

A Parallel Algorithm for EDF-Schedulability Analysis of Multi-Modal Real-Time Systems

Masud Ahmed Nathan Fisher Daniel Grosu
Department of Computer Science
Wayne State University, USA.
Email: {masud, fishern, dgrosu}@wayne.edu

Abstract—Modern real-time embedded systems often require the capability of switching between operating modes to adapt in dynamically changing environments. The development of such real-time multi-modal systems fundamentally relies upon effective schedulability analysis. Recently, researchers have proposed serial schedulability analysis algorithms for multi-modal systems that account for mode changes at both software level (e.g., changing the set of executing tasks) and hardware level (e.g., changing the operating speed of a processor). However, these algorithms have high runtime complexity which limits their practical usage as schedulability analysis in system design-space exploration. In this paper, we design a parallel algorithm as an efficient solution to the problem of determining the schedulability of uniprocessor multi-modal real-time systems scheduled by EDF. By emphasizing a balanced workload distribution and restricting the number of synchronizations, our parallel algorithm achieves a near-perfect speedup observable both theoretically and experimentally. Experimental results show that the runtime of our parallel algorithm is very low even for systems with large number of modes, making it a tractable choice for design-space exploration of real-time multi-modal systems.

Keywords: multi-modal systems, parallel algorithms, earliest deadline first, real-time systems.

I. INTRODUCTION

Current embedded system design has increasingly utilized multiple operating modes to allow a higher degree of dynamism. GPS receivers, smart phones, and wireless sensors utilize multi-mode executions to prolong their battery life. Implementation of these multiple modes can be found in software, hardware, or both. Most modern processors typically have Dynamic Power Management (DPM) capabilities such as clock throttling, clock gating, or dynamic voltage and frequency scaling that provide multiple hardware execution modes. At the same time, a software system may change its timing requirements to adapt to different resource constraints (e.g., an adaptive video-streaming application adjusts its processing rate and decoding quality based upon processing/network availability). Schedulability analysis is an important design step for ensuring predictable timing behavior of real-time systems. Unfortunately, most prior research on multi-modal schedulability analysis focuses primarily on software mode changes which make it unsuitable for real-time control systems that require support for both hardware and software mode changes (e.g., real-time thermal-control system by Hettiarachchi et al. [8]). For such real-time control systems,

a larger number of modes is typically desirable to permit greater adaptability in unfavorable environments. Checking schedulability of multi-modal systems warrants higher computation time as a result of dependencies (due to transitions) between modes; therefore, a large number of modes pose a computational challenge for existing algorithms for sequential schedulability analysis techniques. Thus, parallel schedulability analysis is a promising and practical alternative to traditional techniques.

An efficient parallel schedulability analysis can also reduce significantly the time for design-space exploration [22] that may utilize schedulability tests for determining optimized resource parameters of a multi-mode system. Schedulability analysis using parallel algorithms is a relatively unexplored area for multi-modal real-time applications. For uni-modal systems, there have been solutions with well defined sets of conditions where each condition must pass a set of test cases. In most scenarios, the evaluation of these test case elements can be performed independently. From the perspective of parallel computing, this independent execution behavior makes the problem of uni-modal schedulability less challenging. However, the schedulability analysis of multi-modal real-time systems is complex; the analysis not only depends on each mode itself, but also on the schedulability of all other modes along with mode change sequences. Therefore, recently developed pseudo-polynomial schedulability analysis by Fisher and Ahmed [6] may be a workable solution for the schedulability of systems (given a set of system parameters). However, a sequential implementation of their analysis may not sufficiently scale to be used as an effective tool for determining optimal system parameters. In this paper, we address a fundamental gap in the research literature on parallel schedulability analysis algorithms suitable for design-space exploration (e.g., minimizing the total aggregate hardware resources over all modes) for real-time multi-modal systems.

Contribution & Organization. Our proposed solution is the *first non-trivial* parallel schedulability analysis algorithm for real-time multi-mode systems. We provide a parallel algorithm for evaluating the schedulability conditions developed by Fisher and Ahmed [6]. Our contributions are as follows:

- To achieve an ideal workload distribution, we adopt the most suitable and relatively simple workload distribution policy considering the nature of the conditions (Sec-

tion IV). A balanced distribution of workload (without introducing overhead) is the key for achieving better speedup.

- To achieve a near-ideal speed-up, we design the algorithm such that the communication and synchronization overhead is minimized.
- To characterize the effectiveness of the proposed algorithm, we determine parallel performance metrics (e.g., speedup, efficiency, and cost) and derive conditions to obtain better speedup (Section V).
- To substantiate the performance of the proposed algorithm, we perform experiments upon a cluster of AMD Opteron computers (Section VI). We obtain high parallel efficiency (over 90%) which establishes that the proposed algorithm can be used as an efficient schedulability test for design-space exploration of real-time multi-mode systems.

II. RELATED WORK

Research on uniprocessor multi-mode real-time systems can be divided into two categories: fixed-priority scheduling [11], [13], [20] and EDF scheduling [12], [17]. Tindell et al. [21] introduced a simple protocol where newly-added tasks wait after a mode change until the processor completes the old-mode tasks. In another publication, Tindell et al. [20] defined equations for calculating the waiting time after which a new-mode task can generate jobs. Pedro and Burns [11] and Real and Crespo [13] explored protocols where old-mode tasks may execute concurrently with new-mode tasks. Mechanisms such as constant-bandwidth server (CBS) [1], sporadic server [16], periodic resource model [15], elastic scheduling [4], and rate-based earliest deadline (RBED) [3] also permit a system to change its software/hardware requirements adaptively; however, each of these server-based mechanisms does not guarantee deadlines during transitions between modes. An adaptive hard-real-time extension of CBS, called variable-bandwidth server (VBS) [9] has been developed; however, VBS does not consider hardware-level mode changes.

Recently, real-time calculus (RTC) [19] has been increasingly used to analyze multi-mode real-time systems. Using RTC, Stoimenov et al. [17], [18] and Santinelli et al. [14] investigated schedulability of hardware modes and software modes separately; however, scenarios where both hardware and software must change modes were not addressed. One set of recent results by Phan et al. [12] has addressed multiple hardware and software-level modes and proposed a sequential algorithm for schedulability analysis. However, the proposed algorithm requires a reachability search of a state-space graph and does not scale efficiently with the number of modes. Fisher and Ahmed [6] developed invariants that a schedulable multi-mode system must satisfy; however, the proposed solution still requires pseudo-polynomial time. We exploit these schedulability conditions to develop an efficient parallel schedulability analysis of a multi-modal real-time system.

III. MODELS & DEFINITIONS

We consider a multi-mode system $\mathcal{M}$ with q number of modes. Each mode is scheduled by the Earliest-Deadline-First (EDF) scheduling algorithm. A mode $M^{(i)}$, where $i \in \{1, \dots, q\}$ is the mode index, is associated with the real-time workload and the minimum processor execution guaranteed by the processing resource. The real-time workload $\tau^{(i)}$ of mode $M^{(i)}$ is modeled by the *sporadic tasks model* [10], and the processing resource that guarantees the real-time execution of $\tau^{(i)}$ is modeled by the *explicit-deadline periodic (EDP) resource model* [5]. Throughout this paper, we assume that each of the system parameters are integers.

Sporadic Task Model. A *sporadic task system* $\tau^{(i)} \stackrel{\text{def}}{=} \{\tau_1^{(i)}, \dots, \tau_{n_i}^{(i)}\}$ is a collection of n_i sporadic tasks where each task $\tau_\ell^{(i)} \in \tau^{(i)}$ is characterized by three parameters: *worst-case execution requirement* $e_\ell^{(i)}$, *(relative) deadline* $d_\ell^{(i)}$, and *minimum inter-arrival separation* $p_\ell^{(i)}$ (also called *period*). A sporadic task $\tau_\ell^{(i)}$ may produce a sequence of jobs, separated by at least $p_\ell^{(i)}$ time units, where each job has the maximum execution time $e_\ell^{(i)}$ units, and must complete within $d_\ell^{(i)}$ time units after its arrival. We consider *constrained deadline* tasks; that is, $d_\ell^{(i)} \leq p_\ell^{(i)}$. The utilization of a task $\tau_\ell^{(i)}$ is defined as $u_\ell^{(i)} \stackrel{\text{def}}{=} e_\ell^{(i)} / p_\ell^{(i)}$ and the task system utilization $u^{(i)}$ equals $\sum_\ell u_\ell^{(i)}$. To quantify the maximum workload over an interval of length $t > 0$, we consider the *demand-bound function* $\text{dbf}(\tau_\ell^{(i)}, t)$ which quantifies the maximum cumulative execution requirements of all jobs of $\tau_\ell^{(i)}$ that could have both the arrival time and the deadline in any interval of length t . Baruah et al. [2] have shown that, for a sporadic task $\tau_\ell^{(i)}$, $\text{dbf}(\tau_\ell^{(i)}, t)$ can be calculated as follows:

$$\text{dbf}(\tau_\ell^{(i)}, t) = \max \left(0, \left\lfloor \frac{t - d_\ell^{(i)}}{p_\ell^{(i)}} \right\rfloor + 1 \right) \cdot e_\ell^{(i)}. \quad (1)$$

In Figure 1, the dotted line depicts $\text{dbf}(\tau_1^{(i)}, t)$. The horizontal axis represents the interval length and the vertical axis represents the execution requirement. We denote by $\text{dbf}(\tau^{(i)}, t) \stackrel{\text{def}}{=} \sum_{\tau_\ell^{(i)} \in \tau^{(i)}} \text{dbf}(\tau_\ell^{(i)}, t)$ the demand-bound function of a sporadic task system $\tau^{(i)}$.

Explicit-Deadline Periodic (EDP) Resource Model. The *EDP resource model* [5], [15] is a general resource model for

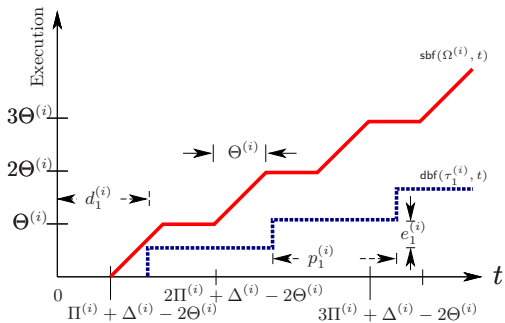


Fig. 1. Demand-bound and supply-bound functions.

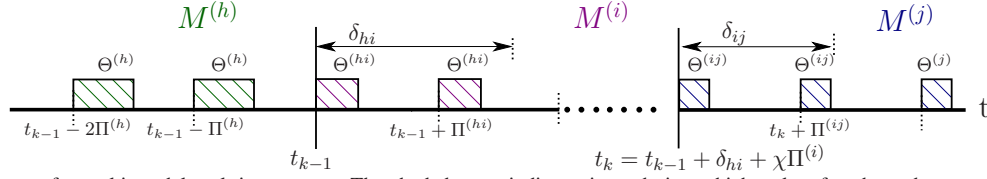


Fig. 2. Execution pattern of a multi-modal real-time system. The shaded areas indicate times during which tasks of each mode execute on the processor.

characterizing the execution of a system upon a periodically-available, non-continuously-executing resource. The hardware processing resource available to each mode $M^{(i)}$ is represented by an EDP resource $\Omega^{(i)} \equiv (\Pi^{(i)}, \Theta^{(i)}, \Delta^{(i)})$, where $\Pi^{(i)}$ is the *resource period*, $\Theta^{(i)}$ is the *resource capacity*, and $\Delta^{(i)}$ is the *resource deadline*. The interpretation of these parameters is that the EDP resource $\Omega^{(i)}$ guarantees a mode $M^{(i)}$ the total execution of at least $\Theta^{(i)}$ units over successive $\Pi^{(i)}$ -length intervals within $\Delta^{(i)} (\leq \Pi^{(i)})$ units of time. Furthermore, we assume that EDF is used to schedule the workload at any point when the resource is providing execution. We consider the *supply-bound function* $\text{sbfb}(\Omega^{(i)}, t)$ to quantify the minimum execution supply that a mode $M^{(i)}$ is guaranteed to receive from $\Omega^{(i)}$ over any interval of length $t \geq 0$. The solid line in Figure 1 presents $\text{sbfb}(\Omega^{(i)}, t)$ for $\Omega^{(i)}$. Easwaran et al. [5] have quantified $\text{sbfb}(\Omega^{(i)}, t)$ as follows:

$$\text{sbfb}(\Omega^{(i)}, t) = \begin{cases} y\Theta^{(i)} + \max(0, t - x - y\Pi^{(i)}), & \text{if } t \geq \Delta^{(i)} - \Theta^{(i)} \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

where $y = \left\lfloor \frac{t - (\Delta^{(i)} - \Theta^{(i)})}{\Pi^{(i)}} \right\rfloor$ and $x = (\Pi^{(i)} + \Delta^{(i)} - 2\Theta^{(i)})$.

Real-Time Mode Change Model. We now describe the *discrete hardware/software real-time multi-mode* model [6]. Each mode is specified by a three-tuple $(\tau^{(i)}, \Omega^{(i)}, N^{(i)})$ which respectively characterizes the real-time workload generated by a sporadic task system, the minimum processor execution guaranteed by an EDP resource, and the minimum mode duration in terms of “number of resource periods” $N^{(i)}$. The interpretation of $N^{(i)}$ is that the system remains in mode $M^{(i)}$ for at least $N^{(i)} \cdot \Pi^{(i)}$ time units. A *mode-change request* $\text{mcr}_k \equiv (M^{(i)}, M^{(j)}, t_k)$ consists of *transition time* (t_k), the *old mode* $M^{(i)}$ executing prior to t_k , and the *new mode* $M^{(j)}$ executing after t_k (where $i, j \in \{1, \dots, q\}$). We assume that if $i < j$ then mcr_i occurs prior to mcr_j . Mode-change request $\text{mcr}_0 \stackrel{\text{def}}{=} (M^{(0)}, \cdot, 0)$ represents the transition from the null-mode $M^{(0)}$ to any mode in $\{M^{(1)}, \dots, M^{(q)}\}$ at startup time.

Tasks may be divided into groups based on their importance at the time of a mode-change request $\text{mcr}_k \equiv (M^{(i)}, M^{(j)}, t_k)$. Some important tasks may need to continue to execute without being affected by the mode-change request (MCR). We call these tasks *unchanged tasks* denoted by $\tau^{(ij)}$. Some less important tasks may be removed from the system immediately at the time of a MCR. We call these tasks *aborted tasks* and denote them by $\alpha^{(ij)}$. For some tasks, immediate termination may leave the system in an inconsistent state; we call such tasks *finished tasks* and characterize them as being members of the set $\tau^{(i)} \setminus (\alpha^{(ij)} \cup \tau^{(ij)})$. We allow a job from a finished task at the time of MCR to complete its remaining

execution.

In order to facilitate quick changes of modes, the system designer may allow a transition period, called the *offset*, which we denote by δ_{ij} (see Figure 2). During the transition period after mcr_k , only the jobs from unchanged tasks and the last generated job from the finished tasks are permitted to execute. At $t_k + \delta_{ij}$ and after, task system $\tau^{(j)}$ may generate and execute jobs along with any remaining execution of jobs from $\tau^{(i)}$. At last, there may be some tasks that are common in both modes, but have some properties changed; we treat these tasks as finished tasks in the old mode. During the transition period after mcr_k , the system designer may provision a different resource $\Omega^{(ij)} \stackrel{\text{def}}{=} (\Pi^{(ij)}, \Theta^{(ij)}, \Delta^{(ij)})$ to achieve a quick mode change response. We assume that the offset δ_{ij} is some multiple of $\Pi^{(ij)}$. Given the above definitions, we may distinguish three phases with respect to a mode-change request $\text{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$ (and the previous request $\text{mcr}_{k-1} = (M^{(h)}, M^{(i)}, t_{k-1})$):

- 1) $[t_{k-1} + \delta_{hi}, t_k)$: jobs of $\tau^{(i)}$ and old-mode jobs of $\tau^{(h)}$ are executed upon $\Omega^{(i)}$;
- 2) $[t_k, t_k + \delta_{ij})$: non-aborted jobs $(\tau^{(i)} \setminus \alpha^{(ij)})$ with remaining execution at t_k may execute upon $\Omega^{(ij)}$;
- 3) $[t_k + \delta_{ij}, t_{k+1})$: non-completed jobs of $(\tau^{(i)} \setminus \alpha^{(ij)})$ at $t_k + \delta_{ij}$ and jobs of $\tau^{(j)}$ execute upon $\Omega^{(j)}$;

In addition, unchanged tasks $(\tau^{(ij)})$ act independently of a mode-change request during $[t_{k-1} + \delta_{hi}, t_{k+1})$. The above task classifications follow the taxonomy by Real and Crespo [13]. In this paper, we restrict the mode-change requests and transition intervals to occur only at period boundaries of the EDP model. This assumption is natural for control systems where mode changes only at sampling periods. That is, for any mode-change request $\text{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$, δ_{ij} must be a multiple of $\Pi^{(ij)}$, and for any two successive mode-change requests $\text{mcr}_{k-1} = (M^{(h)}, M^{(i)}, t_{k-1})$ and $\text{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$, we require that

$$t_k = t_{k-1} + \delta_{hi} + a\Pi^{(i)} \quad (3)$$

for some $a \in \mathbb{N}^+$, where $a \geq N^{(i)}$. We also assume that a non-aborted job may span no more than one mode-change request. In this paper, we consider the following problem.

EDF-Multi-Mode-Sched Problem: Given modes $M^{(1)}, \dots, M^{(q)}$, resources $\Omega^{(ij)}$, offsets δ_{ij} , unchanged tasks $\tau^{(ij)}$, and aborted tasks $\alpha^{(ij)}$ for all $i, j \in \{1, \dots, q\}$ ($i \neq j$), determine whether all jobs, under all legal job arrival sequences and all possible legal mode-change requests according to Equation 3 are EDF-schedulable.

To solve this problem, Fisher and Ahmed [6] developed

schedulability conditions which guarantee that the maximum demand over any interval is always less than the minimum supply received for that interval. For taking mode changes into account, mode-change DBF and SBF functions should be defined with respect to any $\text{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$. In the rest of the paper, we make use of an indicator function $\mu_{\geq 0}(x)$ which is zero if $x < 0$ and is one otherwise; we will also use the notation $(x)_+ \stackrel{\text{def}}{=} \max(0, x)$.

A. Mode-Change SBF

Definition 1 (Pre-MCR-SBF): The function $\text{sbf}_{\text{prior}}(M^{(i)}, t)$ is the minimum execution guaranteed by $\Omega^{(i)}$ prior to mcr_k over the interval $[t_k - t, t_k]$.

Definition 2 (Transition-MCR-SBF): The function $\text{sbf}_{\text{trans}}(M^{(i)}, M^{(j)}, t)$ is the minimum execution during mcr_k guaranteed by $\Omega^{(ij)}$ and $\Omega^{(j)}$ over the interval $[t_k, t_k + t]$.

Definition 3 (Post-MCR-SBF): The function $\text{sbf}_{\text{post}}(M^{(i)}, M^{(j)}, s, t)$ is the minimum execution guaranteed by $\Omega^{(ij)}$ and $\Omega^{(j)}$ over the interval $[t_k + \delta_{ij} - s, t_k + \delta_{ij} + t]$ (for $0 \leq s \leq \delta_{ij}$).

For each of the above defined functions, Fisher and Ahmed [6] derived upper bounds as follows:

$$\text{sbf}_{\text{prior}}(M^{(i)}, t) \geq a\Theta^{(i)} + \min\left(\Theta^{(i)}, \left(t - ((a+1)\Pi^{(i)} - \Theta^{(i)})\right)_+\right). \quad (4)$$

$$\begin{aligned} \text{sbf}_{\text{trans}}(M^{(i)}, M^{(j)}, t) &\geq b\Theta^{(ij)} + c\Theta^{(j)} \\ &+ \min\left(\Theta^{(ij)}, \left(\min(t, \delta_{ij}) - (b\Pi^{(ij)} + \Delta^{(ij)} - \Theta^{(ij)})\right)_+\right) \\ &+ \min\left(\Theta^{(j)}, \left((t - \delta_{ij})_+ - (c\Pi^{(j)} + \Delta^{(j)} - \Theta^{(j)})\right)_+\right) \end{aligned} \quad (5)$$

$$\begin{aligned} \text{sbf}_{\text{post}}(M^{(i)}, M^{(j)}, s, t) &\geq d\Theta^{(ij)} \\ &+ \min\left(\Theta^{(ij)}, \left(s - ((d+1)\Pi^{(ij)} - \Theta^{(ij)})\right)_+\right) \\ &+ f\Theta^{(j)} + \min\left(\Theta^{(j)}, \left(t - (f\Pi^{(j)} + \Delta^{(j)} - \Theta^{(j)})\right)_+\right) \end{aligned} \quad (6)$$

where $a \stackrel{\text{def}}{=} \lfloor \frac{t}{\Pi^{(i)}} \rfloor$, $b \stackrel{\text{def}}{=} \lfloor \frac{\min(t, \delta_{ij})}{\Pi^{(ij)}} \rfloor$, $c \stackrel{\text{def}}{=} \lfloor \frac{(t - \delta_{ij})_+}{\Pi^{(j)}} \rfloor$, $d \stackrel{\text{def}}{=} \lfloor \frac{s}{\Pi^{(ij)}} \rfloor$, and $f \stackrel{\text{def}}{=} \lfloor \frac{t}{\Pi^{(j)}} \rfloor$.

B. Mode Change DBF

Definition 4 (Carry-In Execution): The carry-in execution $\text{ci}(M^{(i)}, M^{(j)})$ from mode $M^{(i)}$ to any other mode $M^{(j)}$ at time t_k is an upper bound on the maximum possible remaining execution (over any legal sequence of MCRs) of non-aborted jobs from mode $M^{(i)}$ for tasks $\tau^{(i)} \setminus \{\tau^{(ij)} \cup \alpha^{(ij)}\}$ at time $t_k + \delta_{ij}$ that arrive prior to t_k and the maximum total execution of unchanged tasks (i.e., $\tau^{(ij)}$) that have arrival before $t_k + \delta_{ij}$.

The $\text{ci}(M^{(i)}, M^{(j)})$ function can be obtained from the convergence of the sequence $\text{ci}_0(M^{(i)}, M^{(j)})$, $\text{ci}_1(M^{(i)}, M^{(j)})$, $\text{ci}_2(M^{(i)}, M^{(j)})$, ... for all $i, j (i \neq j) \in \{1, \dots, q\}$. For any $g \in \mathbb{N}$, $M^{(i)}$, and $M^{(j)}$ ($i \neq j$), the function ci_g is inductively

defined as follows (according to Fisher and Ahmed [6]):

$$\text{ci}_g(M^{(i)}, M^{(j)}) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if } g = 0, \\ \min\left(E_{ij}, F_{ij}\left(\max_{\substack{h=1, \dots, q \\ h \neq i}} \{\text{ci}_{g-1}(M^{(h)}, M^{(i)})\}\right)\right), & \text{if } g > 0. \end{cases} \quad (7)$$

where

$$E_{ij} \stackrel{\text{def}}{=} \sum_{\tau_\ell^{(ij)} \in \tau^{(ij)}} e_\ell^{(ij)} + \left(\sum_{\substack{\tau_\ell^{(i)} \in \tau^{(i)} \setminus \alpha^{(ij)} \\ - \text{sbf}_{\text{trans}}(M^{(i)}, M^{(j)}, \delta_{ij})}} e_\ell^{(i)} + \sum_{\tau_\ell^{(ij)} \in \tau^{(ij)}} \left\lfloor \frac{\delta_{ij}}{p_\ell^{(ij)}} \right\rfloor e_\ell^{(ij)} \right), \quad (8)$$

$$F_{ij}(\zeta) \stackrel{\text{def}}{=} \left(\begin{aligned} &\Psi(M^{(i)}, M^{(j)}, \zeta, \max(\delta_{ij} + d_{\text{max}}^{(ij)}, d_{\text{max}}^{(i)})) \\ &- \sum_{\tau_\ell^{(ij)} \in \tau^{(ij)}} \text{dbf}\left(\tau_\ell^{(ij)}, \max(\delta_{ij} + d_{\text{max}}^{(ij)}, d_{\text{max}}^{(i)}) - \delta_{ij} - p_\ell^{(ij)}\right) \\ &- \text{sbf}_{\text{trans}}(M^{(i)}, M^{(j)}, \delta_{ij}) \end{aligned} \right), \quad (9)$$

$$\Psi(M^{(i)}, M^{(j)}, \zeta, \phi) \stackrel{\text{def}}{=} \sup_{x \geq 0} \{\Psi_x(M^{(i)}, M^{(j)}, \zeta, \phi)\}, \quad (10)$$

$$\begin{aligned} \Psi_x(M^{(i)}, M^{(j)}, \zeta, \phi) &\stackrel{\text{def}}{=} \left\{ \begin{aligned} &\sum_{\tau_\ell^{(i)} \in \tau^{(i)} \setminus \{\tau^{(ij)} \cup \alpha^{(ij)}\}} \left\lfloor \frac{(x - \lambda_\ell^{(i)}(\phi))_+}{p_\ell^{(i)}} \right\rfloor + 1 \times e_\ell^{(i)} \\ &+ \sum_{\tau_\ell^{(i)} \in \alpha^{(ij)}} \left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor \times e_\ell^{(i)} \\ &+ \sum_{\tau_\ell^{(i)} \in \alpha^{(ij)}} \min\left(x - \left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor p_\ell^{(i)}, e_\ell^{(i)}\right) \\ &+ \sum_{\tau_\ell^{(i)} \in \tau^{(ij)}} \text{dbf}(\tau_\ell^{(i)}, x + \phi) \\ &+ \mu_{\geq 0}\left(x - N_i \Pi^{(i)}\right) \cdot \zeta \\ &- \text{sbf}_{\text{prior}}(M^{(i)}, x) \end{aligned} \right\}, \end{aligned} \quad (11)$$

and $\lambda_\ell^{(i)}(t) \stackrel{\text{def}}{=} (d_\ell^{(i)} - t)_+$. The convergence of the above sequence occurs at the smallest $g \in \mathbb{N}$ such that $\forall i, j \in \{1, \dots, q\} \cap i \neq j$ $\text{ci}_g(M^{(i)}, M^{(j)}) = \text{ci}_{g-1}(M^{(i)}, M^{(j)})$. To provide intuition, the function $\text{ci}_g(M^{(i)}, M^{(j)})$ in Equation 7 represents the maximum carry-in from $M^{(i)}$ to $M^{(j)}$ if $g-1$ mode changes have previously occurred. The maximum carry-in may be bounded by the total execution of jobs that are not aborted or are unchanged at the mode change from $M^{(i)}$ to $M^{(j)}$ (i.e., E_{ij} of Equation 8). The maximum carry-in is also bounded by the demand generated by jobs of $M^{(i)}$ accounting for the maximum carry-in from some previous mode change from $M^{(h)}$ (i.e., $\text{ci}_{g-1}(M^{(h)}, M^{(j)})$). The functions F_{ij} and Ψ (Equations 9 and 10) are used to calculate the demand “carried-in” from $M^{(i)}$, as formalized in the next definition.

Definition 5: The carry-in demand-bound function $\text{cidbf}(M^{(i)}, M^{(j)}, \phi)$ for a mode change from $M^{(i)}$ to $M^{(j)}$ at time t_k and $\phi \in \mathbb{R}_{\geq 0}$ is the maximum remaining execution (over any legal sequence of MCRs prior to t_k) of jobs of tasks $\tau^{(i)} \setminus \alpha^{(ij)}$ that arrive prior to t_k (or prior to $t_k + \delta_{ij}$ for $\tau^{(ij)}$ tasks) and have deadlines in the interval $[t_k, t_k + \phi]$.

It may be shown [6] that $\Psi(M^{(i)}, M^{(j)}, \text{ci}(M^{(i)}, M^{(j)}), \phi)$ is an upper bound on the carry-in demand-bound function $\text{cidbf}(M^{(i)}, M^{(j)}, \phi)$ for any mode change from $M^{(i)}$ to $M^{(j)}$ over any interval of length ϕ .

C. Schedulability Conditions

Using the definitions provided in the previous sub-sections, Fisher and Ahmed [6] developed schedulability conditions for multi-mode systems as follows. Over any possible (legal) sequence of mode-change requests, the system is EDF-schedulable, if the following five conditions hold for any two distinct modes $M^{(i)}$ and $M^{(j)}$,

$$\begin{aligned}
\text{SC}_1 : \quad & \sum_{\tau_\ell^{(j)} \in \tau^{(j)}} \text{dbf}(\tau_\ell^{(j)}, t) \leq \text{sbf}(\Omega^{(j)}, t), \quad \forall t \in T^{\text{SC}_1(j)}; \\
\text{SC}_2 : \quad & \sum_{\tau_\ell^{(j)} \in \tau^{(j)} \setminus \tau^{(ij)}} \text{dbf}(\tau_\ell^{(j)}, t) + \sum_{\tau_\ell^{(ij)} \in \tau^{(ij)}} \text{dbf}(\tau_\ell^{(ij)}, t + s), \\
& \leq \text{sbf}_{\text{post}}(M^{(i)}, M^{(j)}, s, t), \\
& \quad \forall s, t : (0 < s \leq \delta_{ij}) \wedge (t \in T^{\text{SC}_2(i,j,s)}); \\
\text{SC}_3 : \quad & \sum_{\tau_\ell^{(ij)} \in \tau^{(ij)}} \text{dbf}(\tau_\ell^{(ij)}, t) \leq \text{sbf}(\Omega^{(ij)}, t), \quad \forall t \in T^{\text{SC}_3(i,j)}; \\
\text{SC}_4 : \quad & \Psi_x(M^{(i)}, M^{(j)}, C_i, \phi) \leq \text{sbf}_{\text{trans}}(M^{(i)}, M^{(j)}, \phi), \\
& \quad \forall \phi, x : (0 < \phi \leq \delta_{ij}) \wedge (x \in T^{\text{SC}_4(i,j,\phi)}); \\
\text{SC}_5 : \quad & \sum_{\tau_\ell^{(j)} \in \tau^{(j)} \setminus \tau^{(ij)}} \text{dbf}(\tau_\ell^{(j)}, t) + \Psi_x(M^{(i)}, M^{(j)}, C_i, \delta_{ij} + t) \\
& - \text{sbf}_{\text{trans}}(M^{(i)}, M^{(j)}, \delta_{ij}) \leq \text{sbf}_{\text{post}}(M^{(i)}, M^{(j)}, 0, t), \\
& \quad \forall t, x : (0 < t \leq \mathcal{T}_{ij}) \wedge (x \in T^{\text{SC}_5(i,j,t)}); \quad (12)
\end{aligned}$$

where $C_i \stackrel{\text{def}}{=} \max_{h=1, \dots, q} \{ \text{ci}(M^{(h)}, M^{(i)}) \}$. $T^{\text{SC}_1(i)}$, $T^{\text{SC}_2(i,j,s)}$, $T^{\text{SC}_4(i,j,\phi)}$, $T^{\text{SC}_5(i,j,t)}$, and $\mathcal{T}_{ij}$ are each a finite set of consecutive positive integers starting from one. Each of these sets are commonly referred as a *testing-set*. We use a generic notation of $\text{SC}_Z(i, j, \phi)$ where $Z \in \{1, \dots, 5\}$ for the superscript of the testing sets. For example, the $\text{SC}_1(i, \emptyset, \emptyset)$ is the superscript for $T^{\text{SC}_1(i)}$ which is the testing set of SC_1 in Equation 12. The last two parameters in this example have the value of $\emptyset$ as they are not used by SC_1 . The largest integer of each testing set $T^{\text{SC}_Z(i,j,\ell)}$ and $\mathcal{T}_{ij}$ is finite and determined by Fisher and Ahmed [6].

Now, we provide intuitive explanations for each condition of Equation 12. Before missing a deadline by an EDF-schedule, the processor is continuously busy. This interval is known as a busy interval. In the busy interval, the resource demand is greater than the processing supply. Five conditions are used to avoid busy intervals where resource demand is greater than

the supply taking mode changes into account. Fisher and Ahmed [6] identified the five different kinds of busy intervals with respect to a mode-change request, which are depicted in Figure 3. SC_1 ensures the schedulability of an individual mode. SC_2 and SC_3 ensure that an individual mode is schedulable along with the demand from unchanged tasks of the old mode after a mode change. SC_4 ensures the schedulability during the transition period while accounting for the carry-in demand from the non-aborted jobs and the mode-change supply function. Similarly, SC_5 ensures schedulability after a transition. The last two conditions account for the carry-in from all past mode-change requests through the ci function while analyzing demand of each individual mode.

IV. PARALLEL ALGORITHM

We design a parallel algorithm for solving the EDF-Multi-Mode-Sched Problem. We first determine the complexity of a serial algorithm for checking all five conditions to realize the size of the problem to be parallelized. The runtime complexity depends on the total aggregate size of all the testing sets. Furthermore, conditions SC_4 and SC_5 require evaluating the ci function to account for the carry-in execution. Fisher and Ahmed [6] showed that the $\text{ci}(M^{(i)}, M^{(j)})$ function, defined iteratively from the sequence $\text{ci}_0(M^{(i)}, M^{(j)})$, $\text{ci}_1(M^{(i)}, M^{(j)})$, $\text{ci}_2(M^{(i)}, M^{(j)})$, $\dots$ for all $i, j (i \neq j) \in \{1, \dots, q\}$, can be calculated in a finite number of iterations which is equal to the summation of execution requirements of all tasks. We define $\mathcal{C}$ as the maximum of the following three values: 1) the summation of the execution requirements of all tasks over all modes, 2) the maximum transition period (i.e., maximum δ_{ij}) and 3) the maximum of the sizes of the sets $T^{\text{SC}_Z(i,j,\phi)}$ and $\mathcal{T}_{ij}$ for any i, j, ϕ and Z . The calculation of the $\text{ci}(M^{(i)}, M^{(j)})$ for each pair requires at most $\mathcal{C}$ inductive computations of ci_g where each such computation would invoke the Ψ_x function at most $\mathcal{C}$ times. As there are $q(q-1)$ pairs of modes, a serial function for calculating the carry-in for all pairs would require $O(q^2 n \mathcal{C}^2)$ time, where n is $\max_{i=1}^q \{n_i\}$; the term n is due to the calculation of demand (dbf) for every testing set element. The complexity of checking all five conditions is dominated by the complexity of checking condition SC_5 which is also $O(q^2 n \mathcal{C}^2)$; therefore, the complexity of the serial schedulability analysis is $O(q^2 n \mathcal{C}^2)$. This pseudo-polynomial complexity could be quite large as $\mathcal{C}$ is potentially exponential in the representation of the multi-modal system. Thus, it is desirable to decrease the analysis time by parallelizing the schedulability analysis.

A. Parallel Platform

We consider a *parallel message-passing system* composed of m identical processors, $P = \{P_1, P_2, \dots, P_m\}$ where the subscript $i \in \{1, \dots, m\}$ for each processor P_i denotes the unique identifier (frequently denoted as *rank*) in the platform. The design of our parallel algorithm considers the data parallel model in which the total workload (testing-set elements) is statically mapped onto processors and each

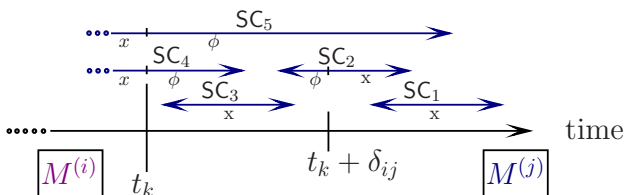


Fig. 3. Busy intervals for each of the schedulability conditions SC_Z .

processor performs similar operations on different testing-set elements. For communication/synchronization, we use the parallel message-passing construct All-to-All-Reduction [7] by which all processors simultaneously involve in a communication/synchronization operation. The All-to-All-Reduction uses an associative operator (e.g., MAX, SUM, OR) to accumulate and combine the data from the buffer of each processor into a single piece of data which is then replicated at all processors.

The performance of a parallel algorithm depends heavily on the underlying workload distribution policy. Balanced distribution of workload along with the minimal overhead due to communication/synchronization is indispensable to reduce the parallel execution time. In the next sub-section, we describe the workload distribution that allows us to obtain a completely balanced workload distribution without any communication/synchronization overhead. Then, we present the parallel algorithm for schedulability analysis and finally characterize its theoretical performance.

B. Workload Distribution

We develop policies to distribute elements of each testing set $T^{\text{SC}_Z(i,j,\phi)}$ among the processors for the parallel algorithm. Our approach emphasizes a balanced distribution of workload among the processors to obtain a near-ideal speedup. Since the workload for the parallel algorithm is entirely dependent upon the testing sets for the schedulability conditions of Equation 12, a naive approach is to assign elements of each testing set $T^{\text{SC}_Z(i,j,\phi)}$ in a round-robin fashion to each of the processors; i.e., processor P_1 would test schedulability condition SC_Z for the first testing-set element, processor P_2 would test SC_Z for the second testing-set element, and so on. In general, after evaluating a testing set element t_i , processor P_k will skip the next m testing set elements which implies that each processor will work with a single element among the m consecutive members of $T^{\text{SC}_Z(i,j,\phi)}$. Thus, the total number of elements to be checked by each processor is at least $\left\lfloor \frac{|T^{\text{SC}_Z(i,j,\phi)}|}{m} \right\rfloor$ and the maximum difference in workload between two successive processors is one testing element. Now consider the next testing set; again, if the first processor P_1 tests the first element, then, in the worst case this processor may receive one more testing set element than the other processors. Therefore, at the end of checking condition $\text{SC}_Z(i,j,\phi)$ of Equation 12, there may be a difference in workload of one testing element among processors. At the end of the execution, this difference could be equal to the total number of testing sets for all five conditions (which is Cq^2). This uneven workload will reduce the speedup of the parallel algorithm. Thus, in our approach we do not always allow the first processor P_1 to test the first element of the testing set, rather we keep track of the processor P_k that tests the last element in previous testing set. Then, we allow the next processor P_{k+1} to test the first element in the current testing set. To support the equal distribution of testing set elements, each processor maintains a root distribution variable r_k , where $r_k \in \{1, 2, \dots, q\}$. The variable r_k indicates the

starting element for P_k for the next testing set. The set below represents the subset of elements of $T^{\text{SC}_Z(i,j,\phi)}$ for which P_k is responsible for testing the condition SC_Z while P_k 's root variable is r_k .

$$T_{k,r_k}^{\text{SC}_Z(i,j,\phi)} = \{x | (x \in T^{\text{SC}_Z(i,j,\phi)}) \wedge (r_k \equiv x \pmod{m})\} \quad (13)$$

We now consider how to update the root variables to ensure a completely-balanced distribution of the testing set elements. For processor P_k with root variable r_k , we can determine the processor that has r_ℓ equal to one for some $\ell \in \{1, \dots, m\}$. The expression $r'_k \stackrel{\text{def}}{=} ((k - r_k) \pmod{m}) + 1$ identifies the rank of this processor. By distributing each of the elements of $T^{\text{SC}_Z(i,j,\phi)}$ in a round-robin fashion (as described in Equation 13), the first processor to receive an element has rank equal to $\ell' \stackrel{\text{def}}{=} ((r'_k + |T^{\text{SC}_Z(i,j,\phi)}| - 1) \pmod{m}) + 1$. Rank ℓ' identifies the processor that will receive the first element in the next testing set distribution. Thus, for any other processor P_k to determine its new root variable, we must calculate $((k - \ell') \pmod{m}) + 1$. Thus, we must use the following update rule:

$$r_k = \left(\left(k - \left((r'_k + |T^{\text{SC}_Z(i,j,\phi)}| - 1) \pmod{m} \right) + 1 \right) \pmod{m} \right) + 1. \quad (14)$$

After distributing the entire workload of all testing sets according to this rule, the difference between any two processors with respect to the number of testing set elements assigned is at most one. In addition to a completely-balanced workload distribution, we observe that the testing set elements do not need to be distributed (via communication or initialization) to the processors. In fact, since the testing set simply consists of consecutive integers, each processor independently generates testing set elements as needed, according to the set defined in Equation 13. Thus, the proposed distribution eliminates the communication overhead due to the workload distribution.

C. Algorithm Description

We now present the pseudocode for **ParallelSA**, our proposed parallel algorithm for schedulability analysis, in Algorithm 1. The algorithm is designed to run concurrently on all available processors. The **ParallelSA** uses two sub-routines **Validate** and **MaxCarry**. The algorithm starts with the initialization of the parallel execution. The rank of the processor (denoted by k) and the total number of processors are determined at this point. A data distribution root r_k , associated with each processor P_k , is initialized to the unique rank k of the processing platform. The algorithm then starts checking each testing set $T^{\text{SC}_Z(i,j,\phi)}$ of condition $\text{SC}_Z(i,j,\phi)$ for all legal values of i, j and Z using the function **Validate**(r_k, Z, i, j, ϕ, C_i) at each processor P_k . If the function **Validate** returns true for the current testing set, the function will continue its execution to the next testing set; otherwise, the algorithm returns false; that is, the multi-mode real-time system $\mathcal{M}$ is not schedulable.

Algorithm 1 ParallelSA($\mathcal{M}$)

```
1: {Processor  $k$  executes:}
2:  $r_k \leftarrow \text{Initialize}()$ 
3: for  $i = 1$  to  $q$  do
4:   if  $\text{Validate}(r_k, 1, i, \emptyset, \emptyset, \emptyset) = \text{false}$  then
5:     return false
6:   end if
7:   for  $j = 1$  to  $q$  ( $j \neq i$ ) do
8:     if  $\text{Validate}(r_k, 3, i, j, \emptyset, \emptyset) = \text{false}$  then
9:       return false
10:    end if
11:    for  $s = 0$  to  $\delta_{ij}$  do
12:      if  $\text{Validate}(r_k, 2, i, j, s, \emptyset) = \text{false}$  then
13:        return false
14:      end if
15:    end for
16:  end for
17:  $\zeta \leftarrow \text{MaxCarry}(\mathcal{M}, k, r_k)$ 
18: for  $i = 1$  to  $q$  do
19:    $C_i \leftarrow \max_{\{h=1, \dots, q\} \wedge h \neq i} \{\zeta_{hi}\}$ 
20:   for  $j = 1$  to  $q$  ( $j \neq i$ ) do
21:     for  $\phi = 0$  to  $\delta_{ij}$  do
22:       if  $\text{Validate}(r_k, 4, i, j, \phi, C_i) = \text{false}$  then
23:         return false
24:       end if
25:     end for
26:   end for
27:   for all  $t = 0$  to  $\tau_{ij}$  do
28:     if  $\text{Validate}(r_k, 5, i, j, t, C_i) = \text{false}$  then
29:       return false
30:     end if
31:   end for
32: end for
33: end for
34: return true
```

The evaluation of inequalities related to condition SC_1 of Equation 12 is performed in Lines 4 to 6. The condition SC_1 is evaluated for each of the q different modes. All remaining four conditions of Equation 12 are defined for pairs of modes; therefore, we use two nested *for*-loops to iterate through the testing sets associated with each such pair of modes. However, we separate the code segment related to conditions SC_4 and SC_5 (Lines 19 to 33) from the rest as the former two conditions require pre-computed carry-in executions calculated by the **MaxCarry** function (Line 18). The function **MaxCarry** could potentially be invoked at the beginning of the algorithm; in that case, all five conditions could be evaluated using one single block of nested loop. However, the function **MaxCarry** is a costly operation, and we allow its execution only if it is required. For unschedulable systems, it may be the case that the system will not satisfy one of the first three conditions: SC_1 , SC_2 , or SC_3 ; therefore, there is no need of invoking the costly **MaxCarry** for such unschedulable systems.

The algorithm **Validate** is a case-based implementation for evaluating each condition $\text{SC}_Z(i, j, \phi)$ using the condition variable Z . Depending on the value of $Z \in \{1, 2, 3, 4, 5\}$, the function selects the appropriate schedulability condition for each testing set element $x \in T_{k, r_k}^{\text{SC}_Z(i, j, \phi)}$. The per-processor testing set $T_{k, r_k}^{\text{SC}_Z(i, j, \phi)}$ is decided by its current data-distribution

Algorithm 2 $\text{Validate}(r_k, Z, i, j, \phi, C_i)$.

```
1: result  $\leftarrow$  true
2: for all  $x$  in  $T_{k, r_k}^{\text{SC}_Z(i, j, \phi)}$  do
3:   if  $Z = 1$  then
4:     if  $\text{sbfb}(\Omega^{(i)}, x) < \text{dbfb}(\tau^{(i)}, x)$  then
5:       result  $\leftarrow$  false; break;
6:     end if
7:   else if  $Z = 2$  then
8:     if  $\text{dbfb}(\tau^{(j)} \setminus \tau^{(ij)}, x) + \text{dbfb}(\tau^{(ij)}, x + \phi) > \text{sbfb}_{\text{post}}(M^{(i)}, M^{(j)}, \phi, x)$  then
9:       result  $\leftarrow$  false; break;
10:    end if
11:   else if  $Z = 3$  then
12:     if  $\text{dbfb}(\tau^{(ij)}, x) > \text{sbfb}(\Omega^{(ij)}, x)$  then
13:       result  $\leftarrow$  false; break;
14:     end if
15:   else if  $Z = 4$  then
16:     if  $\Psi_x(M^{(i)}, M^{(j)}, C_i, \phi) > \text{sbfb}_{\text{trans}}(M^{(i)}, M^{(j)}, \phi)$  then
17:       result  $\leftarrow$  false; break;
18:     end if
19:   else if  $Z = 5$  then
20:      $\text{carry} \leftarrow (\Psi_x(M^{(i)}, M^{(j)}, C_i, \delta_{ij} + \phi) - \text{sbfb}_{\text{trans}}(M^{(i)}, M^{(j)}, \delta_{ij}))_+$ 
21:     if  $\text{carry} + \text{dbfb}(\tau^{(i)} \setminus \tau^{(ij)}, \phi) > \text{sbfb}_{\text{post}}(M^{(i)}, M^{(j)}, 0, \phi)$  then
22:       result  $\leftarrow$  false; break;
23:     end if
24:   end if
25: end for
26: Update  $r_k$  using Equation 14.
27: return All-to-All-Reduce(result, AND)
```

root r_k . As shown in Equation 13, this is an ordered set of evenly separated (of size m) positive integers starting from r_k ; therefore, we allow each processor to generate its dataset associated with each inequality to reduce the overhead related to data distribution. After validating the testing set, the data distribution root r_k at the processor P_k is updated using Equation 14. The function **Validate** synchronizes schedulability results with all other executing processors using an All-to-All-Reduce operation with AND as the reduction operator. The **Validate** returns false even if there is a single violation.

The function **MaxCarry** evaluates the sequence $\text{ci}_0(M^{(i)}, M^{(j)})$, $\text{ci}_1(M^{(i)}, M^{(j)})$, $\text{ci}_2(M^{(i)}, M^{(j)})$, $\dots$ using a *repeat-until* loop to calculate the carry-in $\text{ci}(M^{(i)}, M^{(j)})$ for all pairs of modes and stores all the carry-in executions in a $q \times q$ -matrix ζ . For all pairs, we calculate $\text{ci}_g(M^{(i)}, M^{(j)})$ at each step g from the value calculated at $(g-1)$ -th step, and store the value in ζ_{ij} only if the new value is greater than the previous one. The function marks the change by setting the **change** flag to true. The newly calculated matrix ζ is synchronized using a All-to-All-Reduce operation with a MAX operator for each individual q^2 cell items. The function proceeds to the next step if there is a change in previously calculated carry-in executions (the **change** is true). The algorithm uses a All-to-All-Reduce operation with an OR operator to determine whether the **change** is set to true by at least one processor. The function proceeds to next step only if the **change** has a true value after the synchronization. Otherwise, the function returns with current values stored in $q \times q$ -matrix ζ .

Finally, if the execution of the algorithm reaches Line 34 of

Algorithm 3 MaxCarry($\mathcal{M}, k, r_k$).

```
1: {Returns a  $[q \times q]$  matrix  $\zeta$ .}
2:  $\zeta \leftarrow 0$ 
3: Compute  $E_{ij}$  using Equation 8.
4: repeat
5:   change  $\leftarrow$  false
6:   for  $i = 1$  to  $q$  do
7:      $\text{ci}_{\max}^{(i)} \leftarrow \max_{\substack{h=1, \dots, q \\ h \neq i}} \{\zeta_{hi}\}$ 
8:     for  $j = 1$  to  $q$  do
9:        $c \leftarrow 0$ 
10:       $d \leftarrow \max(\delta_{ij} + d_{\max}^{(ij)}, d_{\max}^{(i)})$ 
11:      for all  $x$  in  $T_{k, r_k}^{\text{SC}_4(i, j, \phi)}$  do
12:         $c \leftarrow \max\left(c, \Psi_x(M^{(i)}, M^{(j)}, \text{ci}_{\max}^{(i)}, d)\right)$ 
13:      end for
14:      Update  $r_k$  using Equation 14.
15:       $c \leftarrow c - \text{sbf}_{\text{trans}}(M^{(i)}, M^{(j)}, \delta_{ij})$ 
16:       $- \sum_{\tau_{\ell}^{(ij)} \in \tau^{(ij)}} \text{dbf}\left(\tau_{\ell}^{(ij)}, d - \delta_{ij} - p_{\ell}^{(ij)}\right)$ 
17:      if  $\min(c, E_{ij}) > \zeta_{ij}$  then
18:         $\zeta_{ij} \leftarrow \min(c, E_{ij})$ 
19:        change  $\leftarrow$  true
20:      end if
21:    end for
22:  end for
23:  All-to-All-Reduce(change, OR)
24:  All-to-All-Reduce( $\zeta$ , MAX)
25: until change = false
26: return  $\zeta$ 
```

ParallelSA, we may safely declare that a multi-mode system $\mathcal{M}$ is EDF-schedulable.

V. PERFORMANCE METRICS

We investigate the asymptotic performance of our proposed parallel algorithm by using well known parallel performance metrics which include *parallel execution time*, *speedup*, *efficiency*, and *cost* [7]. The parallel execution time, denoted by T_m , is the time elapsed between the start and the end of a parallel computation. The value of T_m depends on the actual workload, number of processors m , and the parallel overhead. T_m decreases at a slower rate as m increases. However, the parallel overhead also increases with m ; therefore, after certain value of m , T_m may not experience a noticeable decrease. T_m for our algorithm is given by

$$T_m = O\left(\frac{\mathcal{C}^2 q^2 n}{m} + \mathcal{C} q^2 m + n\right) \quad (15)$$

The first term corresponds to the actual amount of work performed by each of the processors; it is obtained by dividing the total serial workload by the number of processors. The second term represents the overhead due to communication. The communication operation used in our algorithm is the All-to-All reduction among m processors which has a complexity of $O(mk)$, where k is the size of the message on which reduction is performed [7]. Since the reduction operation is invoked $O(\mathcal{C})$ times with a message size of q^2 (line of Algorithm 3) the overhead due to communication is given

by $O(\mathcal{C} q^2 m)$. The third term represents the overhead due to workload imbalances; per the discussion after Equation 14, the difference, between any two processors, in the number of testing set elements is at most one. This testing set element requires $O(n)$ to evaluate any SC_Z .

The speedup S is defined as the ratio of the serial execution time T_s of the best sequential algorithm to the parallel execution time. That is, $S \stackrel{\text{def}}{=} \frac{T_s}{T_m}$. The perfect speedup for a parallel algorithm equals to m which may be difficult to achieve for algorithms that require communication/synchronization for the correct operation. Due to overhead, S may decrease as m increases. The speedup our algorithm is given by:

$$S \approx \frac{\mathcal{C}^2 q^2 n}{\frac{\mathcal{C}^2 q^2 n}{m} + \mathcal{C} q^2 m + n} \quad (16)$$

Efficiency, denoted by E , is a measure of the fraction of time for which the processors are usefully employed in solving the problem; it is defined as the ratio of the speedup to the number of processors, $E \stackrel{\text{def}}{=} \frac{S}{m}$. The efficiency accounts for the parallel overhead, and usually decreases as m increases. The efficiency of our proposed algorithm is given by:

$$E \approx \frac{\mathcal{C}^2 q^2 n}{m\left(\frac{\mathcal{C}^2 q^2 n}{m} + \mathcal{C} q^2 m + n\right)} \quad (17)$$

As mentioned in the previous sections, the last two terms in the denominator of Equations 16 and 17 are due to the parallel overhead which increases with m . As long as the overhead is smaller than the time required to perform the actual computation, the parallel algorithm remains scalable. We now determine conditions to ensure the scalability of ParallelSA using the concept of cost and cost-optimality. The cost is the sum of the time that each processor spends solving the problem, including the time to perform the actual work and the overhead due to communication/synchronization. A parallel algorithm is cost-optimal [7] if the cost has the same growth as the execution time of the fastest known serial algorithm. The following theorem develops conditions to restrict the parallel overhead:

Theorem 1: ParallelSA is cost-optimal if $m = O(\sqrt{\mathcal{C}n})$.

Proof: By definition, the cost of a parallel algorithm is the product of the parallel execution time and the number of processors. The cost of our parallel algorithm is given by:

$$\text{cost} = \mathcal{C}^2 q^2 n + \mathcal{C} q^2 m^2 + nm \quad (18)$$

For the problem considered in this paper, the execution time of the fastest known serial algorithm is $O(\mathcal{C}^2 q^2 n)$. As the growth for the first term of Equation 18 is the same as the growth of the execution time of the fastest serial algorithm, our parallel algorithm is cost-optimal if the second and the third term have the same growth as $O(\mathcal{C}^2 q^2 n)$; that is, $m^2 = O(\mathcal{C}n)$ (for the second term) and $m = O(\mathcal{C}^2 q^2)$ (for the last term). A reasonable assumption of $n \leq \mathcal{C}$ holds since $\mathcal{C}$ is an upper bound on the execution of tasks and each task has an execution of at least one. Since $n\mathcal{C}$ grows slower than $\mathcal{C}^2 q^2$,

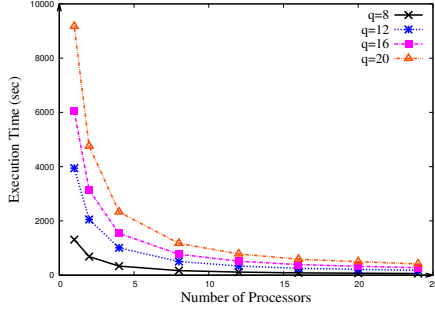


Fig. 4. Execution time vs. number of processors.

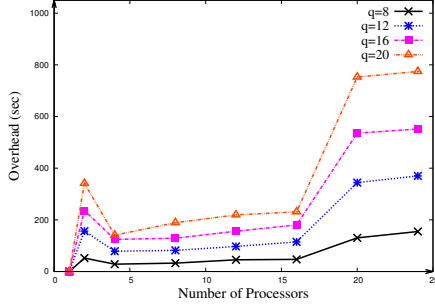


Fig. 5. Parallel overhead vs. the number of processors.

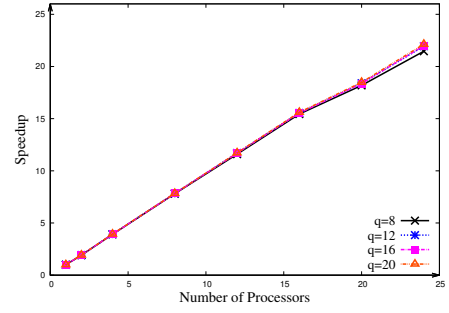


Fig. 6. Speedup vs. number of processors.

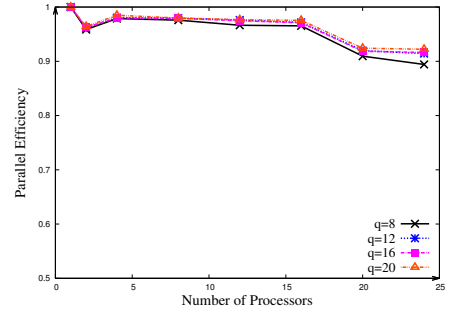


Fig. 7. Efficiency vs. number of processors.

the algorithm is cost-optimal if the first condition is satisfied; that is $m = O(\sqrt{Cn})$ which implies that as long as m grows slower than $\sqrt{Cn}$, ParallelISA remains cost-optimal; thus, the theorem follows. ■

From the analysis in the previous paragraph, it is evident that whenever m grows slower than $\sqrt{Cn}$, the overhead of the ParallelISA algorithm is less than $O(C^2 q^2 n)$. It may be also shown that the speedup of ParallelISA for a computationally large problem ($C \gg n$) is close to m (near-perfect speedup) as the first term in the denominator of Equation 16 dominates for a larger C .

VI. EXPERIMENTAL RESULTS

We perform experiments on a cluster of AMD Opteron computers which is part of the Wayne State University grid. Each computer has two 2.4GHz dual core processors and 4 or 16GB of RAM. The computers are connected through a Gbit Ethernet. We used MPICH-1.2.7 as the standard message passing interface. Value ranges for the parameters of the multi-modal system are listed in Table I. We have previously established the efficacy of the schedulability analysis in [6] over the previous state-of-the-art (Phan et al. [12]); therefore, we measure the efficiency of ParallelISA in this paper.

For the simulation, we generate a set of 12 tasks from the parameters described in the Table I. Of the generated tasks, three are unchanged tasks and two are aborted tasks. We select at least eight tasks from the set for each mode $M^{(i)}$. The resource parameter of a mode is set based on the parameters described in the Table I. In order to check the performance, we considered multi-mode systems with varying number of modes $q \in \{8, 12, 16, 20\}$. The ParallelISA algorithm, for each multi-mode system, is executed at least five times to reduce the effect

on the execution time due to interference from other jobs in the grid. Among them, we took the minimum execution time for each multi-mode system. While checking the schedulability, the ParallelISA algorithm uses a total number of processors from the range $[1, 24]$. In Figure 4, we present the execution time of the ParallelISA algorithm for parallel systems with various numbers of processors. In this figure, the horizontal axis represents the total number of processors m while the vertical axis represents the execution time. Clearly, ParallelISA requires a smaller execution time for a larger number of processors. Note that the decrease in the execution time with the higher number of processors is not linear. This is due to the parallel overhead of our algorithms.

To calculate the overhead of the parallel execution, we consider the algorithm called SUBI (Schedulability Using Bounded Iteration) proposed by Fisher and Ahmed [6] as the best known serial algorithm for the problem. We execute

TABLE I
EXPERIMENT SETUP

Parameter	Value
No. of Modes (q)	$\{8, 12, 16, 20\}$
n_i	8
Range of $p_\ell^{(i)}$	$[2500, 6000]$
Deadline $d_\ell^{(i)}$	$p_\ell^{(i)} \times 0.8$
Utilization $u^{(i)}$	0.45
$\Pi^{(i)}$	1250
$\Theta^{(i)}$	$1.5u^{(i)}\Pi^{(i)}$
$\Delta^{(i)}$	$\Pi^{(i)}$
δ_{ij}	1250
$N^{(i)}$	2
No. of Processors	$\{1, \dots, 24\}$

this algorithm for each multi-mode system using the same hardware resource setting in the grid. The overhead T_o is calculated using the formula $mT_m - T_s$, where T_s is the execution time of SUBI. Figure 5 shows the overhead versus the number of processors used. Like most parallel algorithms, the parallel overhead of our proposed algorithm increases with q due to the increased communication/synchronization cost. However, the overhead does not obtain a noticeable increase after a certain limit on m . This limit depends on the number of modes (e.g., for $q = 8$, the limit is 20).

Figure 6 and Figure 7 show the parallel performance metrics for the ParallelSA. The speedup is calculated with respect to the execution time of SUBI. Figure 6 shows the speedup with respect to the number of processors. The speedup is close to the number of processors. Although it is not discernible in Figure 6, the speedup is slightly better for a larger number of modes. In Figure 7, we present the efficiency of the ParallelSA with respect to the total number of processors used. The parallel efficiency is calculated from the speedup and the number of processors used. The efficiency varies between 90–98% in our experiments. Like the speedup factor, efficiency varies with the number of modes and parameters associated with each mode. We obtain better efficiency for higher number of modes. One possible explanation could be the amount of workload to share among processors which increases with the number of modes.

In Figures 6 and 7, there are spikes at $m = 2$ due to the higher interference from other running jobs in grid. Each computer node in the grid has four cores, and the grid job scheduler assigns a single core for each processor requested unless explicitly specified. For $m = 2$, only two cores of a computer node are used by the ParallelSA, and the remaining two cores may be utilized by other running jobs in the grid. Cores in the same node share memory and cache; therefore, the interference from outside jobs is higher for $m = 2$ than for m being a multiple of four where each node is occupied only by our schedulability test during its execution.

VII. CONCLUSION

In this paper, we proposed an algorithm for the parallel schedulability analysis of real-time systems with multiple hardware and software modes. The proposed parallel schedulability test is designed such that the overhead associated with the parallel execution is minimized to obtain better speedup/efficiency. The experimental results substantiate the efficacy of the proposed algorithm for parallel schedulability analysis; therefore, the algorithm can be used as an effective tool for the exploration of design space while searching for optimal parameters of a multi-mode real-time system. Currently we are working on developing algorithms for allocating the minimum resource supply for each mode. Our ultimate goal is to restrict the runtime complexity of capacity determination to a pseudo-polynomial number of candidates. Obtaining the minimum capacity for a multi-modal system opens doors for further fruitful research in designing real-time control systems that minimize the peak-temperature or energy consumption.

ACKNOWLEDGMENTS

This research has been supported in part by an NSF-CAREER Grant (CNS-0953585), an NSF-CSR Grant (CNS-1116787), an NSF-IGERT Grant (DGE-0654014), and funding from Wayne State University's Office of the Vice President for Research.

REFERENCES

- [1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proc. of the 19th IEEE Real-Time Systems Symp.*, Spain, 1998.
- [2] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proc. of the 11th IEEE Real-Time Systems Symp.*, Florida, 1990.
- [3] S. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *Proc. of the 24th IEEE Real-Time Systems Symp.*, Mexico, 2003.
- [4] G. Buttazzo and L. Abeni. Adaptive rate control through elastic scheduling. In *Proc. of the 39th IEEE Conference on Decision and Control*, Sydney, 2000.
- [5] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using EDP resource models. In *Proc. of the 28th IEEE Real-time Systems Symp.*, Arizona, 2007.
- [6] N. Fisher and M. Ahmed. Tractable real-time schedulability analysis for mode changes under temporal isolation. In *Proc. of the 9th IEEE Symp. on Embedded Systems for Real-Time Multimedia*, Taipei, 2011.
- [7] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing, Second Edition*. Addison Wesley, USA, 2003.
- [8] P. M. Hettiarachchi, N. Fisher, M. Ahmed, L. Y. Wang, S. Wang, and W. Shi. The design and analysis of thermal-resilient hard-real-time systems. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symp.*, Beijing, 2012.
- [9] J. Kim, I. Jung, and B. Lee. A scheduling technique based on the variable bandwidth server for multimedia systems. In *Proc. of the 21st IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering*, Beijing, 2002.
- [10] A. K. Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, MIT, 1983. Technical Report.
- [11] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-time systems. In *Proc. of the 10th Euromicro Workshop on Real-Time Systems*, Berlin, 1998.
- [12] L. Phan, I. Lee, and O. Sokolsky. Compositional analysis of multi-mode systems. In *Proc. of the 22nd Euromicro Conference on Real-Time Systems*, Brussels, 2010.
- [13] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26(2):161–197, 2004.
- [14] L. Santinelli, G. Buttazzo, and E. Bini. Multi-moded resource reservations. In *Proc. of the 17th IEEE Real-Time and Embedded Technology and Applications Symp.*, Chicago, 2011.
- [15] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embed. Comput. Syst.*, 7(3):30–69, 2008.
- [16] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic task scheduling for hard real-time systems. *Real-Time Systems*, 1(1):27–69, 1989.
- [17] N. Stoimenov, S. Perathoner, and L. Thiele. Reliable mode changes in real-time systems with fixed priority or edf scheduling. In *Proc. of the Conference on Design, Automation and Test in Europe*, Nice, 2009.
- [18] N. Stoimenov, L. Thiele, L. Santinelli, and G. Buttazzo. Resource adaptations with servers for hard real-time systems. In *Proc. of the 10th ACM International Conference on Embedded Software*, USA, 2010.
- [19] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. of the IEEE International Symp. on Circuits and Systems*, Geneva, 2000.
- [20] K. Tindell, A. Burns, and A. Wellings. Mode changes in priority preemptively scheduled systems. In *Proc. of the 13th IEEE Real-Time Systems Symp.*, Arizona, 1992.
- [21] K. W. Tindell and A. Alonso. A very simple protocol for mode changes in priority preemptive systems. Technical report, Universidad Politécnica de Madrid, 1996.
- [22] M. Vachharajani. *Microarchitecture Modeling for Design-Space Exploration*. PhD thesis, Princeton University, USA, 2004.