Scheduling Constrained-Deadline Parallel Tasks on Two-type Heterogeneous Multiprocessors

Björn Andersson
Carnegie Mellon University

Gurulingesh Raravi
Xerox Research Centre India

Abstract—Consider the problem of scheduling a taskset on a multiprocessor so that all deadlines are met. Assume (i) constrained-deadline sporadic tasks, i.e., a task generates a sequence of jobs and the deadline of a job is no greater than the minimum inter-arrival time of the task that generates the job, (ii) stage-parallelism, i.e., a task comprises one or more stages with a stage comprising one or many segments so that segments in the same stage are allowed to execute in parallel and a segment is allowed to execute only if all segments of the previous stage have finished, (iii) two-type heterogeneous multiprocessor platform, i.e., there are processors of two types, type-1 and type-2, and for each task, there is a specification of its execution speed on a type-1 processor and on a type-2 processor, and (iv) intra-type migration, i.e., a job can migrate between processors of the same type but for a task, all jobs of this task must execute on the same processor type. We present an algorithm for this problem; it assigns each task to a processor type and then schedules tasks on processors of each type with global-Earliest-Deadline-First. This algorithm has pseudo-polynomial time complexity and speedup factors. A work by Holenderski et. al. [34] comes closest to ours as it also deals with the problem of scheduling parallel tasks on heterogeneous multiprocessors. However, the approach presented in [34] has no proven speedup factor. The algorithms for assigning implicit-deadline sporadic tasks (i.e., a task generates a sequence of jobs and a job has a deadline that is equal to the minimum inter-arrival time of the task that generates the job) to processors and to processor types for two-type platforms [1], [5]–[8] have lower time complexity than the algorithms for t-type platforms [2]–[4], [9]–[14] while maintaining their performance bound. In addition, an algorithm for scheduling arbitrary-deadline sporadic tasks (i.e., a task generates a sequence of jobs and a job has a deadline that may be less than or greater than or equal to the minimum inter-arrival time of the task that generates the job) on t-type platforms is known as well [15]. However, they do not support parallel tasks. The research community has also presented algorithms with proven speedup factors for scheduling parallel tasks on homogeneous multiprocessors [16]–[22]. Further, there are other algorithms [23]–[33] with no proven speedup factors for scheduling parallel tasks on heterogeneous multiprocessors — some of them [23]–[28] are for constrained-deadline tasks (i.e., a task generates a sequence of jobs and a job has a deadline that may be less than or equal to the minimum inter-arrival time of the task that generates the job) and the others [29]–[33] are for implicit-deadline tasks. Unfortunately, none of these works support heterogeneous multiprocessors (and moreover most of these algorithms [23]–[33] have no proven speedup factors). A work by Holenderski et. al. [34] comes closest to ours as it also deals with the problem of scheduling parallel tasks on heterogeneous multiprocessors. However, the approach presented in [34] has no proven speedup factor.

Related work. The algorithms for assigning implicit-deadline sporadic tasks (i.e., a task generates a sequence of jobs and a job has a deadline that is equal to the minimum inter-arrival time of the task that generates the job) to processors and to processor types for two-type platforms [1], [5]–[8] have lower time complexity than the algorithms for t-type platforms [2]–[4], [9]–[14] while maintaining their performance bound. In addition, an algorithm for scheduling arbitrary-deadline sporadic tasks (i.e., a task generates a sequence of jobs and a job has a deadline that may be less than or greater than or equal to the minimum inter-arrival time of the task that generates the job) on t-type platforms is known as well [15]. However, they do not support parallel tasks. The research community has also presented algorithms with proven speedup factors for scheduling parallel tasks on homogeneous multiprocessors [16]–[22]. Further, there are other algorithms [23]–[33] with no proven speedup factors for scheduling parallel tasks on heterogeneous multiprocessors — some of them [23]–[28] are for constrained-deadline tasks (i.e., a task generates a sequence of jobs and a job has a deadline that may be less than or equal to the minimum inter-arrival time of the task that generates the job) and the others [29]–[33] are for implicit-deadline tasks. Unfortunately, none of these works support heterogeneous multiprocessors (and moreover most of these algorithms [23]–[33] have no proven speedup factors). A work by Holenderski et. al. [34] comes closest to ours as it also deals with the problem of scheduling parallel tasks on heterogeneous multiprocessors. However, the approach presented in [34] has no proven speedup factor.

This research. In this paper, we present a pseudo-polynomial algorithm for scheduling constrained-deadline parallel tasks on a two-type heterogeneous multiprocessor and prove its speedup factor. Our approach assigns each task to a processor type and then uses global-Earliest-Deadline-First (gEDF) on the processors of each type to schedule the
### Title and Subtitle
Scheduling Constrained-Deadline Parallel Tasks on Two-type Heterogeneous Multiprocessors

### Author(s)
Raravi / Bjorn A. Andersson Gurulingesh

### Performing Organization Name(s) and Address(es)
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213

### Distribution/Availability Statement
Approved for public release, distribution unlimited.

### Supplementary Notes
The original document contains color images.

### Security Classification of:
- Report: unclassified
- Abstract: unclassified
- This Page: unclassified

### Limitation of Abstract
SAR

### Number of Pages
13

---

Form Approved
OMB No. 0704-0188
respective tasks. We show that our new algorithm has pseudo-polynomial time complexity and speedup factor at most 5. We study the constrained-deadline sporadic task model and we consider parallelism with stages (i.e., a task is described with one or more stages with each stage comprising one or many segments such that segments in the same stage are allowed to execute in parallel but a segment is only allowed to execute if all segments of the previous stage have finished execution). This work makes the following contribution: it presents the first algorithm for scheduling parallel real-time tasks on a heterogeneous multiprocessor with a proven speedup factor.

**Organization of the paper.** The rest of this paper is organized as follows. Section II states the system model. Section III lists previous results on parallel scheduling on homogeneous multiprocessors and also proves new lemmas that we use later in the paper. Section IV presents our new algorithm for two-type heterogeneous multiprocessors and proves its speedup factor and time complexity. Section V concludes.

**II. System Model**

We consider the problem of scheduling a set $\tau$ of constrained-deadline sporadic tasks on a two-type heterogeneous multiprocessor platform $\Pi$ comprising $m_1$ processors of type-1 and $m_2$ processors of type-2. A task $\tau_i \in \tau$ is characterized by a minimum inter-arrival time $T_i$ and a deadline $D_i$ such that $D_i \leq T_i$. Each task $\tau_i$ generates a sequence of jobs, with the first job arriving at any time and subsequent jobs arriving at least $T_i$ time units apart.

The execution of a task $\tau_i$ is described by $n_{s_i}$, $n_{seg_{i,j}}$, and $C_{i,j}$ with the interpretation that a job of $\tau_i$ has $n_{s_i}$ stages with stage $j$ comprising $n_{seg_{i,j}}$ segments with each segment of stage $j$ having execution requirement at most $C_{i,j}$ — see Fig. 1. A segment finishes when it performs a number of units of execution equal to its execution requirement. A segment executing contiguously for $L$ time units on a processor of speed $s$ performs $L \times s$ units of execution. A segment of a job is allowed to execute only if all segments of its previous stage have finished. A job finishes when all segments of its last stage have finished. If a job of $\tau_i$ finishes at most $D_i$ time units after its arrival, then it meets its deadline.

On a two-type platform, the execution speed of a job depends on the type of processor on which it executes. Let $r_{i,t}^1$ and $r_{i,t}^2$ denote the execution speeds of a job of task $\tau_i$ when it executes on a processor of type-1 and type-2 respectively.

We now define terms that we use in the rest of the paper.

**Definition 1 (Legal jobset).** If, for each task in the taskset $\tau$, the task is assigned the number of jobs it generates and each job is assigned an arrival time such that the minimum inter-arrival time constraint is satisfied and each segment of a job is assigned an execution requirement such that the upper bound on execution requirement of a segment is respected, then we say that the resulting jobset is a legal jobset with respect to $\tau$.

**Definition 2 (Intra-migrative schedule).** A schedule is intra-migrative if both of the following conditions are true: (i) jobs are allowed to migrate between processors of the same type and (ii) for each task, it holds that, if a job executes on a processor of one type then all other jobs of this task execute on processors of the same type.

**Definition 3 (Intra-migrative feasible taskset).** A taskset $\tau$ is intra-migrative feasible on a two-type platform $\Pi$ if for each jobset that is legal with respect to $\tau$ there exists an intra-migrative schedule in which all deadlines are met.

**Definition 4 (S-Schedulable task set).** A taskset $\tau$ is $S$-schedulable on a two-type platform $\Pi$ if for each jobset that is legal with respect to $\tau$, for each schedule that $S$ can generate from the jobset, it holds that the schedule is intra-migrative and all deadlines are met.

**Definition 5 (Speed of the computing platform).** If $\Pi$ is a two-type platform then let $\Pi \times x$ denote a two-type platform where the speed of each processor is multiplied by $x$.

**Definition 6 (Speedup factor).** A scheduler $S$ has a speedup factor $SF_S$ if, for each taskset $\tau$, for each two-type platform $\Pi$, it holds that: if $\tau$ is intra-migrative feasible on $\Pi$ then $\tau$ is $S$-schedulable on $\Pi \times SF_S$.

In order to simplify our discussion in the rest of the paper, we rewrite our model to an equivalent formulation as follows. Instead of using $C_{i,j}$, $r_{i,t}^1$, and $r_{i,t}^2$, we use parameters $C_{i,j}$, $r_{i,j}^1$, and $s$ selected as follows: $C_{i,j} = C_{i,j}^1$ and $C_{i,j}^2/s = C_{i,j}^1/r_{i,t}^1$. We let $s.t.$ mean such that and $: = $ mean it holds that. We let $\{x|f(x)\}$ denote a set of elements so that an element $x$ is in the set if and only if $f(x)$ is true. For convenience, we write the predicate $(\forall t > 0 : x)$ to mean the predicate $(\forall t \text{ s.t. } t > 0 : x)$. For convenience, we also define $\text{DMAX} = \max_{\tau_i \in \tau} D_i$, $\text{DMIN} = \min_{\tau_i \in \tau} D_i$, and $\text{TMAX} = \max_{\tau_i \in \tau} T_i$.

**III. Schedulability Analysis of Parallel Tasks on a Homogeneous Multiprocessor**

It is known that there is no optimal online algorithm for scheduling constrained-deadline sporadic tasks on a homogeneous multiprocessor (even for tasks without parallelism) [35]. Therefore, in this paper, we use global-Earliest-Delay-First (gEDF) scheduling policy as it has a good speedup factor [36]. There is a brute-force approach [37] which provides exact...
scheduling test for gEDF but it has a very large time-complexity and it does not support parallel tasks and it requires that tasks parameters are integers. Therefore, in this paper, we use a sufficient (not exact) schedulability test for parallel tasks.

The research literature provides many sufficient schedulability tests for gEDF for tasks that are not parallel — see for example [38]–[40]. Of particular interest is [40] which shows this schedulability test (see Eq. (6)) for parallel tasks on a homogeneous multiprocessor. Since this formulation is forced-forward demand-bound function; it is a function which describes the maximum amount of execution a given task can demand in a time interval of duration $t$. Later work [18] extended this for parallel tasks and this was done by defining ffdbf for parallel tasks. Fig. 2 shows this schedulability test (see Eq. (6)) for parallel tasks on a homogeneous multiprocessor [18] comprising $m$ processors. It also shows a feasibility test (see Eq. (7)) for parallel tasks on a homogeneous multiprocessor. Since this formulation is for homogeneous multiprocessors, we do not have the 1 and 2 on $C_{i,j}$. Some basic properties of ffdbf are shown below.

Lemma 1. $\forall t_0 > 0, \forall t > t_0 : \text{ffdbf} (\tau_i, t_0, v, s) \leq \text{ffdbf} (\tau_i, t, v, s)$

Proof: Follows from inspection of terms in Eqs. (1)-(4). ■

Lemma 2. $\forall l \in \mathbb{Z} : \text{ffdbf} (\tau_i, t + l \times T_i, v, s) = \text{ffdbf} (\tau_i, t, v, s) + l \times C_i$
Lemma 4. In Fig. 3 shows the definition of approximate C geneorous multiprocessor. We will use notations in Fig. 2 and Lemma 3. From Eq. (1) we obtain: C = τ i (v, s) = τ i (v, s). This states the lemma.

Proof: Follows from Eq. (4).

Lemma 5. ffdbf (τ i, mi(τ), v, s) + C i ≤ ffdbf (τ i, 2 × mi(τ), v, s)

Proof: Applying Lemma 2 with l = 1, yields ffdbf (τ i, mi(τ), v, s)+C i = ffdbf (τ i, mi(τ) + T i, v, s).

Applying this on the above and using Lemma 1 yields ffdbf (τ i, mi(τ), v, s) + C i = ffdbf (τ i, 2 × mi(τ), v, s). This states the lemma.

We will now introduce a function ffdbf* (used to over-approximate ffdbf) such that for inputs where t is less than or equal to mi(τ), it holds that ffdbf* (τ i, t, v, s, τ) is a staircase function and for t greater than mi(τ), it holds that ffdbf* (τ i, t, v, s, τ) increases linearly with t. Formally, Eq. (9) in Fig. 3 shows the definition of ffdbf* (τ i, t, v, s, τ).

Lemma 6. ∀t > 0: ffdbf (τ i, t, v, s) ≤ ffdbf* (τ i, t, v, s, τ)

Proof: See Appendix.

Definition 7. TS(τ, θ) def = t (2^n log_2 t) = t) & (DMIN × (1 − θ) ≤ t ≤ mi(τ)) ∪ (2^n log_2 (DMIN × (1−θ)))

Lemma 5. ∀t ∈ TS(τ, θ) : ffdbf* (τ i, t, v, s, τ) = ffdbf (τ i, 2 × t, v, s)

Proof: Follows from definition of ffdbf* and Definition 7.

IV. NEW SCHEDULING ALGORITHM AND SPEEDUP FACTOR

In this section, we discuss scheduling on a two-type heterogeno- nous multiprocessor. We will use notations in Fig. 2 and Fig. 3 but with 1 as superscript; this superscript indicates that the quantity if based on C^1_{i,j}. Ditto for type-2. For example, from Eq. (1) we obtain: C_i^1 def = ∑_{j=1}^{n_{seg}} (n_{seg} × C_{i,j}^1) and C_i^2 def = ∑_{j=1}^{n_{seg}} (n_{seg} × C_{i,j}^2).

A. Developing the new algorithm

The problem of scheduling constrained-deadline parallel sporadic tasks on a two-type heterogeneous multiprocessor can be solved in two steps. Step 1: Before run-time, assign tasks to processor types so that (i) tasks assigned to type-1 are gEDF-schedulable on the processors of type-1 and (ii) tasks assigned to type-2 are gEDF-schedulable on the processors of type-2. Step 2: At run-time, schedule all tasks assigned to type-1 with gEDF on processors of type-1 and schedule all tasks assigned to type-2 with gEDF on processors of type-2.

Since Step 2 is straightforward, we focus our discussion on Step 1.

Step 1 could be solved as follows. Let x_i^1 = 1 indicate that task τ_i is assigned to type-1 processors and let x_i^2 = 1 indicate that task τ_i is assigned to type-2 processors. Let X denote the matrix of x_i values for all tasks in τ. Then, by using Eq. (6), one could solve Step 1 by assigning values to x_i variables such that all the constraints in Fig. 4 are satisfied. Intuitively, C1 in Fig. 4 states that according to the schedulability test of Eq. (6), the tasks assigned to type-1 processors are gEDF-schedulable on type-1 processors. C2 is analogous for type-2 processors. C3 combined with C4 states that a task is either assigned to type-1 or type-2. C4 states that x_i-variables are integers. Unfortunately, creating an algorithm that assigns values to x_i such that all the constraints in Fig. 4 are satisfied is challenging because (i) it involves an exists-quantifier (3x ∈ C1 and 3x ∈ C2) and (ii) it involves a forall-quantifier (∀x ∈ C1 and ∀x ∈ C2) and (iii) it has integer variables. Hence, we will now present other constraints so that if these other constraints are satisfied then the constraints in Fig. 4 are satisfied as well.

Let θ_1 and θ_2 be non-negative parameters that we can choose. Then, instead of asking if there exists a θ in C1 in Fig. 4 with certain properties, let us only consider those θ such that θ/s = θ_1. Then it follows that if there is a task τ_i with x_i^1 = 1 and θ_i^1 > θ_1 × s(π) then C1 is violated. Hence, if θ_1 is given and θ/s = θ_1 and if θ_i^1 > θ_1 × s(π) then it follows that a necessary condition to satisfy the constraints in Fig. 4 is that x_i^1 = 0. We can reason analogously for θ_2 and C2. For this reason, we introduce the following sets:

H12 def = {τ_i ∈ τ | (θ_i^1 > θ_1 × s(π)) \land \theta_i^2 > θ_2 × s(π))} (10)
H1 def = {τ_i ∈ τ | (θ_i^1 ≤ θ_1 × s(π)) \land \theta_i^2 > θ_2 × s(π))} (11)
H2 def = {τ_i ∈ τ | (θ_i^1 > θ_1 × s(π)) \land \theta_i^2 ≤ θ_2 × s(π))} (12)
L def = {τ_i ∈ τ | (θ_i^1 ≤ θ_1 × s(π)) \land \theta_i^2 ≤ θ_2 × s(π))} (13)

Observe that τ = H12 ∪ H1 ∪ H2 ∪ L. We let H12(θ_1, θ_2, τ) denote H12 for the parameters θ_1, θ_2, τ. Analogously for H1, H12, and L.

Clearly, if θ_1 and θ_2 are given and θ/s = θ_1 and θ/s = θ_2 and if there is a task in H12 then it is impossible to satisfy the constraints in Fig. 4. Also, if θ_1 is given and θ/s = θ_1 then a necessary condition to satisfy the constraints in Fig. 4 is to set, for each task τ_i in H1, x_i^1 = 1. Analogously, if θ_2 is given and θ/s = θ_2 then a necessary condition to satisfy the constraints in Fig. 4 is to set, for each task τ_i in H2, x_i^2 = 1. This gives us the constraints in Fig. 5. It can be seen that if θ_1 and θ_2 are given and for a matrix X, it holds that all
If it is C1 then we can reason as follows: There must be a

Proof: Suppose that the lemma was false. Then there exists \( \tau, \Pi, \theta_1, \theta_2, X \) such that \( X \) satisfies Fig. 6 and \( X \) does not satisfy Fig. 5. Note that it can only be C1 or C2 (or both) in Fig. 5 that are violated. Constraints in Fig. 5 are satisfied then all constraints in Fig. 4 are satisfied as well.

Note that there is still a \( \forall t \) in C1 and C2 in Fig. 5. We will now present a set of constraints where we only check a finite number of \( t \) — see Fig. 6.

Lemma 6. If \( X \) satisfies Fig. 6 then \( X \) satisfies Fig. 5.
Let us explore three possibilities: 

Case 1: $t > m_i(\tau)$. Note that $m_i(\tau)$ is an element in $TS(\tau, \theta_1)$. This gives us from Fig. 6:

$$
\left( \sum_{\tau_i \in \tau} \text{ffdbf}^1(\tau_i, t, \theta_1, s) \times x_i^1 \right) 
\leq 
(m_1 - (m_1 - 1) \times \theta_1) \times t \times s
$$

(14)

Let us define $C_8$ in Fig. 6, it also holds that:

$$
\left( \sum_{\tau_i \in \tau} \left( C_i/T_i \right) \times x_i^1 \right) 
\leq 
(m_1 - (m_1 - 1) \times \theta_1) \times s
$$

(15)

Multiplying Eq. (16) by $(t - m_1(\tau))$ and adding to Eq. (15) and then combining with Eq. (14) yields:

$$
\left( \sum_{\tau_i \in \tau} \text{ffdbf}^1(\tau_i, \text{mi}(\tau), \theta_1, s, \tau) \right) + C_i + \frac{C_i}{T_i} \times (t - m_1(\tau)) \times x_i^1 
\leq 
\left( \sum_{\tau_i \in \tau} \text{ffdbf}^1(\tau_i, t, \theta_1, s) \times x_i^1 \right)
$$

(16)

Since $TS(\tau, \theta_1)$, we can apply Lemma 5 on the left-most term. Doing so and then applying Lemma 3 yields:

$$
\left( \sum_{\tau_i \in \tau} \text{ffdbf}^1(\tau_i, \text{mi}(\tau), \theta_1, s) \right) + C_i + \frac{C_i}{T_i} \times (t - m_1(\tau)) \times x_i^1 
\leq 
\left( \sum_{\tau_i \in \tau} \text{ffdbf}^1(\tau_i, t, \theta_1, s) \times x_i^1 \right)
$$

(17)

Note that the left-hand side is the expression of $\text{ffdbf}^1$ (the second case of Eq. 9). Hence:

$$
\left( \sum_{\tau_i \in \tau} \text{ffdbf}^1(\tau_i, t, \theta_1, s, \tau) \times x_i^1 \right) 
\leq 
\left( \sum_{\tau_i \in \tau} \text{ffdbf}^1(\tau_i, t, \theta_1, s) \times x_i^1 \right)
$$

(18)

But this contradicts Lemma 4. 

Case 2: $t < \text{DMIN} \times (1 - \theta_1)$. For such a $t$, it holds that $\text{ffdbf}^1(\tau_i, t, \theta_1, s)$ is zero. But this violates Eq. 14. 

Case 3: $\text{DMIN} \times (1 - \theta_1) \leq t \leq m_1(\tau)$. Let us define $t_1$ as $t_1 = 2^{\lceil \log_2 t \rceil + 1}$ and let $0_0$ be $t_1/2$. It is easy to see that $t_0 \leq t < t_1$. Note that $t_0$ is an element in $TS(\tau, \theta_1)$. This gives us from Fig. 6:

$$
\left( \sum_{\tau_i \in \tau} \text{ffdbf}^1(\tau_i, t_0, \theta_1, s, \tau) \times x_i^1 \right) 
\leq 
(m_1 - (m_1 - 1) \times \theta_1) \times t_0 \times s
$$

(19)

Since $t_0 \leq t$ it clearly holds that:

$$(m_1 - (m_1 - 1) \times \theta_1) \times t_0 \times s \leq (m_1 - (m_1 - 1) \times \theta_1) \times t \times s
$$

(20)

Lemma 5 yields: $\text{ffdbf}^1(\tau_i, t_0, \theta_1, s, \tau) = \text{ffdbf}^1(\tau_i, t_1, \theta_1, s)$. Combining this with Eq. (14), Eq. (17), and Eq. (18) yields:

$$
\sum_{\tau_i \in \tau} \text{ffdbf}^1(\tau_i, t_1, \theta_1, s) \times x_i^1 
\leq 
\sum_{\tau_i \in \tau} \text{ffdbf}^1(\tau_i, t, \theta_1, s) \times x_i^1
$$

Using this and $t < t_1$ and Lemma 1 yields a contradiction. If $C_2$ is violated then we can reason analogously to the case when $C_1$ is violated.

It can be seen that if the lemma is false then each case results in contradiction. Hence, the lemma is true. 

Note that in Fig. 6, there is a finite number of constraints and this is what we want. However, the $X$ variables are integers and this makes the problem a Mixed-Integer Linear Program (MILP); the research literature currently neither offers a polynomial time algorithm for solving general MILP nor for solving MILP with the special structure of Fig. 6. For Linear Programming (LP), polynomial time algorithms are known though (see [41], for example). We will now discuss how to exploit this. Fig. 7 shows an LP; it differs from Fig. 6 in that $X$ variables are real numbers instead of integers and it is also more constrained — $s/2$ instead of $s$ in C1,C2,C8,C9. With the solution to this LP, we obtain a new optimization problem — see Fig. 8. This optimization problem is as follows. First, we solve the LP (specified by Fig. 7) and obtain a solution $X'$. With this solution $X'$, we consider the MILP (specified by Fig. 6) and require that for those $i$ such that $x_i^1 = 1$ or $x_i^2 = 1$, the value of $x_i^1$ should be equal to $x_i^1$ and the value of $x_i^2$ should be equal to $x_i^2$. There may be some $i$s that remain; these will be assigned values by solving a MILP (as specified by Fig. 8).

**Lemma 7. If $\theta_1$ and $\theta_2$ are given and $X$ satisfies Fig. 8 then $X$ satisfies Fig. 4.**

**Proof:** Follows from the discussion in this subsection. 

Hence, solving Fig. 8 yields an assignment of tasks to processor types.

**B. Stating the new algorithm**

We let solvePTMILP$(\tau, \Pi, \theta_1, \theta_2)$ denote a function which takes as input a taskset $\tau$ and a computer platform $\Pi$ and $\theta_1$ and $\theta_2$ and returns a tuple $(f, X)$ where $f$ is a boolean and $X$ is a matrix with the following interpration: if Fig. 8 is feasible then $f$ is true and $X$ is the solution; if Fig. 8 is infeasible then $f$ is false and $X$ is undefined.

Algorithm 1 lists the pseudo-code for evaluating the function solvePTMILP$(\tau, \Pi, \theta_1, \theta_2)$.

**Definition 8.**

$$
R(\Pi) = 4 + \max \left( 1 - \frac{1}{m_1}, 1 - \frac{1}{m_2} \right)
$$

Algorithm 2 shows how the assignment of tasks to processor types works.

**Theorem 1.** If $(f, X) = \text{solvePTMILP}(\tau, \Pi, \theta_1, \theta_2)$ and $f$ is true and tasks are assigned to processor types according
Algorithm 1: An algorithm for evaluating the function \(\text{solvePTMILP}(\tau, \Pi, \theta_1, \theta_2)\).

Input : A taskset \(\tau\) and a two-type platform \(\Pi\) and \(\theta_1, \theta_2\)
Output: An assignment of tasks to processor types indicated by matrix \(X\)
1 if \(H12 = \emptyset\) then
2 Solve the LP in Fig. 7 and obtain a vertex solution
3 if this LP is feasible then
4 Let \(\chi\) denote this solution.
5 Let \(F\) denote a set of indices of tasks in \(L\) such that \((x^i_1 \neq 1) \lor (x^i_2 \neq 1)\).
6 Let us introduce \(X_{\text{found}}\) which is an assignment of values to the \(x_i\)-variables whose subscript index is in \(F\); this assignment is initialized to be undefined.
7 Let us introduce a local variable \(\text{foundPTMILP}\) that is boolean and initialize it to false.
8 foreach assignment of 0-1 to the \(x_i\)-variables whose subscript index is in \(F\) do
9 Evaluate if Fig. 8 is satisfied for this assignment
10 if the above evaluation yields true then
11 Let \(X\) denote the assignment of 0-1 to the \(x_i\)-variables whose subscript index is in \(F\)
12 if \(\text{foundPTMILP} = \text{false}\) then
13 Set \(\text{foundPTMILP}\) to true
14 Set \(\text{found}\) to \(\chi\)
15 end
16 end
17 if \(\text{foundPTMILP}\) then
18 Form the matrix \(X\) as follows:
19 For each \(i \in F\): Assign \(x^i_1\) and \(x^i_2\) according to \(X_{\text{found}}\).
20 For each \(i \in L \setminus F\): Assign \(x^i_1\) and \(x^i_2\) according to \(X\).
21 For each \(i \in H1\): Assign \(x^i_1 = 1\) and \(x^i_2 = 0\).
22 For each \(i \in H2\): Assign \(x^i_1 = 0\) and \(x^i_2 = 1\).
23 return \((\text{true}, X)\)
24 else
25 return \((\text{false}, X')\)
26 end
27 else
28 return \((\text{false}, X)\), where \(X\) is undefined.
29 end
30 else
31 return \((\text{false}, X)\), where \(X\) is undefined.
32 end

Algorithm 2: The new intra-migrative task assignment algorithm for two-type heterogeneous multiprocessors.

Input : A taskset \(\tau\) and a two-type platform \(\Pi\)
Output: An assignment of tasks to processor types indicated by matrix \(X\)
1 \((f, X) := \text{solvePTMILP}(\tau, \Pi, 1/R(\Pi), 1/R(\Pi))\)
2 if \((f = \text{true})\) then
3 declare SUCCESS and stop
4 else
5 declare FAILURE and stop
6 end

all deadlines will be met at run-time.

Proof: Follows from Theorem 1.

C. Proving the time complexity of the new algorithm

Lemma 8. \(|\text{TS}(\tau, \theta_1)| = [\log_2 \frac{\text{mi}(\tau)}{\text{DMIN} \times (1-\theta_1)}] + 2\) and \(|\text{TS}(\tau, \theta_2)| = [\log_2 \frac{\text{mi}(\tau)}{\text{DMIN} \times (1-\theta_2)}] + 2\)

Proof: Follows from the definition of \(\text{TS}\) — see Definition 7.

Lemma 9. After line 5 of Algorithm 1 has executed, it holds that: \(|F| \leq [\log_2 \frac{\text{mi}(\tau)}{\text{DMIN} \times (1-\theta_1)}] + [\log_2 \frac{\text{mi}(\tau)}{\text{DMIN} \times (1-\theta_2)}] + 6\).

Proof: In the LP, solved on line 2 of Algorithm 1, there are \(2 \times L\) variables and there are \([\log_2 \frac{\text{mi}(\tau)}{\text{DMIN} \times (1-\theta_1)}] + [\log_2 \frac{\text{mi}(\tau)}{\text{DMIN} \times (1-\theta_2)}] + 6 + |L|\) constraints \((2 + 2 \times |F|) + 2 \times |F| \leq [\log_2 \frac{\text{mi}(\tau)}{\text{DMIN} \times (1-\theta_1)}] + [\log_2 \frac{\text{mi}(\tau)}{\text{DMIN} \times (1-\theta_2)}] + 6 + |L|\). Rewriting yields: \(|F| \leq [\log_2 \frac{\text{mi}(\tau)}{\text{DMIN} \times (1-\theta_1)}] + [\log_2 \frac{\text{mi}(\tau)}{\text{DMIN} \times (1-\theta_2)}] + 6\). This states the lemma.

Lemma 10. The number of iterations of the for-loop on line 8 of Algorithm 1 is at most: \(\left(\frac{\text{TMAX} + \text{DMAX}}{\text{DMIN}}\right)^2 \times \frac{2}{\text{DMIN}} \times \frac{2}{1-\theta_1} \times \frac{2}{1-\theta_2}\).

Proof: The number of iterations is at most \(2^{|F|}\). Using Lemma 9 yields that the number of iterations is at most:

\[2^{|F|}\leq [\log_2 \frac{\text{mi}(\tau)}{\text{DMIN} \times (1-\theta_1)}] + [\log_2 \frac{\text{mi}(\tau)}{\text{DMIN} \times (1-\theta_2)}] + 6\]

Observing that \(\text{mi}(\tau) \leq 2 \times (\text{TMAX} + \text{DMAX})\) and re-writing yields the lemma.

Lemma 11. The time complexity of Algorithm 1 is \(O\left(\text{poly}\left(\frac{\text{TMAX} + \text{DMAX}}{\text{DMIN}}\right)^2 \times \frac{1}{(1-\theta_1) \times (1-\theta_2)}\right)\).

Proof: Follows from the facts that (i) linear programs can be solved in polynomial time [41] and hence line 2 of Algorithm 1 can be performed in polynomial time and (ii) the for-number of combinations iterated through in the for-loop of line 8 is at most \(\left(\frac{\text{TMAX} + \text{DMAX}}{\text{DMIN}}\right)^2 \times \frac{2}{\text{DMIN}} \times \frac{2}{1-\theta_1} \times \frac{2}{1-\theta_2}\). (Follows from Lemma 10.)

Theorem 3. The time complexity of Algorithm 2 is \(O\left(\text{poly}\left(\frac{\text{TMAX} + \text{DMAX}}{\text{DMIN}}\right)^2\right)\).

Proof: Since Algorithm 2 calls solvePTMILP once with \(\theta_1 = \theta_2 = 1/R(\Pi)\) it follows that (using Lemma 11) the time complexity of Algorithm 2 is
\[ O\left(\text{poly} + \left(\frac{\TMAX + \DMAX}{\DMIN}\right)^2 \times \frac{1}{(1 - 1/R(\Pi)) \times (1 - 1/R(\Pi))}\right) . \] Observing that \( 4 \leq R(\Pi) \) yields that the time complexity of Algorithm 2 is \( O\left(\text{poly} + \left(\frac{\TMAX + \DMAX}{\DMIN}\right)^2\right) . \)

D. Proving the speedup factor of the new algorithm

We will start by discussing necessary conditions for intra-migrative feasibility and then to prove the speedup factor.

**Lemma 12.** Consider a taskset \( \tau \) and a computer platform \( \Pi \). If \( \tau \) is intra-migrative feasible on \( \Pi \) then there exists a matrix \( X \) such that all constraints in Fig. 9 are satisfied.

**Proof:** Follows from the fact that Eq. (7) is a necessary condition for feasibility.

**Lemma 13.** Consider a taskset \( \tau \) and a computer platform \( \Pi \). If \( \tau \) is intra-migrative feasible on \( \Pi \times 1/R(\Pi) \) then there exists a matrix \( X \) such that all constraints in Fig. 10 are satisfied.

**Proof:** Follows from applying Lemma 12 on \( \Pi \times (1/R(\Pi)) \) and then considering \( t \to \infty \) on C1 and C2 yields C8 and C9 respectively.

**Lemma 14.** \( \forall Q \geq 0 : \)
\[
(\forall t \geq 0 : \left(\sum_{\tau_i \in \tau} \text{ffdbf}^1(\tau_i, t, \theta_1, s)\right) \leq m \times t \times Q) \Rightarrow \]
\[
(\forall t \in TS(\tau, \theta_1) : \left(\sum_{\tau_i \in \tau} \text{ffdbf}^{\star 1}(\tau_i, t, \theta_1, s, \tau)\right) \leq m \times t \times Q \times 2) \]

**Proof:** See Appendix.

**Lemma 15.** \( \forall Q \geq 0 : \)
\[
(\forall t \geq 0 : \left(\sum_{\tau_i \in \tau} \text{ffdbf}^2(\tau_i, t, \theta_2, s)\right) \leq m \times t \times Q) \Rightarrow \]
\[
(\forall t \in TS(\tau, \theta_2) : \left(\sum_{\tau_i \in \tau} \text{ffdbf}^{\star 2}(\tau_i, t, \theta_2, s, \tau)\right) \leq m \times t \times Q \times 2) \]

**Proof:** See Appendix.

**Lemma 16.** Consider a taskset \( \tau \) and a computer platform \( \Pi \). If \( X \) satisfies Fig. 10 then \( X \) satisfies Fig. 11.

**Proof:** Follows from applying Lemma 14 on C1 in Fig. 10 and applying Lemma 15 on C2 in Fig. 10.

**Lemma 17.** Consider a taskset \( \tau \) and a computer platform \( \Pi \). If \( \tau \) is intra-migrative feasible on \( \Pi \times 1/R(\Pi) \) then there exists a matrix \( X \) such that all constraints in Fig. 11 are satisfied.

**Proof:** Follows Lemma 13 and Lemma 16.

**Lemma 18.** Consider a taskset \( \tau \) and a computer platform \( \Pi \). If \( \tau \) is intra-migrative feasible on \( \Pi \times 1/R(\Pi) \) then there exists a matrix \( X \) such that all constraints in Fig. 12 are satisfied.

**Proof:** Algebraic manipulations of \( R(\Pi) \) (from Definition 8) yields:
\[
m_1 \times t \times \frac{s}{R(\Pi)} \times 2 \leq (m_1 - \frac{1}{R(\Pi)}) \times t \times \frac{s}{2} \quad (19)
\]
\[
m_2 \times t \times \frac{s}{R(\Pi)} \times 2 \leq (m_2 - \frac{1}{R(\Pi)}) \times t \times \frac{s}{2} \quad (20)
\]
\[
m_1 \times \frac{s}{R(\Pi)} \leq (m_1 - \frac{1}{R(\Pi)}) \times \frac{s}{2} \quad (21)
\]
\[
m_2 \times \frac{s}{R(\Pi)} \leq (m_2 - \frac{1}{R(\Pi)}) \times \frac{s}{2} \quad (22)
\]
Hence, if \( X \) satisfies Fig. 11 then it also satisfies Fig. 12. Combining this with Lemma 17 yields the lemma.

**Lemma 19.** Consider a taskset \( \tau \) and a computer platform \( \Pi \). If there exists a matrix \( X \) such that all constraints in Fig. 12 are satisfied then Algorithm 2 declares SUCCESS.

**Proof:** Let us suppose that the lemma was false. Then there is a taskset \( \tau \) and a computer platform \( \Pi \) such that there exists a matrix \( X \) such that all constraints in Fig. 12 are satisfied (23) and Algorithm 2 declares FAILURE.

Relaxing C4 in Fig. 12 yields:

there exists a matrix \( X \) such that all constraints in Fig. 13 are satisfied (24)

Eq. (23) yields
\[
H12(1/R(\Pi), 1/R(\Pi), \tau) = \emptyset
\]
(25)
Consider Fig. 7 with \( \theta_1 = \theta_2 = 1/R(\Pi) \) and compare with Fig. 13. They are identical. Hence:

there is a matrix \( X \) such that all constraints in Fig. 7 are satisfied for \( \theta_1 = \theta_2 = 1/R(\Pi) \) (26)

From the statement that Algorithm 2 declares FAILURE it follows that Algorithm 1 fails for the input \( \tau, \Pi, 1/R(\Pi), 1/R(\Pi) \). Since it fails, let us explore the possible lines at which it can fail.

**Case 1. Algorithm declares FAILURE on line 32.**

The condition of the case yields \( H12(1/R(\Pi), 1/R(\Pi), \tau) \neq \emptyset \). But this contradicts Eq. (25).

**Case 2. Algorithm declares FAILURE on line 29.**

From the condition of the case, it follows that there exists no matrix \( X \) such that all constraints in Fig. 7 are satisfied for \( \theta_1 = \theta_2 = 1/R(\Pi) \)

But this contradicts Eq. (26).

**Case 3. Algorithm declares FAILURE on line 26.**

From the condition of the case, it follows that a solution PTMILP is false when the algorithm declares FAILURE on line 26. Let us partition \( \tau \) into \( F \) and \( \tau \setminus F \). Note that for \( \tau \setminus F \) it holds that \( X' \) satisfies Fig. 7 and since this set of tasks have \( x_i^1 \) and \( x_i^2 \)
being integers (follows from the fact that it does not contain the tasks in $F$), it follows that $X'$ also satisfies the following

Fig. 9: Constraints expressing a necessary intra-migrative feasibility condition.

C1. $\forall t \geq 0: \sum_{x \in \mathcal{R}} \mathbb{f}(x, t, 1) \times x_t \leq m_1 \times t \times s$
C2. $\forall t \geq 0: \sum_{x \in \mathcal{R}} \mathbb{f}(x, t, 1) \times x_t \leq m_2 \times t \times s$
C3. $\forall r \in \tau: x_t^1 + x_t^2 = 1$
C4. $\forall r \in \tau: x_t^1 \in \{0, 1\}$ and $x_t^2 \in \{0, 1\}$
C5. $\forall r \in \mathcal{R}(1, 1, \tau): x_t^1 = 1$
C6. $\forall r \in \mathcal{R}(1, 1, \tau): x_t^2 = 1$
C7. $H12(1, 1, \tau) = \emptyset$

Fig. 10: Constraints expressing a necessary intra-migrative feasibility condition; rewritten.

C1. $\forall t \in TS(1, 1, \tau, 1/R(\tau)) : \left( \sum_{x \in \mathcal{R}} \mathbb{f}(x, t, 1) \times x_t \right) \leq m_1 \times t \times (s/R(\tau)) \times 2$
C2. $\forall t \in TS(1, 1, \tau, 1/R(\tau)) : \left( \sum_{x \in \mathcal{R}} \mathbb{f}(x, t, 1) \times x_t \right) \leq m_2 \times t \times (s/R(\tau)) \times 2$
C3. $\forall r \in \tau: x_t^1 + x_t^2 = 1$
C4. $\forall r \in \tau: x_t^1 \in \{0, 1\}$ and $x_t^2 \in \{0, 1\}$
C5. $\forall r \in \mathcal{R}(1,R(\tau), 1/R(\tau), \tau): x_t^1 = 1$
C6. $\forall r \in \mathcal{R}(1,R(\tau), 1/R(\tau), \tau): x_t^2 = 1$
C7. $H12(1/R(\tau), 1/R(\tau), \tau) = \emptyset$
C8. $\sum_{x \in \mathcal{R}} \mathbb{f}(x, t, 1) \times x_t \leq m_1 \times (s/R(\tau))$
C9. $\sum_{x \in \mathcal{R}} \mathbb{f}(x, t, 1) \times x_t \leq m_2 \times (s/R(\tau))$

Fig. 11: Constraints expressing a necessary intra-migrative feasibility condition; rewritten further.

C1. $\forall t \in TS(1, 1, \tau, 1/R(\tau)) : \left( \sum_{x \in \mathcal{R}} \mathbb{f}(x, t, 1) \times x_t \right) \leq (m_1 - (m_1 - 1) \times 1/R(\tau)) \times t \times s \times 1/2$
C2. $\forall t \in TS(1, 1, \tau, 1/R(\tau)) : \left( \sum_{x \in \mathcal{R}} \mathbb{f}(x, t, 1) \times x_t \right) \leq (m_2 - (m_2 - 1) \times 1/R(\tau)) \times t \times s \times 1/2$
C3. $\forall r \in \tau: x_t^1 + x_t^2 = 1$
C4. $\forall r \in \tau: x_t^1 \in \{0, 1\}$ and $x_t^2 \in \{0, 1\}$
C5. $\forall r \in \mathcal{R}(1,R(\tau), 1/R(\tau), \tau): x_t^1 = 1$
C6. $\forall r \in \mathcal{R}(1,R(\tau), 1/R(\tau), \tau): x_t^2 = 1$
C7. $H12(1/R(\tau), 1/R(\tau), \tau) = \emptyset$
C8. $\sum_{x \in \mathcal{R}} \mathbb{f}(x, t, 1) \times x_t \leq ((m_1 - (m_1 - 1) \times 1/R(\tau)) \times s \times 1/2)$
C9. $\sum_{x \in \mathcal{R}} \mathbb{f}(x, t, 1) \times x_t \leq ((m_2 - (m_2 - 1) \times 1/R(\tau)) \times s \times 1/2)$

Fig. 12: Constraints expressing a necessary intra-migrative feasibility condition; rewritten even more.

C1. $\forall t \in TS(1, 1, \tau, 1/R(\tau)) : \left( \sum_{x \in \mathcal{R}} \mathbb{f}(x, t, 1) \times x_t \right) \leq (m_1 - (m_1 - 1) \times 1/R(\tau)) \times t \times s \times 1/2$
C2. $\forall t \in TS(1, 1, \tau, 1/R(\tau)) : \left( \sum_{x \in \mathcal{R}} \mathbb{f}(x, t, 1) \times x_t \right) \leq (m_2 - (m_2 - 1) \times 1/R(\tau)) \times t \times s \times 1/2$
C3. $\forall r \in \tau: x_t^1 + x_t^2 = 1$
C4. $\forall r \in \tau: x_t^1 \geq 0$ and $x_t^2 \geq 0$
C5. $\forall r \in \mathcal{R}(1,R(\tau), 1/R(\tau), \tau): x_t^1 = 1$
C6. $\forall r \in \mathcal{R}(1,R(\tau), 1/R(\tau), \tau): x_t^2 = 1$
C7. $H12(1/R(\tau), 1/R(\tau), \tau) = \emptyset$
C8. $\sum_{x \in \mathcal{R}} \mathbb{f}(x, t, 1) \times x_t \leq ((m_1 - (m_1 - 1) \times 1/R(\tau)) \times s \times 1/2)$
C9. $\sum_{x \in \mathcal{R}} \mathbb{f}(x, t, 1) \times x_t \leq ((m_2 - (m_2 - 1) \times 1/R(\tau)) \times s \times 1/2)$

Fig. 13: Constraints expressing a necessary intra-migrative feasibility condition; rewritten to LP.
This material has been approved for the Software Engineering Institute, a federally funded research C-0003 with Carnegie Mellon University for the operation of.

Deadline-First on each type of processors. Our new algorithm assigns tasks to processor types and then apply global-Earliest-cessors are of two types and we presented a new algorithm that speedup factor. We did so by focusing on constrained-deadline processors on heterogeneous shared-memory multiprocessors,” in ECRTS, 2011.


A. Proof of Lemma 4

In this section, we prove Lemma 4 and we do this incrementally, i.e., by proving some basic results and then merging them to obtain the desired result.

**Lemma 20.** \( \forall t > \text{mi}(\tau) : \text{ffdbf}^* (\tau_i, t, v, s) \leq \text{ffdbf} (\tau_i, \text{mi}(\tau), v, s) + (C_i + C_i \times (t - \text{mi}(\tau))) \)

*Proof:* Algebraic manipulations yield:

\[
t = \text{mi}(\tau) + (t - \text{mi}(\tau)) = \left\lfloor \frac{t - \text{mi}(\tau)}{T_i} \right\rfloor \times T_i + (t - \text{mi}(\tau)) \mod T_i \leq \text{mi}(\tau) + \left\lfloor \frac{t - \text{mi}(\tau)}{T_i} \right\rfloor \times T_i + 1
\]

Using this on Lemma 1 yields:

\[
\text{ffdbf} (\tau_i, t, v, s) \leq \text{ffdbf} (\tau_i, \text{mi}(\tau), v, s) + 2 \times \left( \frac{t - \text{mi}(\tau)}{T_i} \right) + C_i
\]

Relaxing the bound on the right-hand side and rewriting yields:

\[
\text{ffdbf} (\tau_i, t, v, s) \leq \text{ffdbf} (\tau_i, \text{mi}(\tau), v, s) + (t - \text{mi}(\tau)) \times C_i + C_i
\]

This states the lemma.

**Lemma 21.** \( \forall t > 0, t \leq \text{mi}(\tau) : \text{ffdbf} (\tau_i, t, v, s) \leq \text{ffdbf} (\tau_i, 2^{\lfloor \log_2 t \rfloor + 1}, v, s) \)

*Proof:* Follows from Lemma 1 (monotonicity) and observing that \( t \leq 2^{\lfloor \log_2 t \rfloor + 1} \).

We now restate Lemma 4 and prove it.

**Lemma 4.** \( \text{ffdbf} (\tau_i, t, v, s) \leq \text{ffdbf}^* (\tau_i, t, v, s, \tau) \)

*Proof:* We need to consider two cases.

Case 1. \( t > \text{mi}(\tau) \): For this case Lemma 20 along with the definition of \( \text{ffdbf}^* \) in Eq. (9) proves the lemma.

Case 2. \( t \leq \text{mi}(\tau) \): For this case Lemma 21 along with the definition of \( \text{ffdbf}^* \) in Eq. (9) proves the lemma.
Lemma 14. This states the lemma.

Proof: Suppose that the lemma was false. Then it holds that there exists a $Q$, $v$ and $s$ such that $Q \geq 0$, $v \geq 0$, $s \geq 0$ and

$$\forall t \geq 0: \left( \sum_{\tau \in \tau} \text{ffdbf} (\tau_i, t, v, s) \right) \leq m \times t \times Q$$

Applying this on Eq. (28) yields:

$$\forall t \in (\text{mi}(\tau), \text{mi}(\tau) + T_i] : \text{ffdbf} (\tau_i, t, v, s) + C_i + \frac{C_i}{T_i} \times T_i \leq \text{ffdbf} (\tau_i, t, v, s) + 2 \times C_i \tag{28}$$

From $t \in (\text{mi}(\tau), \text{mi}(\tau) + T_i]$, we obtain that: $t - \text{mi}(\tau) \leq T_i$. Applying this on Eq. (28) yields:

$$\text{ffdbf} (\tau_i, t, v, s) + C_i + \frac{C_i}{T_i} \times (t - \text{mi}(\tau)) \leq \text{ffdbf} (\tau_i, t, v, s) + 2 \times C_i \tag{29}$$

Using the definition of $\text{ffdbf}^*$ yields:

$$\forall t \in (\text{mi}(\tau), \text{mi}(\tau) + T_i] : \text{ffdbf}^* (\tau_i, t, v, s, \tau) \leq \text{ffdbf} (\tau_i, t, v, s) + 2 \times C_i$$

Hence the proof.

Lemma 23. $\forall t > \text{mi}(\tau) : \text{ffdbf}^* (\tau_i, t, v, s, \tau) \leq \text{ffdbf} (\tau_i, t, v, s) + 2 \times C_i$

Proof: We prove this by contradiction. If the lemma was false then there exist a $t$ such that $t > \text{mi}(\tau)$ and

$$\text{ffdbf}^* (\tau_i, t, v, s, \tau) > \text{ffdbf} (\tau_i, t, v, s) + 2 \times C_i$$

If $t > \text{mi}(\tau) + T_i$, then decreasing $t$ by $T_i$ decreases the left-hand side and the right-hand side of the above inequality by the same amount ($C_i$). Hence, we can decrease $t$ by $T_i$ until it holds that $t \in (\text{mi}(\tau), \text{mi}(\tau) + T_i]$. And this gives us that there exists a $t$ such that $t \in (\text{mi}(\tau), \text{mi}(\tau) + T_i]$ and

$$\text{ffdbf}^* (\tau_i, t, v, s, \tau) > \text{ffdbf} (\tau_i, t, v, s) + 2 \times C_i$$

But this contradicts Lemma 22 and hence it is not possible that lemma under discussion is false. Hence the proof.

Lemma 24. $\forall t > \text{mi}(\tau) : \text{ffdbf}^* (\tau_i, t, v, s, \tau) \leq \text{ffdbf} (\tau_i, t, v, s) + 2 \times C_i$

Proof: Since $t > \text{mi}(\tau)$, it follows that $t > \text{TMAX} + \text{DMAX}$ and then it follows that:

$$\text{ffdbf} (\tau_i, t, v, s) \geq 2 \times C_i \tag{30}$$

Using Lemma 23 and Eq. (30) yields:

$$\forall t > \text{mi}(\tau) : \frac{\text{ffdbf}^* (\tau_i, t, v, s, \tau)}{\text{ffdbf} (\tau_i, t, v, s)} \leq \frac{\text{ffdbf} (\tau_i, t, v, s) + 2 \times C_i}{\text{ffdbf} (\tau_i, t, v, s)} \leq 1 + \frac{2 \times C_i}{\text{ffdbf} (\tau_i, t, v, s)} \leq 1 + \frac{2 \times C_i}{2 \times C_i} \leq 2 \tag{31}$$

Rewriting yields: $\forall t > \text{mi}(\tau) : \text{ffdbf}^* (\tau_i, t, v, s, \tau) \leq \text{ffdbf} (\tau_i, t, v, s) \times 2$

This states the lemma.
this on the last constraint yields:

\[
\left( \sum_{\tau_i \in \tau} \text{ffdbf} (\tau_i, 2 \times t_0, v, s) \right) \leq m \times 2 \times t_0 \times Q \\
\wedge \\
\left( \sum_{\tau_i \in \tau} \text{ffdbf} (\tau_i, 2 \times t_0, v, s) \right) > m \times t_0 \times Q \times 2
\]

This is a contradiction. End of Case 2.

It can be seen that if the lemma is false then for each case, we obtain a contradiction. Hence, the lemma is true.

**Lemma 15.** \( \forall Q \geq 0, \forall s \geq 0 : \)

\( \forall t \geq 0 : \left( \sum_{\tau_i \in \tau} \text{ffdbf}^2 (\tau_i, t, \theta_2, s) \right) \leq m \times t \times Q \)

\( \Rightarrow \)

\( \forall t \in TS(\tau, \theta_2) : \left( \sum_{\tau_i \in \tau} \text{ffdbf}^*^2 (\tau_i, t, \theta_2, s) \right) \leq m \times t \times Q \times 2 \)

**Proof:** Analogous to the proof of Lemma 14.