (τ_1, τ_3) will execute up-to C_i^{HI}. Recall that, m_i is the degree of parallelism of τ_i. Hence, τ_1 cannot execute at $t = 5$ as it needs three cores to execute while only two cores (P_3 and P_4) are idle. After a mode switch, all* HI*-criticality jobs (including the ones which are currently executing) will execute up-to their* HI*-criticality WCET.*

Motivations behind this model. Some commonly used parallel computing programming standards (e.g., OpenACC [10]) support the gang task model. OpenACC is one of the parallel computing programming standards used for the GPU architecture which is a hot research topic (few to mention [20], [21], [22]). GPU architecture is popular because of the features like (1) highly threaded but low context switch latency architecture, (2) high parallelism and (3) minimal dependency between data elements, etc. Previous works on GPU scheduling considered limited or no preemption policy [20], [21]. However, this work is motivated by some recent attempts to incorporate the *preemptive* support in GPUs. For example, a prototype has been implemented and tested with preemptive support (at the pixel level and the thread level) in a virtualized environment in a recent work [23]. Its prototype is EDF based, and enhanced with a bandwidth isolation mechanism (e.g., constant/total bandwidth servers [24]) for the graphics and computing workloads. Also, the prototype is tested on a recent NVIDIA Tegra-based system on a chip (SoCs) [25]. Since some recent works study the preemptive support in the GPU architecture, there is a need for a comprehensive study of gang task scheduling using GEDF.

Now, we introduce some definitions and preliminaries which will be frequently used in later sections of this paper.

Definition* 1.** ***(MC-correct schedule): *Scheduling strategy must ensure an* MC-correct *schedule, as defined below [17].*

- *If the system stays in normal condition (i.e., each task in the system finishes execution within its* LO*-criticality WCET), all tasks must meet their deadlines.*
- *If the system transits into a critical condition (i.e., there exists a* HI*-criticality task executing beyond its* LO*-criticality WCET), all* HI*-criticality tasks must meet their deadlines, while* LO*-criticality tasks need not so.*

Definition* 2.** ***(Executing/Non-Executing interval) *An interval $[t_1, t_2)$ (where $t_1 < t_2$) is an **executing** interval for a task τ_i if m_i out of M cores are executing the current active job released by τ_i throughout this interval. Otherwise, $[t_1, t_2)$ is a **non-executing** interval for τ_i. An illustrative example is*

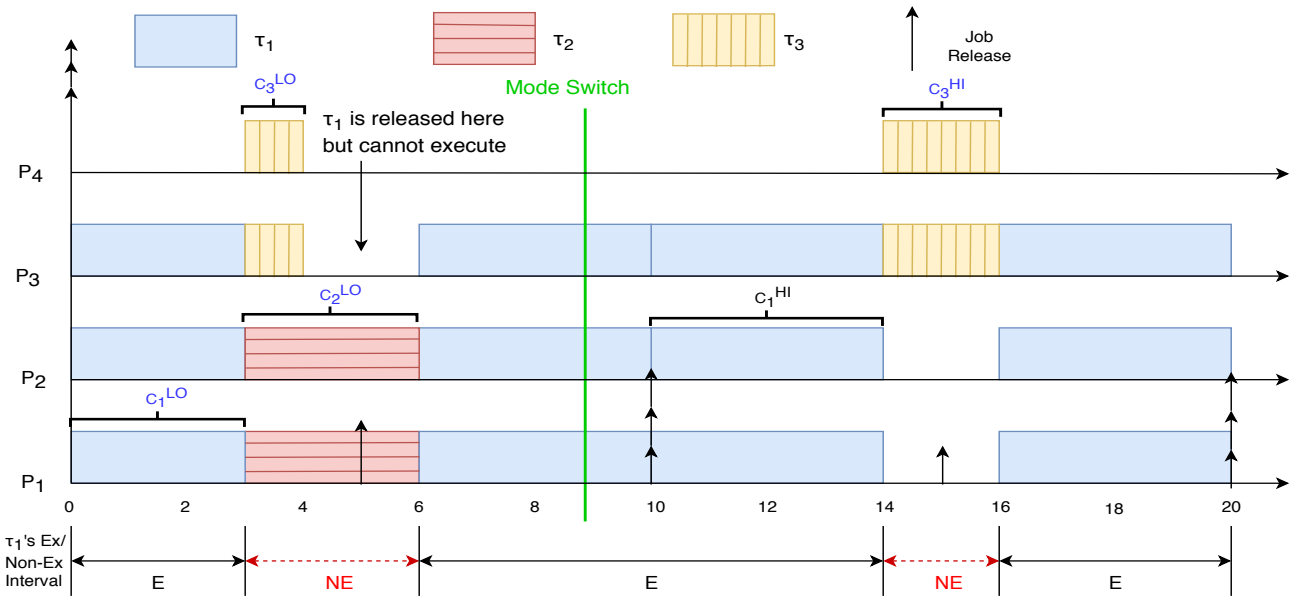

Fig. 1: A GEDF scheduling of the MC gang task-set from Table I and the executing (E)/non-executing (NE) intervals of τ_1.

shown in Figure 1 by pointing the executing and non-executing intervals for task τ_1.

Definition* 3.** ***(Active/pending task) *If there exists a task $\tau_i \in \tau$, such that it has a job $\tau_{i,j}$ where $r_{i,j} \leq t < d_{i,j}$. Here, $r_{i,j}$ and $d_{i,j}$ respectively denotes the release time and deadline of $\tau_{i,j}$, then τ_i is considered as an active task at time t. A job is* pending *if it is released but not finished [9].*

***Definition* 4.** *Maximum possible number of idle cores (Δ_i) for a task τ_i refers to the maximum number of available cores (that are not executing any job) at any time during τ_i's non-executing intervals in which it has a pending job [9].*

Example 3. *Let us consider a task-set $\tau = \{\tau_1, \tau_2, \tau_3, \tau_4\}$ have to be scheduled on ten cores. Degree of parallelism for these tasks are given as: $m_1 = 6$, $m_2 = m_4 = 4$ and $m_3 = 3$. For this task-set, $\Delta_1 = \Delta_2 = \Delta_4 = 3$ and $\Delta_3 = 2$. This is because τ_1 cannot execute at time t (although it has a pending job) if both τ_2 (or τ_4) and τ_3 are executing at t. The degree of parallelism for τ_2 (or τ_4) and τ_3 is 4 and 3 respectively. So, the maximum number of idle cores for τ_1 is Δ_1, where $\Delta_1 = 10-(4+3) = 3$. We can calculate the value of Δ_2, Δ_3 and Δ_4 in the same approach (refer to Algorithm 1 in [9]).*

System behavior and scope of this work. It is expected that an MC system starts execution in normal mode. The system-wide mode transition is triggered if a HI-criticality task τ_i has received cumulative execution length beyond its LO-criticality WCET and did not signal its finishing. Likewise the Vestal model [1], after a mode switch, no LO-criticality tasks get any service guarantee. After mode transition (from LO-criticality to HI-criticality), at the first idle instant, the system switches back to the LO-criticality mode again. All other scenarios (e.g., a HI-criticality task runs for more than its HI-criticality WCET) are considered as *erroneous*, where no guarantees will be made and hence is not considered in this work.

III. GEDF-VD for Dual-Criticality System

Now we describe our algorithm for the MC task systems considering the GEDF-VD algorithm. In this work, we consider an implicit deadline (So, we use the terms *deadline* and *period* interchangeably) sporadic task systems on preemptive identical multi-core platforms. We integrate an uniprocessor MC scheduling technique (*EDF-VD* [18]) with a multiprocessor gang task scheduling technique (*GEDF* [9]) and derive a new algorithm named GEDF-VD (Subsections III-A and III-B). In our approach, we determine a *scaling factor*, which scales the deadline of all HI-criticality tasks at LO-criticality mode. This factor will be calculated in such a way that the correctness of the system can be guaranteed at both LO- and HI-criticality modes (Subsections III-C and III-D).

A. *EDF-VD and GEDF-VD: An Overview*

EDF-VD. In case of a mode switch (LO to HI), to generate an *MC-correct schedule* (Definition 1), a scheduler must ensure that all HI-criticality tasks meet their deadlines (while LO-criticality tasks can be sacrificed). To guarantee this criterion, a specific amount of CPU time must be reserved for those HI-criticality tasks even if the system is running at LO-criticality mode. This reservation of time can be achieved by shortening the deadlines of HI-criticality tasks under normal mode—those are virtual deadlines.

In EDF-VD, deadlines of all HI-criticality tasks are shortened by multiplying them with a *scaling factor,* and this modified deadline is called the virtual deadline. During run-time (at LO-criticality mode), all HI-criticality tasks are executed according to their virtual deadline, and all LO-criticality tasks

execute with their actual/original deadlines according to EDF. Upon a mode switch, only the HI-criticality tasks are executed in EDF order with respect to their actual/original deadlines.

In the case of a LO- to HI-criticality mode-switch, a HI-criticality task demands additional computational requirements. Setting a virtual deadline for the HI-criticality tasks leaves enough time so that the extra workload can be finished within their actual deadlines. If the virtual deadline is too short, it increases the system density at normal (i.e., LO-criticality) mode, while a large virtual deadline threats the schedulability of the system after a LO- to HI-criticality mode switch. The trick is to determine a balanced scaling factor x, such that the correctness under both execution modes can be guaranteed. [18] showed the steps to calculate the minimum x that guarantees the schedulability of all tasks in the system. They also proved that by reducing the deadline for HI-criticality tasks at LO-criticality mode, system schedulability can be improved.

Remark 1. *In this work, we consider a completely different Gang task workload model in a multi-core platform. As a result, the approach to calculate the scaling factor x in [18], as well as the schedulability test, are no longer applicable for our case. We propose a novel approach to calculate a feasible scaling factor x in this section.*

GEDF-VD. Now, we provide an overview of our algorithm (GEDF-VD) considering an implicit-deadline sporadic MC gang task system τ to be scheduled on M identical cores. The GEDF-VD algorithm starts by checking whether GEDF can successfully schedule the *regular task system*. A regular task system denotes that, all LO-criticality tasks will execute up-to their LO-criticality WCET and all HI-criticality tasks will execute up-to their HI-criticality WCET. It returns *SUCCESS* immediately if the regular task system is schedulable. Otherwise, all HI-criticality tasks can execute up-to their LO-criticality WCETs and their deadline is shortened (i.e., virtual deadline) and set to $\hat{T}_i = xT_i$, while all LO-criticality tasks execute up-to their LO-criticality WCETs with their original deadline. If any of the currently executing job (of a HI-criticality task) executed beyond its LO-criticality WCET and did not signal its completion by $\hat{T}_i$, the scheduler immediately discards all currently active LO-criticality jobs. Also, the deadline for all HI-criticality jobs is changed to their release time plus their actual deadline. Subsection III-B provides a detailed description of GEDF-VD algorithm.

B. GEDF-VD: A Detailed Description

In this subsection, we describe the GEDF-VD scheduling approach in a two-phase process. First, we describe what happens prior to run-time (denoted as a *pre-processing phase*). In this phase, GEDF-VD determines whether (or not) it is required to set a virtual deadline for the HI-criticality tasks. A lower and an upper bound of the virtual deadline is also calculated in this phase. Then, we discuss how the jobs are scheduled at run-time (denoted as *handling the dispatched jobs at run-time*). We present the pseudo-code for (the run-time part of) GEDF-VD in Algorithm 1.

Algorithm 1: GEDF-VD (online part)

Input: A dual-criticality task-set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ and a feasible x.

```
/* Handling tasks at run-time */
Whenever a job is released by tasks τ_i at time instant t
if τ_i ∈ τ_HI then
    d_{i,j} = t + xT_i;
end
if τ_i ∈ τ_LO then
    d_{i,j} = t + T_i;
end
Schedule all active jobs by GEDF according to d_ij's.
if ∃τ_{i,j} ∈ τ_HI that is not finished by d_{i,j} at time t' then
    /* Mode Switch */
    for ∀τ_i ∈ τ_HI do
        d_{i,j} = d_{i,j} + (1 − x)T_i
    end
    Discard all τ_i ∈ τ_LO
    Schedule τ' = {τ_HI} by GEDF.
end
```

Pre-processing phase. In this phase, we perform a schedulability test for *ordinary (non-MC)* GEDF to determine whether (or not) it can successfully schedule: (i) all $\tau_i \in \tau_{\text{LO}}$ up-to their LO-criticality WCET (c_i^{LO}), and (ii) all $\tau_i \in \tau_{\text{HI}}$ up-to their HI-criticality WCET (c_i^{HI}). If the GEDF test fails, then, for each HI-criticality task $\tau_i \in \tau_{\text{HI}}$, a virtual deadline $\hat{T}_i$ is computed (Step-2), and they execute up-to their LO-criticality WCET (c_i^{LO}).

Step 1. We start by checking whether the task-set can be successfully scheduled by GEDF. If so, then GEDF directly schedules the system. Else, we modify the task deadlines (Step 2).

Step 2: An additional *virtual deadline* parameter $\hat{T}_i$ is calculated for each HI-criticality task τ_i, where $\hat{T}_i = xT_i$. A schedulability test for GEDF-VD is provided next. Furthermore, when the schedulability test is passed, x can be arbitrarily chosen from the range $[A, B]$ while GEDF-VD is guaranteed to generate an MC-correct schedule, where A and B are defined and can be easily calculated for any given system by the following equations:

$$A = \max\{A_1, A_2\}; \tag{1}$$

$$A_1 = \max_{i:\tau_i \in \tau_{\text{LO}}} \left\{ \frac{U_{\text{HI}}^{\text{LO}}}{M - \Delta_i - U_{\text{LO}}^{\text{LO}}} \right\}; \tag{2}$$

$$A_2 = \max_{i:\tau_i \in \tau} \left\{ \frac{m_i U_{\text{HI}}^{\text{LO}} + u_i^{\text{LO}}(M - \Delta_i - m_i)}{m_i(M - \Delta_i - U_{\text{LO}}^{\text{LO}})} \right\}; \tag{3}$$

$$B = \min_{i:\tau_i \in \tau_{\text{HI}}} \left\{ 1 - \frac{m_i U_{\text{HI}}^{\text{HI}} + u_i^{\text{HI}}(M - \Delta_i - m_i)}{m_i \times (M - \Delta_i)} \right\}. \tag{4}$$

A schedulability test for GEDF-VD. The following theorem provides a sufficient schedulability test for GEDF-VD.

Theorem 1. *An MC gang task system is schedulable under GEDF-VD upon M identical unit-speed processors if both conditions hold:*

$$U_{\text{LO}}^{\text{LO}} < M - \max_i\{\Delta_i\}, \tag{5}$$

$$A \leq B. \tag{6}$$

We will prove this theorem later by proving Lemmas 1 and 2 in Subsections III-C and III-D.

Recall that, $\Delta_i < M$ for all i. Therefore, $m_i U_{\text{HI}}^{\text{LO}} + u_i^{\text{LO}}(M - \Delta_i - m_i) = m_i(U_{\text{HI}}^{\text{LO}} - u_i^{\text{LO}}) + u_i^{\text{LO}}(M - \Delta_i) > 0$, which with (5) together implies $A > 0$; and also $m_i U_{\text{HI}}^{\text{HI}} + u_i^{\text{HI}}(M - \Delta_i - m_i) = m_i(U_{\text{HI}}^{\text{HI}} - u_i^{\text{HI}}) + u_i^{\text{HI}}(M - \Delta_i) > 0$, which implies $B < 1$. Thus, both (5) and (6) being true implies that $0 < A \leq B < 1$, which guarantees that any x chosen from $[A, B]$ must be a valid scaling factor such that $0 < x < 1$.

Run-time dispatch. Similar to GEDF, at any specific time instant, a task with the earliest deadline gets the highest priority. In case of ties, task with a smaller index is favored. Let a binary variable ξ indicate the system-criticality level, then consider the following two possible cases:

Case 1. System is at LO-criticality mode ($\xi = 0$), j^{th} job of task τ_i arrives at time t:

(i) If τ_i is a LO-criticality task, set the deadline as $d_{i,j} = t + T_i$, else set $d_{i,j} = t + \hat{T}_i$, where $\hat{T}_i = xT_i$.

(ii) If any of the currently executing jobs executes for more than c_i^{LO} and does not signal completion, then the system switches to the HI-criticality mode (Case 2).

Case 2. While the system is at HI-criticality mode ($\xi = 1$):

(i) Discard all LO-criticality tasks (or use background scheduling).

(ii) Update the deadline for the currently active HI-criticality jobs into release time (of these jobs) plus their actual relative deadline.

(iii) For any future HI-criticality task τ_i that releases a job at time t, the deadline is set to $t + T_i$.

(iv) When there is an idle instant, switch to the LO-criticality mode (Case 1)[2].

C. *Proof of Correctness at* LO*-Criticality Mode*

In this subsection, we show that GEDF-VD and its schedulability test given by Theorem 1 are able to guarantee MC correctness at LO-criticality mode.

Lemma 1. *If both* (5) *and* (6) *are true, GEDF-VD guarantees that all* LO*-criticality tasks meet their deadlines and all* HI*-criticality tasks meet their* virtual *deadlines during* LO*-criticality mode.*

[2]Note that HI-criticality mode exists for certification purposes. Such both directions of mode switch should be unlikely events during run time. Please also refer to the discussions about apriori verification and run-time robustness in Section I.

Proof. According to Theorem 2 in [9], given any real-time implicit deadline sporadic gang task system τ, GEDF can schedule it successfully if

$$\begin{aligned} U_{sum} &\leq (M - \Delta_i) \times (1 - \frac{u_i}{m_i}) + u_i \\ \iff U_{sum} &\leq M - \Delta_i + u_i(1 - \frac{M - \Delta_i}{m_i}) \end{aligned} \tag{7}$$

holds for all $\tau_i \in \tau$. The virtual deadline increases the utilization of these HI-criticality tasks (and hence the whole system). Note that, in the LO-criticality mode, each HI-criticality task is scheduled by its virtual relative deadline xT_i while each LO-criticality task is scheduled by its actual deadline T_i. Therefore, it is sufficient to view each LO-criticality task as a sporadic task with utilization u_i^{LO} and view each HI-criticality task as a sporadic task with utilization u_i^{LO}/x, in order to meet every LO-criticality deadline and every HI-criticality virtual deadline in LO-criticality mode. Then, for every i such that $\tau_i \in \tau$, we discuss the two cases for $M - \Delta_i - m_i$. Therefore, it suffice to evaluate (7) under such utilizations for every task τ_i. We show this by two cases: 1) $\tau_i \in \tau_{\text{HI}}$, and 2) $\tau_i \in \tau_{\text{LO}}$.

Case 1: $\tau_i \in \tau_{\text{HI}}$**.** In this case, using (7) as a result from [9], we just need the following inequality to hold for any $\tau_i \in \tau_{\text{HI}}$.

$$\begin{aligned} & U_{\text{LO}}^{\text{LO}} + \frac{U_{\text{HI}}^{\text{LO}}}{x} \leq M - \Delta_i + \frac{u_i^{\text{LO}}}{x}(1 - \frac{M - \Delta_i}{m_i}) \\ \iff & \frac{U_{\text{HI}}^{\text{LO}}}{x} + \frac{u_i^{\text{LO}}}{x}(\frac{M - \Delta_i}{m_i} - 1) \leq M - \Delta_i - U_{\text{LO}}^{\text{LO}} \\ \iff & \frac{m_i U_{\text{HI}}^{\text{LO}} + u_i^{\text{LO}}(M - \Delta_i - m_i)}{m_i \cdot x} \leq M - \Delta_i - U_{\text{LO}}^{\text{LO}} \end{aligned} \tag{8}$$

Notice that (5) implies

$$M - \Delta_i - U_{\text{LO}}^{\text{LO}} > 0 \text{ for all } i \text{ such that } \tau_i \in \tau, \tag{9}$$

and (6) allows $x \in [A, B]$ can be chosen so that $x \geq A$, which, by (1) and (3), implies

$$x \geq \frac{m_i U_{\text{HI}}^{\text{LO}} + u_i^{\text{LO}}(M - \Delta_i - m_i)}{m_i(M - \Delta_i - U_{\text{LO}}^{\text{LO}})} \text{ for all } i \text{ such that } \tau_i \in \tau. \tag{10}$$

It is clear that (9) and (10) imply (8).

Case 2: $\tau_i \in \tau_{\text{LO}}$**.** In this case, using (7) as a result from [9], we just need the following condition to hold for any $\tau_i \in \tau_{\text{LO}}$.

$$U_{\text{LO}}^{\text{LO}} + \frac{U_{\text{HI}}^{\text{LO}}}{x} \leq M - \Delta_i + u_i^{\text{LO}}(1 - \frac{M - \Delta_i}{m_i}) \tag{11}$$

Subcase 2.1: $M - \Delta_i - m_i \leq 0$**.** In this case, $M - \Delta_i \leq m_i \implies 1 - \frac{M - \Delta_i}{m_i} \geq 0$. Therefore, the following inequality implies (11):

$$U_{\text{LO}}^{\text{LO}} + \frac{U_{\text{HI}}^{\text{LO}}}{x} \leq M - \Delta_i. \tag{12}$$

Notice that (5) implies

$$M - \Delta_i - U_{\text{LO}}^{\text{LO}} > 0 \text{ for all } i \text{ such that } \tau_i \in \tau, \tag{13}$$

and (6) allows $x \in [A, B]$ can be chosen so that $x \geq A$, which, by (1) and (2), implies

$$x \geq \frac{U_{\text{HI}}^{\text{LO}}}{M - \Delta_i - U_{\text{LO}}^{\text{LO}}} \text{ for all } i \text{ such that } \tau_i \in \tau_{\text{LO}}. \tag{14}$$

It is clear that (13) and (14) imply (11).

Subcase 2.2: $M - \Delta_i - m_i > 0$. In this case, $M - \Delta_i > m_i \implies 1 - \frac{M-\Delta_i}{m_i} < 0$. So, $\frac{u_i^{\text{LO}}}{x}(1 - \frac{M-\Delta_i}{m_i}) < u_i^{\text{LO}}(1 - \frac{M-\Delta_i}{m_i})$, as $0 < x < 1$. Therefore, the following inequality implies (11).

$$U_{\text{LO}}^{\text{LO}} + \frac{U_{\text{HI}}^{\text{LO}}}{x} \leq M - \Delta_i + \frac{u_i^{\text{LO}}}{x}(1 - \frac{M - \Delta_i}{m_i}) \tag{15}$$

By the same reasoning as that for Case 1, (15) always holds because (9) and (10) are "for any $\tau_i \in \tau$" and both HI- and LO-criticality tasks are included in the set τ. That is, (11) is also true in Case 2.2 here.

Combining Cases 1 and 2 (the latter includes Subcases 2.1 and 2.2), the lemma follows. ■

D. *Proof of Correctness at* HI*-Criticality Mode*

In this subsection, we show that GEDF-VD and its schedulability test given by Theorem 1 are able to guarantee MC correctness at HI-criticality mode.

Lemma 2. *If both* (5) *and* (6) *are true, GEDF-VD guarantees that all* HI*-criticality tasks meet their deadlines during* HI*-criticality mode.*

Proof. At the mode switch point from the lo- to HI-criticality mode, a job from any task $\tau_i \in \tau_{\text{HI}}$ must be either completed or has a deadline at least $(1 - x)T_i$ after this mode-switch point; otherwise, an earlier time instant would have been the mode switch point.

Afterwards, any job from any task $\tau_i \in \tau_{\text{HI}}$ has at least $T_i > (1 - x)T_i$ (as $0 < x < 1$) time units from their releases in the HI-mode to their corresponding deadlines.

Therefore, viewing each task $\tau_i \in \tau_{\text{HI}}$ in the HI-criticality mode as a sporadic task with utilization $\frac{u_i^{\text{HI}}}{(1-x)}$ and using (7) as a result from [9], the following inequality is sufficient to ensure that all HI-criticality tasks meet their actual deadlines during HI-criticality mode. For all i such that $\tau_i \in \tau_{\text{HI}}$,

$$m_i \times \frac{U_{\text{HI}}^{\text{HI}}}{(1-x)} \leq m_i \times (M - \Delta_i) - \frac{u_i^{\text{HI}}}{1-x} \times (M - \Delta_i - m_i) \tag{16}$$

Notice that (6) allows $x \in [A, B]$ can be chosen so that $x \leq B$, which, by (4), implies the following equation holds for all i such that $\tau_i \in \tau_{\text{HI}}$:

$$x \leq 1 - \frac{m_i U_{\text{HI}}^{\text{HI}} + u_i^{\text{HI}}(M - \Delta_i - m_i)}{m_i \times (M - \Delta_i)} \tag{17}$$

Furthermore, Equation (17) is equivalent to Equation (16), as $0 < x < 1$ and $\Delta_i < M$. Thus, the lemma follows. ■

Finishing up. We establish Theorem 1 by combining Lemma 1 and 2, and it serves as a sufficient schedulability test for GEDF-VD to schedule MC gang task sets on M identical processors. In addition, Figure 2 gives a high-level intuition for validating Theorem 1, given that Lemma 1 and 2 have been proven. Note that, we did leverage some insights (in our analysis) from prior works on MC scheduling and that on gang scheduling. However, our analysis is not a straightforward combination of these earlier works due to the increased complexity in our system model. For example, in the speedup bound analysis for MC scheduling of ordinary sporadic tasks, *an individual task's utilization is at most the speed of a processor* is a straightforward and necessary feasibility condition, while it no longer holds for the gang tasks.

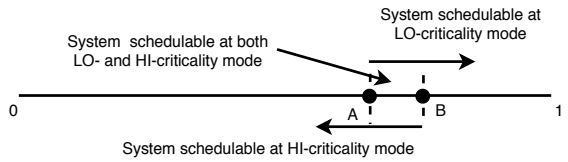

Fig. 2: Any value of the scaling factor x, where $A \leq x \leq B$, guarantees an MC-correct schedule.

IV. Speedup Bound Analysis

In this section, we evaluate the effectiveness of our algorithm GEDF-VD based on *speedup bound* metric, which is a widely accepted tool for evaluating the effectiveness of multiprocessor scheduling algorithms [19]. We will first provide the related definition and some existing results, and then (in Subsection IV-A) will derive the speedup bound for gang tasks under GEDF algorithm considering the *non-MC* systems. This is the first speedup bound for (non-MC) gang tasks under GEDF scheduling policy which lays the foundation for deriving a speedup bound for MC gang tasks. Finally, in Subsection IV-B, considering the *MC* sporadic gang tasks, we prove a speedup bound for our proposed algorithm GEDF-VD.

Definition* 5.** ***(Speedup factor and speedup bound) *For a scheduler $\mathcal{S}$, a speedup factor $\mathcal{V}$ ($\mathcal{V} \geq 1$) (also known as resource augmentation factor) means that any task set that is schedulable by an optimal scheduler on a platform of speed-1 core will be schedulable by $\mathcal{S}$ on a platform of speed-$\mathcal{V}$ core(s).*

For a scheduler $\mathcal{S}$, a speedup bound refers to the lower bound of the speedup factor $\mathcal{V}$ achievable by it. A speedup bound for a scheduler $\mathcal{S}$ provides an estimation of how far the performance of $\mathcal{S}$ is from an optimal scheduler, the lower the better.

Limitations. Our speedup factors results in this section rely on the following assumption that

$$m_i \leq \frac{M+1}{2} \text{ for all } \tau_i \in \tau, \tag{18}$$

That is, the speedup factors results in this section apply only to systems that satisfy the condition (18). Nonetheless, condition (18) was *not* required for the schedulability test and analysis in the last section, and therefore those schedulability results apply to a wider range of MC gang task systems. In practice condition (18) is often satisfied, because the number of cores in the modern platforms is increasing.

Note that gang tasks cannot be scheduled on uniprocessor platforms due to their natures of the *mandatory* parallel processor access request. Therefore, in order to compare with a potential optimal scheduler on a uniprocessor, we propose

a *De-ganging* transformation between a multiprocessor gang task set and a corresponding Liu-and-Layland (LL) task set:

- **De-ganging**: Given a gang task set $\tau = \{\tau_1, ..., \tau_n\}$, for each task $\tau_i = \{m_i, c_i, T_i\}$, construct m_i LL tasks $\{\tau_i^{'(1)}, ..., \tau_i^{'(m_i)}\}$, each with the same execution length and period, i.e., $\tau_i^{'(j)} = \{c_i, T_i\}$ for any $j = 1, ..., m_i$. For mapping of the other way around, any deganged LL task set can be clustered into n groups, where there are m_i tasks from the i^{th} group sharing the same execution time c_i and the same period T_i, resulting in a gang task $\tau_i = \{m_i, c_i, T_i\}$ of the same "total" utilization. The extension to MC task set is trivial—treat c_i as a vector and maintain the values during the transfer.

A moment thought should convince the reader that it suffices to restrict our attention to the de-ganged LL task set when deriving the speedup bound, as the de-ganged LL task set being schedulable is *necessary* for the corresponding gang task set to be schedulable. This transformation does not change the overall set utilization and thus do not change the utilization-based necessary schedulability conditions (i.e., basis of the speedup proofs). Throughout the proofs in this section, the following Greek letters will be used frequently:

$$\begin{aligned} \psi &= M/(2 - \frac{1}{M}); \\ \phi &= \frac{\sqrt{5}+1}{2} \quad \text{(i.e., golden ratio);} \\ \Phi &= \frac{\sqrt{5}-1}{2} \end{aligned} \tag{19}$$

A. *Speedup Bound for Gang Tasks under GEDF*

In this subsection, we derive the speedup bound (shown in Theorem 2) for the algorithm GEDF considering the *non-MC gang task set* τ, executing on $\mathcal{V}$-speed cores. This is the first speedup bound result for gang task under GEDF scheduling. This analysis lays the basis for deriving the speedup bound for the proposed MC gang task scheduler.

Theorem 2. *Given any de-ganged task set that is schedulable on a speed-M uni-processor, the corresponding gang task set will pass the schedulabililty test of GEDF upon a M-core system, each of speed $\mathcal{V} = 2 - 1/(M + 1 - \min_i\{m_i\})$.*

Proof. Because $1 \leq m_i \leq M$ for any i, we know that for all $\tau_i \in \tau$,

$$\begin{aligned} \mathcal{V} &= 2 - \frac{1}{M + 1 - \min_i\{m_i\}} \\ &\geq 2 - \frac{1}{M + 1 - m_i} \\ &= \frac{2M + 1 - 2m_i}{M + 1 - m_i}; \end{aligned} \tag{20}$$

From feasibility of the LL task set on a speed-M uniprocessor, we have $U_{sum} \leq M$. So,

$$\begin{aligned} (20) &\implies \mathcal{V} \geq \frac{U_{sum} + M + 1 - 2m_i}{M + 1 - m_i} \\ &\iff \frac{U_{sum}}{\mathcal{V}} \leq M - (m_i - 1) + \frac{(2m_i - M - 1)}{\mathcal{V}} \\ &\iff m_i \frac{U_{sum}}{\mathcal{V}} \leq m_i M - m_i(m_i - 1) + \frac{m_i}{\mathcal{V}}(2m_i - M - 1) \\ &\implies m_i \frac{U_{sum}}{\mathcal{V}} \leq m_i M - m_i(m_i - 1) + \frac{u_i}{\mathcal{V}}(2m_i - M - 1) \\ &\qquad [u_i \leq m_i, 2m_i - M - 1 \leq 0] \end{aligned} \tag{21}$$

The condition $2m_i - M - 1 \leq 0$ is equivalent to (18); while $u_i \leq m_i$ is true for any gang task because the utilization of each gang task τ_i is $u_i = m_i(c_i/T_i)$, where $c_i \leq T_i$. Note that u_i can also be viewed as the total utilization of the m_i de-ganged LL tasks that correspond to the gang task τ_i. Again, de-ganging preserves the utilization of the set. From Equation (21):

$$\begin{aligned} & m_i \frac{U_{sum}}{\mathcal{V}} \leq m_i M - (m_i - 1)m_i + (m_i - 1)\frac{u_i}{\mathcal{V}} + \frac{u_i}{\mathcal{V}}(m_i - M) \\ \implies & m_i \frac{U_{sum}}{\mathcal{V}} \leq m_i M - (m_i - 1)(m_i - \frac{u_i}{\mathcal{V}}) - \frac{u_i}{\mathcal{V}}(M - m_i) \\ \implies & m_i \frac{U_{sum}}{\mathcal{V}} \leq m_i M - \Delta_i(m_i - \frac{u_i}{\mathcal{V}}) - \frac{u_i}{\mathcal{V}}(M - m_i) \end{aligned}$$

[From Definition 4: $0 \leq \Delta_i \leq m_i - 1$]

$$\begin{aligned} &= m_i(M - \Delta_i) - (M - \Delta_i - m_i)\frac{u_i}{\mathcal{V}} \quad \text{[re-arrange]} \\ &= (M - \Delta_i)(m_i - \frac{u_i}{\mathcal{V}}) + m_i\frac{u_i}{\mathcal{V}} \quad \text{[re-arrange]} \\ \implies & \frac{U_{sum}}{\mathcal{V}} \leq (M - \Delta_i)(1 - \frac{u_i}{m_i\mathcal{V}}) + \frac{u_i}{\mathcal{V}} \\ & \text{[divide } m_i \text{ on both sides, re-arrange], for all i} \end{aligned} \tag{22}$$

The equation above implies that the Corresponding gang set is GEDF schedulable on M speed-$\mathcal{V}$ processors (Theorem 2 in [9]). Note that the last step is true because under speed of $\mathcal{V}$, all utilizations in the test should be treated as the ones under speed 1 divided by $\mathcal{V}$, in order to apply the original schedulability test under a speed-1 platform. ■

Theorem 2 indicates that the speedup factor of the GEDF schedulability test in Theorem 2 of [9] (for gang task set) is no greater than $\mathcal{V} = 2 - 1/(M + 1 - \min_i\{m_i\})$. Because $1 \leq m_i \leq M$ for any i, $\mathcal{V} \leq 2 - \frac{1}{M}$. Therefore, the following corollary follows directly from Theorem 2.

Corollary 1. *Given any de-ganged task set that is schedulable on a speed-M uni-processor, the corresponding gang task set will pass the schedulability test of GEDF upon a M-core system, each of speed $(2 - \frac{1}{M})$.*

Furthermore, scaling all speeds by a factor of $1/(2 - \frac{1}{M})$ lead to the following corollary.

Corollary 2. *Given any de-ganged task set that is schedulable on a speed-ψ uni-processor, the corresponding gang task set*

will pass the schedulability test of GEDF upon M unit-speed processors, where $\psi = M/(2 - \frac{1}{M})$.

B. *Speedup Bound for Gang Tasks under GEDF-VD*

The previous subsection proved the speedup bound for non-MC task under GEDF. We now brings MC and virtual deadlines into the picture, and derive the speedup bound for *MC gang task set* τ under GEDF-VD. From the definitions of ϕ and Φ in Equation (19), the following properties hold:

$$1 + \Phi = \phi = \frac{1}{\Phi} \tag{23}$$

$$\Phi + \Phi^2 = 1 \tag{24}$$

Theorem 3. *Given any de-ganged MC task set that is schedulable on a speed-$(\psi \cdot \Phi)$ uniprocessor, the corresponding MC gang task set will be schedulable under GEDF-VD upon M unit-speed processors, where $\psi = M/(2-\frac{1}{M})$ and $\Phi = \frac{\sqrt{5}-1}{2}$.*

Proof. The de-ganged MC task set being schedulable on a speed-$(\psi \cdot \Phi)$ uni-processor implies

$$\max\{U_{\text{LO}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}}, U_{\text{HI}}^{\text{HI}}\} \le \psi \cdot \Phi. \tag{25}$$

We proceed the rest of this proof in two cases.

Case 1: $U_{\text{HI}}^{\text{LO}} \ge \Phi \cdot U_{\text{LO}}^{\text{LO}}$**.** By (25) and the condition of Case 1,

$$\psi \cdot \Phi \ge U_{\text{LO}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}} \ge (1 + \Phi) U_{\text{LO}}^{\text{LO}}$$
$$\implies U_{\text{LO}}^{\text{LO}} \le \frac{\Phi}{1+\Phi} \cdot \psi = \Phi^2 \cdot \psi. \text{ [by (23)]}$$

Then, by the above and (25),

$$U_{\text{LO}}^{\text{LO}} + U_{\text{HI}}^{\text{HI}} \le \Phi^2 \cdot \psi + \Phi \cdot \psi = \psi. \text{ [by (24)]}$$

Thus, no virtual deadline needs to be set at all. Both HI- and LO-criticality tasks are scheduled by GEDF according to their actual deadlines on M unit-speed processors. By Corollary 2, no deadline will be missed.

Case 2: $U_{\text{HI}}^{\text{LO}} < \Phi \cdot U_{\text{LO}}^{\text{LO}}$**.** By (25) and the condition of Case 2,

$$\psi \cdot \Phi \ge U_{\text{LO}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}} > (\frac{1}{\Phi} + 1) U_{\text{HI}}^{\text{LO}}$$
$$= \frac{1+\Phi}{\Phi} \cdot U_{\text{HI}}^{\text{LO}} = \frac{1}{\Phi^2} \cdot U_{\text{HI}}^{\text{LO}}. \text{ [by (23)]}$$

That is,

$$U_{\text{HI}}^{\text{LO}} < \Phi^3 \cdot \psi. \tag{26}$$

Then, we have

$$\begin{aligned} U_{\text{LO}}^{\text{LO}} + \frac{U_{\text{HI}}^{\text{LO}}}{1 - U_{\text{HI}}^{\text{HI}}/\psi} &= U_{\text{LO}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}} \cdot \frac{U_{\text{HI}}^{\text{HI}}/\psi}{1 - U_{\text{HI}}^{\text{HI}}/\psi} \\ &\le U_{\text{LO}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}} \cdot \frac{\Phi}{1-\Phi} \\ &\quad [U_{\text{HI}}^{\text{HI}} \le \psi \cdot \Phi \text{ by (25)}] \\ &= U_{\text{LO}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}} + U_{\text{HI}}^{\text{LO}} \cdot \frac{\Phi}{\Phi^2} \text{ [by (24)]} \\ &< \psi \cdot \Phi + \Phi^3 \cdot \psi \cdot \frac{\Phi}{\Phi^2} \text{ [by (25) and (26)]} \\ &= (\Phi + \Phi^2)\psi \text{ [rearrange]} = \psi, \text{ [by (24)]} \end{aligned}$$

which is concluded as

$$U_{\text{LO}}^{\text{LO}} + \frac{U_{\text{HI}}^{\text{LO}}}{1 - U_{\text{HI}}^{\text{HI}}/\psi} < \psi. \tag{27}$$

In this case, one could take $x = \frac{U_{\text{HI}}^{\text{LO}}}{\psi - U_{\text{LO}}^{\text{LO}}}$ as the scaling factor to set the virtual deadlines for HI-criticality tasks. Because the de-ganged task set is schedulable on a speed-$(\psi \cdot \Phi)$ uniprocessor, $U_{\text{LO}}^{\text{LO}} \le \psi \cdot \Phi$, which implies $x > 0$, as $\Phi < 1$, $\psi > 0$, and $U_{\text{HI}}^{\text{LO}} > 0$. On the other hand, $U_{\text{HI}}^{\text{LO}} < \Phi \cdot U_{\text{LO}}^{\text{LO}}$ in Case 2, so

$$\begin{aligned} x = \frac{U_{\text{HI}}^{\text{LO}}}{\psi - U_{\text{LO}}^{\text{LO}}} &< \frac{\Phi \cdot U_{\text{LO}}^{\text{LO}}}{\psi - U_{\text{LO}}^{\text{LO}}} \\ \le \frac{\Phi \cdot \psi \cdot \Phi}{\psi - \psi \cdot \Phi} = \frac{\Phi^2}{1-\Phi} &= 1. \text{ [by (24)]} \end{aligned}$$

Thus, in this case, $0 < x < 1$ is guaranteed under this particular setting and therefore this x can always be used as the scaling factor to set the virtual deadlines for GEDF-VD.

Then, we first show that all LO-criticality tasks meet their actual deadlines and all HI-criticality tasks meet their *virtual* deadlines during the LO-criticality mode.

$$U_{\text{LO}}^{\text{LO}} + \frac{U_{\text{HI}}^{\text{LO}}}{x} = U_{\text{LO}}^{\text{LO}} + \psi - U_{\text{LO}}^{\text{LO}} = \psi.$$

By Corollary 2, the above equation implies that, using GEDF-VD to schedule the gang task set on M unit-speed processors, all LO-tasks meet their actual deadlines and all HI-tasks meet their *virtual* deadlines during the LO-mode. Next, we show that all HI-criticality tasks, including any carryover (HI-) jobs across the mode-switch point, meet their *actual* deadlines during the HI-criticality mode.

Because the virtual deadlines are set as $x \cdot T_i$ for each HI-criticality task τ_i, every HI-criticality job including the one triggering the mode switch will have *at least* $(1-x)T_i$ time units to finish its *at most* C_i^{HI} execution and to release its next job. It suffices to consider the schedulability when replacing each HI-criticality task in the HI-criticality mode by a implicit-deadline sporadic task with period $(1-x)T_i$ and execution C_i^{HI}. It can be done by checking their total utilization

$$\sum_{\tau_i \in \tau_{\text{HI}}} \frac{C_i^{\text{HI}}}{(1-x)T_i} = \frac{U_{\text{HI}}^{\text{HI}}}{1-x}.$$

On the other hand, by (27), we have

$$\frac{U_{\text{HI}}^{\text{LO}}}{\psi - U_{\text{LO}}^{\text{LO}}} < 1 - U_{\text{HI}}^{\text{HI}}/\psi,$$

and $1 - x > U_{\text{HI}}^{\text{HI}}/\psi$ holds since we set $x = \frac{U_{\text{HI}}^{\text{LO}}}{\psi - U_{\text{LO}}^{\text{LO}}}$. Thus,

$$\frac{U_{\text{HI}}^{\text{HI}}}{1-x} < \frac{U_{\text{HI}}^{\text{HI}}}{U_{\text{HI}}^{\text{HI}}/\psi} = \psi.$$

By Corollary 2, the above equation implies that, using GEDF-VD to schedule the gang task set on M unit-speed processors, all HI-criticality tasks, including any carryover (HI-) jobs across the mode-switch point, meet their *actual* deadlines during the HI-criticality mode. Thus, it concludes the proof and the theorem follows. ∎

Finally, we can easily use Theorem 3 to derive a speedup bound for GEDF-VD to schedule MC gang task sets on identical processors, as stated in the following theorem.

Theorem 4. *If any potentially optimal algorithm can schedule a MC gang task set on M unit-speed processors, GEDF-VD is able to schedule the same MC gang task set on M speed-$(\sqrt{5}+1)$ processors.*

Proof. Theorem 3 directly implies that:

> If any potentially optimal algorithm can schedule a MC gang task set on M speed-$(\psi \cdot \Phi/M)$ processors, GEDF-VD is able to schedule the same MC gang task set on M unit-speed processors.

This is because for a MC gang task set to be schedulable on M speed-$(\psi \cdot \Phi/M)$ processors, it is necessary for its corresponding de-ganged MC task set to schedulable on a speed-$(\psi \cdot \Phi)$ uniprocessor. Note that, by definitions: $\psi = \frac{M}{2-\frac{1}{M}}$ and $\Phi = \frac{\sqrt{5}-1}{2}$, the following statement is true:

> If any potentially optimal algorithm can schedule a MC gang task set on M speed-$(\frac{1}{2-\frac{1}{M}} \cdot \frac{\sqrt{5}-1}{2})$ processors, GEDF-VD is able to schedule the same MC gang task set on M unit-speed processors.

Scaling the speed unit up by $(2-\frac{1}{M})\frac{\sqrt{5}+1}{2}$ (please note that $\frac{\sqrt{5}-1}{2} \cdot \frac{\sqrt{5}+1}{2} = 1$), the above statement can be re-written as:

> If any potentially optimal algorithm can schedule a MC gang task set on M unit-speed processors, GEDF-VD is able to schedule the same MC gang task set on M speed-$(2-\frac{1}{M})\frac{\sqrt{5}+1}{2}$ processors.

Since $(2-\frac{1}{M})\frac{\sqrt{5}+1}{2} < \sqrt{5}+1$, the theorem follows. ■

V. Evaluation

In this section, we evaluate the performance of GEDF-VD through simulation results. As our work is the first to propose MC gang task scheduling, there is no perfect baseline for comparison. We have performed many experiments by varying different factors to observe the efficiency of our algorithm.

A. Experimental Setup

Workload generation. We generate MC gang tasks based on the following parameters.

- M : The number of processor cores.
- $m_{min}, m_{max}, m_{avg}$: The minimum, maximum, and average value for m (i.e., degree of parallelism), respectively. We generate the task set by varying these three parameters, where $m_{min}, m_{max} \in [1, M]$ and $m_{min} \leq m_{avg} \leq m_{max}$.
- U_{avg} : The average utilization for the task set. We have varied U_{avg} value from $0.05 \times M$ to $0.95 \times M$ with $0.05 \times M$ difference at each step.
- $P_{\text{HI}} = 0.5$: The probability of a task $\tau_i \in \tau_{\text{HI}}$.
- $R = 4$: Denotes the maximum ratio of u_i^{HI} to u_i^{LO}. u_i^{HI} is generated uniformly from $[u_i^{\text{LO}}, R \times u_i^{\text{LO}}]$.

At first, for a specific value of n (number of tasks per task set), we generate the m values for each task. m is uniformly generated from $[m_{min}, m_{max}]$ range in a way so that the average m for all tasks remains equal to m_{avg}. Next, for a specific value of average utilization U_{avg}, we calculate the average utilization u_i^a for each task by following the log-normal distribution. Note that, for n number of gang tasks, there are total $\sum_{i=1}^{n} m_i = m_{avg} \times n$ amount of single task instances in each task set. For the sake of a proper distribution, we extend the *UUniFast algorithm* [26] for Gang task. We use log-normal distribution over $\sum_{i=1}^{n} m_i$ task instances similarly as UUnifast, but for a single task, we take the average of all of its instances as the task's average utilization. The values of u_i^{LO} is uniformly generated from $[\frac{2 \times u_i^a}{R+1}, u_i^a]$ so that the value of u_i^{HI} is always in the range $[u_i^{\text{LO}}, R \times u_i^{\text{LO}}]$.

TABLE II: Acceptance ratio for different amount of tasks generated under various average utilization

$U_{avg} \rightarrow$ # of tasks$\downarrow$	2	2.5	3	3.5	4	4.5	5	5.5	6
8	100	100	97	59	5	5	4	3	1
12	100	99	94	65	2	2	2	0	0
16	100	100	98	50	0	0	0	0	0

Simulation setup. We performed the simulation for average utilization ranging from $0.05M$ to $0.95M$ with a step size of $0.05M$. For each average, 100 task sets (each with 10 tasks) are generated.

B. Evaluation Results

We execute a set of gang tasks under our proposed algorithm by varying different parameters. We present the simulation results for various scenarios in Figure 3, 4, 5, and in Table II, and report the percentage of the *acceptance ratio* (the ratio of the number of schedulable task sets over the total number of task sets) for each case.

Effect of changing the degree of parallelism in a range with lower difference. In this set of experiments, for $M = 8$, we vary a task's degree of parallelism (m) in a different range. Here, difference between the upper and lower bound in each range is kept equal. The acceptance ratio under varying degree of parallelism is reported in Figure 3. This figure indicates that in boundary cases (where the degree of parallelism is very low or very high) acceptance ratio changes proportionally with respect to the degree of parallelism. This behavior can be explained with the help of Equations (8) and (16). When m increases or decreases by a large amount, acceptance ratio will increase or decrease respectively. However, for a small change of m, acceptance ratio may not change proportionally. This is because, the schedulability conditions provided by Equations (8) and (16) are also effected by the maximum number of idle cores (Δ_i) which is dependent on m.

Effect of changing the total number of cores. In Figure 4, we report the acceptance ratio of the task set by varying the number of cores in the system, M. In this set of experiments, we set a value for m_{avg} which is uniformly generated from a range of $[\frac{M}{2}, \frac{3M}{4}]$. Simulations are conducted for $M = 4, 8, 16$ and 32 and the average utilization is weighted with respect to the value

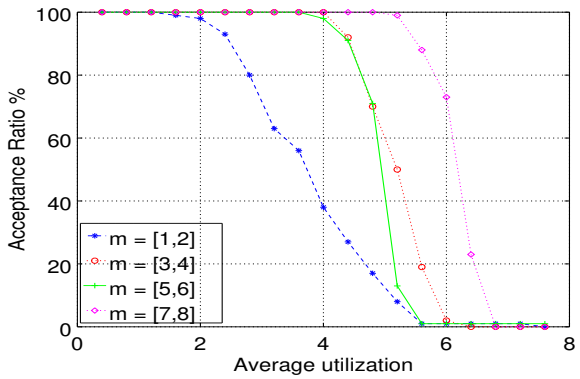

Fig. 3: Acceptance ratio for GEDF-VD in an 8-core platform under same ranges of degrees of parallelism.

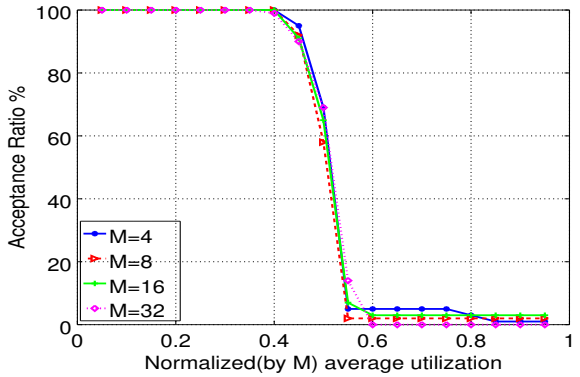

Fig. 4: Acceptance ratio for GEDF-VD in an M-core platform, where M varies.

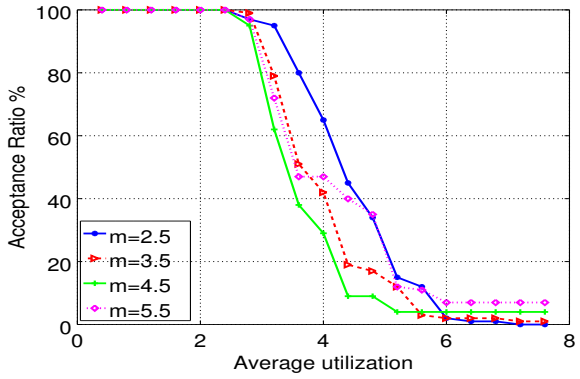

Fig. 5: Acceptance ratio for GEDF-VD in an 8-core platform under different m_{avg}.

of M. Figure 4 shows that the acceptance ratio is not affected by different values of M and remains almost unchanged.

Effect of changing number of tasks per task set. In this set of experiments, we have randomly generated 100 task sets with 8, 12 and 16 tasks per task set (with U_{avg} changing from 2 to 6 with a step size of 0.5) and report the acceptance ratio in Table II. From the reported data, it is clear that acceptance ratio of the task set is not affected by the number of tasks per set. This result indicates the effectiveness of our proposed algorithm under varying number of tasks in a task set.

Effect of changing m_{avg} value. In Figure 5, we show the acceptance ratio by varying m_{avg} in an 8-core platform. This result demonstrates that there may not be any direct relationship between m_{avg} and the acceptance ratio.

VI. Related Work

Since Vestal's first proposal [1] of MC workload model, much work has focused on scheduling MC tasks (refer to Burns et al. [3] for a survey). For the uniprocessor platforms, many algorithms were proposed based on both fixed priority (e.g., Li et al. [27], Baruah et al. [28]) and dynamic priority scheduling(e.g., Easwaran et al. [29]). On the other hand, numerous MC scheduling algorithms were proposed for multiprocessor platforms [30], [31], [32], [33], [34]. Considering the multiprocessor platforms, Lee et al. [30] and Baruah et al. [31] proposed fluid-based MC models, and a semi-partitioned based scheme is proposed by Awan et al. [33].

Considering different parallel tasks models (e.g., synchronous task model [12], DAG model [13], [14], [35], [36], [37] and gang models [7], [8], [9]) there have been a number of works that have provided the energy efficiency technique, schedulability analysis, and the speedup bound (i.e., resource augmentation bound) for various scheduling strategies. For synchronous tasks under GEDF scheduling, Andersson et al. [12] proved a resource augmentation bound of 2 with constrained deadlines tasks. Considering DAG tasks (with arbitrary deadlines) under GEDF, Li et al. [38] and Bonifaci et al. [13] simultaneously proved a resource augmentation bound of 2. Bonifaci et al. [13] also showed the bound to be 3 under global rate-monotonic scheduling. For implicit deadline DAG tasks under federated scheduling, a resource augmentation bound of 2 is showed by Li et al. [14]. Kato et al. [7] introduced gang task scheduling based on global EDF. Dong et al. [9] proposed a schedulability analysis based on lag-based reasoning. Few other related works, such as Goossens et al. [39] provided schedulability tests for fixed task-priority scheduling of real-time periodic gang tasks. Another notable work by Goossens et al. [8] proposed a DP-Fair based scheduling of periodic gang tasks and proved a speedup bound which is no larger than $(2 - 1/m)$.

Although a good number of works studied MC scheduling and parallel tasks scheduling individually, very few works [15], [16], [17], [40] studied the scheduling of MC parallel tasks. Rambo et al. [40] proposed a replica-aware co-scheduling approach (that is a combination of strict priority preemptive (SPP) policy and gang scheduling policy) for mixed-critical systems. Baruah et al. [16] and Li et al. [17] proposed the MC scheduling of DAG models, while Liu et al. [15] proposed the MC scheduling of synchronous task model. Unlike these works, we consider the gang task model, where a task cannot execute if the number of available cores is less than its degree of parallelism. This constraint makes the scheduling problem highly challenging.

VII. Conclusion

Parallel computing with real-time constraints is gaining popularity due to its broad applicability and system efficiency. WCET measurements are pessimistic due to increased uncertainty. So, there is an emerging need to introduce MC into parallel computation models and system designs. In this work, we leverage two existing algorithms (EDF-VD and GEDF) to schedule MC gang tasks efficiently. We derive the first speedup bound for GEDF schedulability of (non-MC) gang tasks and further derived the bound for GEDF-VD of MC gang tasks. This work is an initial step of more substantial efforts in bringing richer system modeling and analysis into the emerging need in many applications for parallel computing and MC. In the future, by ensuring the a priori verification, we plan to consider run-time robustness, with moldable or malleable models for MC gang tasks. Also, we plan to evaluate our results (in this paper) by implementations on applicable hardware platforms.

Acknowledgment

This work is supported by NSF grant CNS-1850851, start-up and REP grants from Texas State University, and research grants council of Hong Kong GRF 15204917 and GRF 15213818.

References

[1] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *RTSS*. IEEE, 2007.
[2] H. Li, "Scheduling mixed-criticality real-time systems," Ph.D. dissertation, The University of North Carolina at Chapel Hill, 2013.
[3] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, pp. 1–69, 2013.
[4] S. Baruah, "Mixed-criticality scheduling theory: Scope, promise, and limitations," *IEEE DESIGN AND TEST*, vol. 35, no. 2, pp. 31–37, 2018.
[5] A. Esper, G. Nelissen, V. Nélis, and E. Tovar, "How realistic is the mixed-criticality real-time system model?" in *RTNS*. ACM, 2015.
[6] R. Ernst and M. Di Natale, "Mixed criticality systemsa history of misconceptions?" *IEEE Design & Test*, vol. 33, no. 5, pp. 65–74, 2016.
[7] S. Kato and Y. Ishikawa, "Gang EDF scheduling of parallel task systems," in *RTSS*. IEEE, 2009.
[8] J. Goossens and P. Richard, "Optimal scheduling of periodic gang tasks," *Leibniz transactions on embedded systems*, vol. 3, no. 1, pp. 04–1, 2016.
[9] Z. Dong and C. Liu, "Analysis techniques for supporting hard real-time sporadic gang task systems," in *RTSS*. IEEE, 2017.
[10] 2017, https://www.openacc.org/.
[11] A. Burns and R. Davis, "A survey of research into mixed criticality systems," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 82, 2018.
[12] B. Andersson and D. de Niz, "Analyzing global-edf for multiprocessor scheduling of parallel tasks," in *OPODIS*. Springer, 2012.
[13] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese, "Feasibility analysis in the sporadic dag task model," in *ECRTS*. IEEE, 2013.
[14] J. Li, J. J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah, "Analysis of federated and global scheduling for parallel real-time tasks," in *ECRTS*. IEEE, 2014.
[15] G. Liu, Y. Lu, S. Wang, and Z. Gu, "Partitioned multiprocessor scheduling of mixed-criticality parallel jobs," in *RTCSA*. IEEE, 2014.
[16] S. Baruah, "The federated scheduling of systems of mixed-criticality sporadic dag tasks," in *RTSS*. IEEE, 2016.
[17] J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill, and C. Lu, "Mixed-criticality federated scheduling for parallel real-time tasks," *Real-Time Systems*, vol. 53, no. 5, pp. 760–811, 2017.
[18] S. Baruah, V. Bonifaci, G. DAngelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *ECRTS*. IEEE, 2012.
[19] S. Baruah, M. Bertogna, and G. Buttazzo, *Multiprocessor Scheduling for Real-Time Systems*. Springer, 2015.
[20] G. A. Elliott, B. C. Ward, and J. H. Anderson, "GPUSync: A framework for real-time GPU management," in *RTSS*. IEEE, 2013.
[21] S. Xiao and W.-c. Feng, "Inter-block GPU communication via fast barrier synchronization," in *IPDPS*. IEEE, 2010.
[22] M. Yang, T. Amert, K. Yang, N. Otterness, J. H. Anderson, F. D. Smith, and S. Wang, "Making OpenVX really" real time"," in *RTSS*. IEEE, 2018.
[23] N. Capodieci, R. Cavicchioli, M. Bertogna, and A. Paramakuru, "Deadline-based scheduling for GPU with preemption support," in *RTSS*. IEEE, 2018.
[24] M. Spuri and G. C. Buttazzo, "Efficient aperiodic service under earliest deadline scheduling." in *RTSS*, 1994, pp. 2–11.
[25] 2017, http://www.nvidia.com/page/home.html.
[26] M. Bolado, H. Posadas, J. Castillo, P. Huerta, P. Sanchez, C. Sánchez, H. Fouren, and F. Blasco, "Platform based on open-source cores for industrial applications," in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 2. IEEE, 2004, pp. 1014–1019.
[27] H. Li and S. Baruah, "An algorithm for scheduling certifiable mixed-criticality sporadic task systems," in *RTSS*. IEEE, 2010.
[28] S. Baruah, V. Bonifaci, G. D'angelo, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "Mixed-criticality scheduling of sporadic task systems," in *ESA*. Springer, 2011.
[29] A. Easwaran, "Demand-based scheduling of mixed-criticality sporadic tasks on one processor," in *RTSS*. IEEE, 2013.
[30] J. Lee, K. Phan, X. Gu, J. Lee, A. Easwaran, I. Shin, and I. Lee, "MC-Fluid: Fluid model-based mixed-criticality scheduling on multiprocessors," in *RTSS*. IEEE, 2014.
[31] S. Baruah, A. Eswaran, and Z. Guo, "MC-Fluid: simplified and optimally quantified," in *RTSS*. IEEE, 2015.
[32] S. Tobuschat and R. Ernst, "Efficient latency guarantees for mixed-criticality networks-on-chip," in *RTAS*. IEEE, 2017.
[33] M. Awan, K. Bletsas, P. Souto, and E. Tovar, "Semi-partitioned mixed-criticality scheduling," in *ARCS*. Springer, 2017.
[34] R. Trüb, G. Giannopoulou, A. Tretter, and L. Thiele, "Implementation of partitioned mixed-criticality scheduling on a multi-core platform," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, p. 122, 2017.
[35] Z. Guo, A. Bhuiyan, A. Saifullah, N. Guan, and H. Xiong, "Energy-efficient multi-core scheduling for real-time DAG tasks," 2017.
[36] A. Bhuiyan, Z. Guo, A. Saifullah, N. Guan, and H. Xiong, "Energy-efficient real-time scheduling of DAG tasks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 5, p. 84, 2018.
[37] Z. Guo, A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, and N. Guan, "Energy-efficient real-time scheduling of DAGs on clustered multi-core platforms," in *RTAS*. IEEE, 2019.
[38] J. Li, K. Agrawal, C. Lu, and C. Gill, "Analysis of global edf for parallel tasks," in *ECRTS*. IEEE, 2013.
[39] J. Goossens and V. Berten, "Gang FTP scheduling of periodic and parallel rigid real-time tasks," *arXiv preprint arXiv:1006.2617*, 2010.
[40] E. A. Rambo and R. Ernst, "Replica-aware co-scheduling for mixed-criticality," in *ECRTS 2017*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.