Report Number: WUCSE-2012-14

2012

Real-Time Scheduling of Parallel Tasks under a General DAG Model

Abusayeed Saifullah, David Ferry, Chenyang Lu, and Christopher Gill

Due to their potential to deliver increased performance over single-core processors, multi-core processors have become mainstream in processor design. Computation-intensive real-time systems must exploit intra-task parallelism to take full advantage of multi-core processing. However, existing results in real-time scheduling of parallel tasks focus on restrictive task models such as the synchronous model where a task is a sequence of alternating parallel and sequential segments, and parallel segments have threads of execution that are of equal length. In this paper, we address a general model for deterministic parallel tasks, where a task is represented as a DAG with different nodes... **Read complete abstract on page 2.**

Real-Time Scheduling of Parallel Tasks under a General DAG Model

Abusayeed Saifullah, David Ferry, Chenyang Lu, and Christopher Gill

Complete Abstract:

Due to their potential to deliver increased performance over single-core processors, multi-core processors have become mainstream in processor design. Computation-intensive real-time systems must exploit intra-task parallelism to take full advantage of multi-core processing. However, existing results in real-time scheduling of parallel tasks focus on restrictive task models such as the synchronous model where a task is a sequence of alternating parallel and sequential segments, and parallel segments have threads of execution that are of equal length. In this paper, we address a general model for deterministic parallel tasks, where a task is represented as a DAG with different nodes having different execution requirements. We make several key contributions towards both preemptive and non-preemptive realtime scheduling of DAG tasks on multi-core processors. First, we propose a task decomposition that splits a DAG into sequential tasks. Second, we prove that parallel tasks, upon decomposition, can be scheduled using preemptive global EDF with a resource augmentation bound of 4. This bound is as good as the best known bound for more restrictive models, and is the first for a general DAG model. Third, we prove that the decomposition has a resource augmentation bound of 4 plus a non-preemption overhead for non-preemptive global EDF scheduling. To our knowledge, this is the first resource augmentation bound for nonpreemptive scheduling of parallel tasks. Through simulations, we demonstrate that the achieved bounds are safe and sufficient.

2012-14

Real-Time Scheduling of Parallel Tasks under a General DAG Model

Authors: Abusayeed Saifullah, David Ferry, Kunal Agrawal, Chenyang Lu, and Christopher Gill

Corresponding Author: saifullah@wustl.edu

Web Page: http://www.cse.wustl.edu/~saifullaha/

Abstract: Due to their potential to deliver increased performance over single-core processors, multi-core processors have become mainstream in processor design. Computation-intensive real-time systems must exploit intra-task parallelism to take full advantage of multi-core processing. However, existing results in real-time scheduling of parallel tasks focus on restrictive task models such as the synchronous model where a task is a sequence of alternating parallel and sequential segments, and parallel segments have threads of execution that are of equal length. In this paper, we address a general model for deterministic parallel tasks, where a task is represented as a DAG with different nodes having different execution requirements. We make several key contributions towards both preemptive and non-preemptive realtime scheduling of DAG tasks on multi-core processors. First, we propose a task decomposition that splits a DAG into sequential tasks. Second, we prove that parallel tasks, upon decomposition, can be scheduled using preemptive global EDF with a resource augmentation bound of 4. This bound is as good as the best known bound for more restrictive models, and is the first for a general DAG model. Third, we prove that the decomposition has a resource augmentation bound of 4 plus a non-preemption overhead for non-preemptive global EDF scheduling. To our knowledge, this is the first resource augmentation bound for nonpreemptive scheduling of parallel tasks. Through simulations,

Type of Report: Other

Real-Time Scheduling of Parallel Tasks under a General DAG Model

Abusayeed Saifullah, David Ferry, Kunal Agrawal, Chenyang Lu, and Christopher Gill
Department of Computer Science and Engineering
Washington University in St. Louis

***Abstract*—Due to their potential to deliver increased performance over single-core processors, multi-core processors have become mainstream in processor design. Computation-intensive real-time systems must exploit intra-task parallelism to take full advantage of multi-core processing. However, existing results in real-time scheduling of parallel tasks focus on restrictive task models such as the synchronous model where a task is a sequence of alternating parallel and sequential segments, and parallel segments have threads of execution that are of equal length. In this paper, we address a general model for deterministic parallel tasks, where a task is represented as a DAG with different nodes having different execution requirements. We make several key contributions towards both preemptive and non-preemptive real-time scheduling of DAG tasks on multi-core processors. First, we propose a task decomposition that splits a DAG into sequential tasks. Second, we prove that parallel tasks, upon decomposition, can be scheduled using preemptive global EDF with a resource augmentation bound of 4. This bound is as good as the best known bound for more restrictive models, and is the first for a general DAG model. Third, we prove that the decomposition has a resource augmentation bound of 4 plus a non-preemption overhead for non-preemptive global EDF scheduling. To our knowledge, this is the first resource augmentation bound for non-preemptive scheduling of parallel tasks. Through simulations, we demonstrate that the achieved bounds are safe and sufficient.**

I. Introduction

Due to slowing down of the rate of increase of clock frequencies, most processor chip manufacturers have recently moved to increasing performance of processors by increasing the number of cores on each chip. Intel's 80-core Teraflops Research Chip [1], Tilera's 100-core TILE-Gx processor, AMD's 12-core Opteron processor [2], and a 96-core processor developed by ClearSpeed [3] are some notable examples of multi-core chips. With the rapid evolution of multi-core processor technology, however, real-time system software and programming models have failed to keep pace. In particular, most classic results in real time scheduling concentrate on sequential tasks running on multiple processors or cores [4]. While these systems allow many tasks to execute on the same multi-core host, they do not allow an individual task to run any faster on a multi-core machine than on a single-core one.

If we want to scale the capabilities of individual tasks with the number of cores, it is essential to develop new approaches for tasks with intra-task parallelism, where real-time tasks themselves are parallel tasks which can utilize multiple cores at the same time. Such intra-task parallelism may enable more stringent timing guarantees for complex real-time systems that require heavy computation such as video surveillance, computer vision, radar tracking, and hybrid real-time structural testing [5] whose stringent timing constraints are difficult to meet on traditional single-core processors.

There has been some recent work on real-time scheduling for parallel tasks, but it has been mostly restricted to the *synchronous task model* [6], [7]. In the synchronous model, each task consists of a sequence of segments with synchronization points at the end of each segment. In addition, each segment of a task contains threads of execution that are of *equal* length. For such synchronous tasks, our previous result [6] proves a resource augmentation bound of 4.

While the synchronous task model represents the kind of tasks generated by the *parallel for* loop construct that is common to many parallel languages such as OpenMP [8] and CilkPlus [9], most parallel languages also have other constructs for generating parallel programs, notably *fork-join* constructs. A program that uses fork-join constructs will generate a *non-synchronous* task, generally represented as a *Directed Acyclic Graph (DAG)*, where each thread (sequence of instructions) is a node and edges represent dependencies between threads. Our previous work [6] considers a restricted version of the DAG model, where each node (thread) requires unit computation. For the unit-node DAG model, the scheduler first converts each task to a synchronous task, and then applies the analysis followed for a synchronous model.

All previous work on parallel real-time tasks considers *preemptive scheduling*, where threads are allowed to preempt each other in the middle of execution. While this is a reasonable model, preemption can often be a high-overhead operation since it often involves a system call and a context switch. An alternative scheduling model is to consider *node-level non-preemptive scheduling* (simply called non-preemptive scheduling in this paper), where once the execution of a particular node (thread) starts, the thread cannot be preempted by any other thread. Most parallel languages and libraries have yield points at the end of threads (nodes of the DAG), allowing low-cost, user-space preemption at these yield points. For these languages and libraries, schedulers that require preemption only when threads end (in other words, where threads do not preempt each other) can be implemented entirely in user-space (without interaction with the kernel), and therefore have low overheads. In addition, this model also has cache benefits.

In this paper, we generalize the previous work in two ways. First, we consider a general task model, where tasks are represented by general DAGs where threads (nodes) can

have *arbitrary* execution requirements. Second, we address both *preemptive* and node-level *non-preemptive* scheduling for these DAGs. Note that if the decomposition proposed in [6] for unit-node DAG is applied to a general DAG, every thread (node) will further spilt into smaller threads. Since all subtasks of a segment synchronize at its end, there is no easy way of assuring non-preemption of a thread. In particular, this paper makes the following contributions.

- We propose a novel task decomposition to transform the nodes of a general DAG into sequential tasks. This decomposition does not convert non-synchronous tasks to synchronous tasks and therefore, unlike that in [6], it does not require splitting threads into shorter threads. Hence, our proposed decomposition allows non-preemptive scheduling, where threads (nodes of the DAG) are never preempted.
- We prove that parallel tasks in the general DAG model, upon decomposition, can be scheduled using preemptive global EDF with a resource augmentation bound of 4. This bound is as good as the best known bound for more restrictive models [6] and, to our knowledge, is the *first* for a general DAG model.
- We prove that the proposed decomposition requires a resource augmentation bound of 4 plus a non-preemption overhead of the tasks when using non-preemptive global EDF scheduling. To our knowledge, this is the *first* bound for *non-preemptive* scheduling of parallel real-time tasks.
- Our preliminary, short-scale simulations indicate that the bounds are safe. For most task sets, the resource augmentation required is at most 2 for preemptive scheduling and 3 for non-preemptive scheduling, which is significantly smaller than theoretical bound.

The rest of the paper is organized as follows. Section II reviews related work. Section III describes the task model. Section IV presents the new task decomposition. Sections V and VI present analyses for preemptive and non-preemptive global EDF scheduling, respectively. Section VII presents the simulation results. Section VIII offers conclusions.

II. Related Work

There has been a substantial amount of work on traditional multiprocessor real-time scheduling focused on sequential tasks [4]. Some work has addressed scheduling for parallel tasks [10]–[16], but it does not consider task deadlines. *Soft real-time scheduling* (where the goal is to meet a certain subset of deadlines based on application-specific criteria) has been studied for various parallel task models and for various optimization criteria [17]–[22]. For example, many investigations [17]–[20] focus on cache performance for multithreaded tasks, where the number of parallel threads in a task cannot exceed the number of cores. Others consider task models where a task is executed on up to a given number of processors, and focus on metrics such as the makespan [21] and total work done by tasks that meet their deadlines [22].

Hard real-time scheduling (where the goal is to meet all task deadlines) is intractable for most cases of parallel tasks without resource augmentation [23]. Some early work makes simplifying assumptions about task models [24]–[28]. For example, [24]–[26] address the scheduling of *malleable tasks*, where tasks can execute on varying number of processors without loss in efficiency. The study in [27] considers non-preemptive EDF scheduling of *moldable tasks*, where the actual number of processors used by a particular task is determined before starting the system and remains unchanged. Gang EDF scheduling [28] of moldable parallel tasks requires users to select (at submission time) a fixed number of processors upon which their task will run, and the task must then always use that number of threads.

Recently, *preemptive* real-time scheduling has been studied in [6], [7] for *synchronous* parallel tasks with implicit deadlines. In [7], every task is an alternate sequence of parallel and sequential *segments* with each parallel segment consisting of multiple threads of *equal* length that synchronize at the end of the segment. All parallel segments in a task have an *equal* number of threads which cannot *exceed* the number of processor cores. It transforms every thread to a subtask, and proves a resource augmentation bound of 3.42 under partitioned Deadline Monotonic (DM) scheduling. For the synchronous model with arbitrary numbers of threads in segments, our earlier work in [6] proves a resource augmentation bound of 4 and 5 for global EDF and partitioned DM scheduling, respectively. For the *unit-node DAG model* where each node has unit execution requirement, this approach converts each task to a synchronous task, and then applies the same approach.

In this paper, we consider a more general model of deterministic parallel real-time tasks where each task is modeled as a DAG, and different nodes of the DAG may have *different* execution requirements. For preemptive scheduling, in particular, we prove the same resource augmentation bound of 4 as [6]. In addition, we consider *non-preemptive* global EDF scheduling, and prove a resource augmentation bound which, to our knowledge, is the *first* bound for non-preemptive scheduling of parallel tasks.

III. Parallel Task Model

We consider n periodic parallel tasks to be scheduled on a multi-core platform consisting of m identical cores. The task set is represented by $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$. Each task $\tau_i, 1 \leq i \leq n$, is represented as a Directed Acyclic Graph (DAG), where the *nodes* stand for different execution requirements, and the *edges* represent dependencies between the nodes.

A node in τ_i is denoted by $W_i^j, 1 \leq j \leq n_i$, with n_i being the total number of nodes in τ_i. The *execution requirement* of node W_i^j is denoted by E_i^j. A directed edge from node W_i^j to node W_i^k, denoted as $W_i^j \rightarrow W_i^k$, implies that the execution of W_i^k cannot start unless W_i^j has finished execution. W_i^j, in this case, is called a *parent* of W_i^k, while W_i^k is its *child*. A node may have 0 or more parents or children. A node can start execution only after all of its parents have finished execution. Figure 1 shows a task τ_i with $n_i = 10$ nodes.

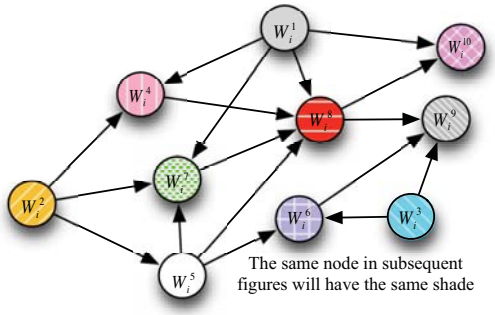


Fig. 1. A parallel task τ_i represented as a DAG

The *total execution requirement* of τ_i is the sum of the execution requirements of all of its nodes, and is denoted by C_i (time units). The *period* of task τ_i is denoted by T_i. The deadline D_i of each task τ_i is considered implicit, i.e., $D_i = T_i$. Task set τ is said to be *schedulable* by algorithm $\mathbb{A}$, if $\mathbb{A}$ can schedule τ such that every $\tau_i \in \tau$ can meet deadline D_i.

IV. Task Decomposition

We consider scheduling parallel tasks by decomposing them into sequential subtasks. This strategy allows us to leverage existing schedulability analysis for multiprocessor scheduling (both preemptive and non-preemptive). In this section, we present the decomposition of a parallel task under general DAG model. The method decomposes a task into nodes. Thus, each node of a task becomes a sequential subtask with execution requirement equal to the execution requirement of the node. All nodes of a DAG are assigned appropriate deadlines and release offsets such that when they execute as individual subtasks all dependencies among them in the DAG (i.e., in the original task) are preserved. Thus, an implicit deadline DAG is decomposed into a set of constrained deadline sequential subtasks with each subtask corresponding to a node of the DAG. We use the terms 'subtask' and 'node' interchangeably.

Note that for schedulability analysis of parallel tasks, conventional utilization bound approaches are not useful [6], [7]. Instead, determining a resource augmentation bound represents a promising approach [6], [7]. A *resource augmentation* quantifies how much we have to increase the processor (core) speed, with respect to an optimal algorithm for the original task set, to guarantee the schedulability of the decomposed tasks. Analysis for bounding this value is mostly based on the *densities* of the decomposed tasks. In the following, we first present terminology used in decomposition. Then, we present the proposed technique for decomposition, followed by a density analysis of the decomposed tasks.

A. Terminology

The *execution requirement* (i.e., the *work*) C_i of task τ_i is the sum of the execution requirements of all nodes in τ_i. Thus, C_i is the *maximum execution time* of task τ_i on a multi-core platform where each processor core has unit speed. That is, C_i is its execution time on a unit-speed single-core processor if it is never preempted. We use $C_{i,\nu}$ to denote the maximum execution time of task τ_i on a multi-core platform where each processor core has speed ν. For τ_i with n_i nodes, each with execution requirement E_i^j, C_i and $C_{i,\nu}$ are expressed as

$$C_i = \sum_{j=1}^{n_i} E_i^j; \qquad C_{i,\nu} = \frac{1}{\nu}\sum_{j=1}^{n_i} E_i^j = \frac{C_i}{\nu} \tag{1}$$

For task τ_i, the *critical path length*, denoted by P_i, is the sum of execution requirements of the nodes on a critical path. A *critical path* is a directed path that has the maximum execution requirement among all other paths in DAG τ_i. Thus, P_i is the *minimum execution time* of task τ_i meaning that it needs at least P_i time units on unit-speed processor cores even when the number of cores m is infinite. Therefore, its deadline T_i (i.e., period) must be no less than P_i.

$$T_i \geq P_i \tag{2}$$

We use $P_{i,\nu}$ to denote the critical path length of task τ_i on a multi-core platform where each processor core has speed ν, which is expressed as $P_{i,\nu} = \frac{P_i}{\nu}$.

The *utilization* u_i of task τ_i, and the *total utilization* $u_{\text{sum}}(\tau)$ for the set of n tasks τ are defined as follows:

$$u_i = \frac{C_i}{T_i}; \qquad u_{\text{sum}}(\tau) = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

If the total utilization u_{sum} is greater than m, then no algorithm can schedule τ on m identical unit-speed processor cores.

The *density* δ_i of task τ_i, and the *total density* $\delta_{\text{sum}}(\tau)$ and the *maximum density* $\delta_{\max}(\tau)$ for the task set τ are given by

$$\delta_i = \frac{C_i}{D_i}; \quad \delta_{\text{sum}}(\tau) = \sum_{i=1}^{n} \delta_i; \quad \delta_{\max}(\tau) = \max\{\delta_i | 1 \leq i \leq n\}$$

The *demand bound function* (DBF) of a task τ_i is the largest cumulative execution requirement of all jobs generated by τ_i that have both arrival times and deadlines within a contiguous interval of t time units. For τ_i, the DBF is given by

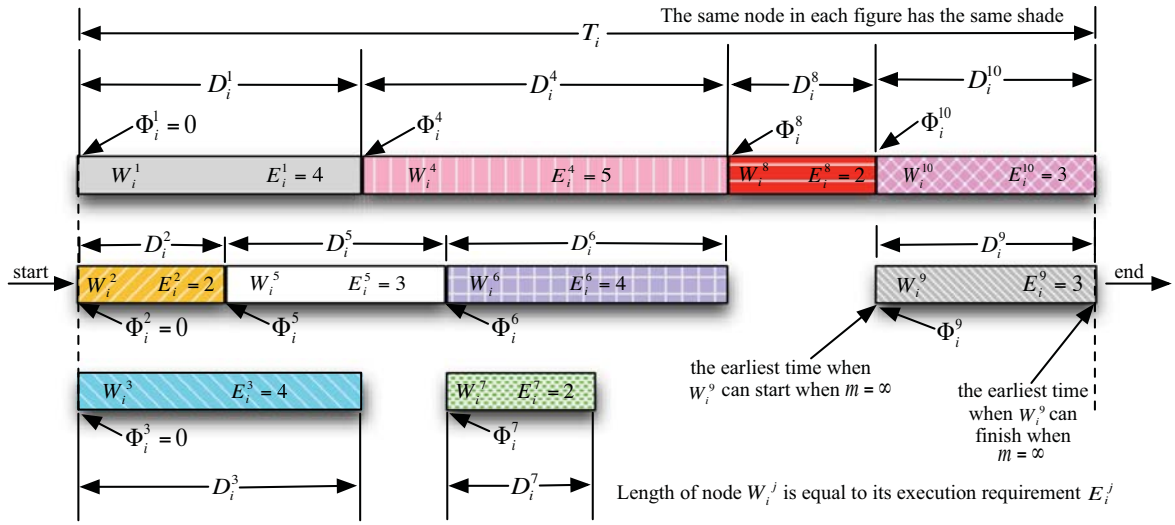
$$DBF(\tau_i, t) = \max\left(0, \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) C_i\right) \tag{3}$$

Based on the DBF, the *load* of the set of n tasks τ, denoted by $\lambda(\tau)$, is defined as follows

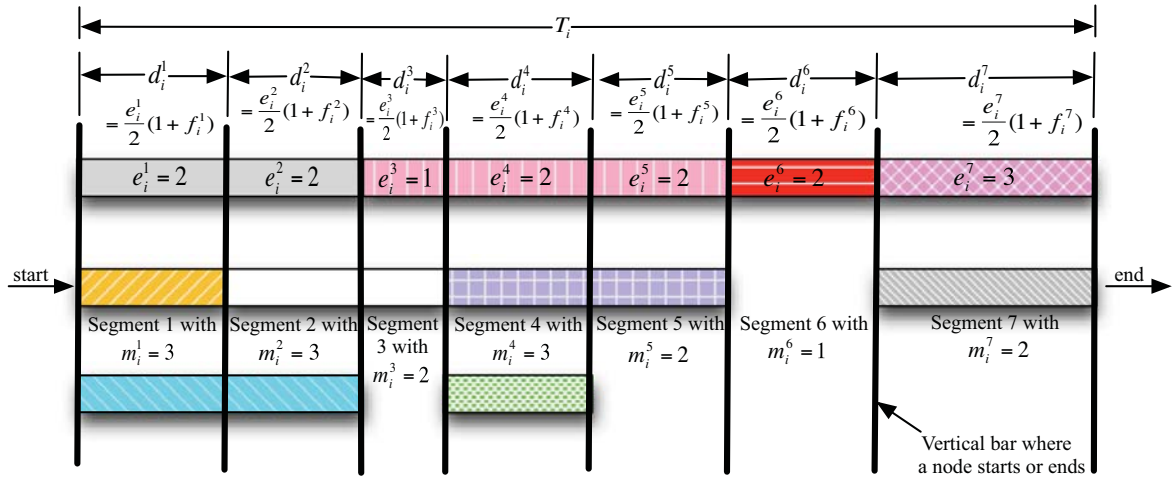
$$\lambda(\tau) = \max_{t>0} \left(\frac{\sum_{i=1}^{n} DBF(\tau_i, t)}{t} \right) \tag{4}$$

B. Decomposition Technique

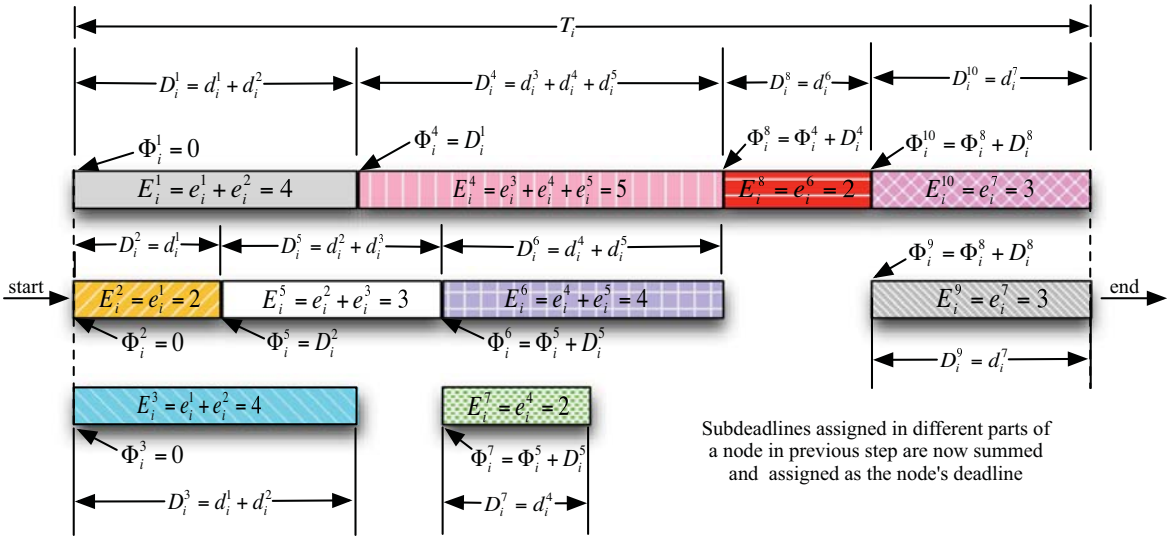
In our decomposition, each node of a task becomes an individual sequential subtask with its own execution requirement and an assigned constrained deadline. To preserve the dependencies in the original DAG, each node is assigned a release offset. Since a node cannot start execution until all of its parents finish, its release offset is equal to the maximum sum of the release offset and deadline among its parents. That is, a node starts after its *latest* parent finishes. The (relative) deadlines of the nodes are assigned by distributing



(a) τ_i^{∞}: a timing diagram for DAG τ_i (of Figure 1) when it executes on an infinite number of processor cores



(b) Slack distribution in τ_i^{syn} (a synchronous model with equal length threads in each segment for τ_i^{∞})



(c) Calculating offset and deadline for each node of τ_i by removing intermediate subdeadlines in the node determined in τ_i^{syn}

Fig. 2. Decomposition of τ_i into nodes by assigning an offset and deadline to each node

the available slack of the task. We calculate the slack for each task considering a multi-core platform where each processor core has speed 2. The *slack* for task τ_i, denoted by L_i, is defined as the difference between its deadline and its critical path length on 2-speed processor cores. That is,

$$L_i = D_i - P_{i,2} = T_i - P_{i,2} = T_i - \frac{P_i}{2} \tag{5}$$

For task τ_i, the deadline and the offset assigned to node W_i^j are denoted by D_i^j and Φ_i^j, respectively. Since we assign slack considering 2-speed processor cores, deadline D_i^j and offset Φ_i^j are also based on 2-speed processor cores. That is, these deadlines may not necessarily be met on unit-speed processor cores. Once appropriate values of D_i^j and Φ_i^j are determined for each node W_i^j (respecting the dependencies in the DAG), task τ_i is decomposed into nodes. Upon decomposition, the dependencies in the DAG need not be considered, and each node can execute as a traditional multiprocessor task. Hence, the decomposition technique for τ_i boils down to determining D_i^j and Φ_i^j for each node W_i^j.

We now present steps to determine D_i^j and Φ_i^j for each node W_i^j of τ_i. Each step is also followed by an example using the DAG τ_i of Figure 1. To do so, we assign an example execution requirement E_i^j to each node W_i^j as $E_i^1 = 4$, $E_i^2 = 2$, $E_i^3 = 4$, $E_i^4 = 5$, $E_i^5 = 3$, $E_i^6 = 4$, $E_i^7 = 2$, $E_i^8 = 2$, $E_i^9 = 3$, $E_i^{10} = 3$. This gives $C_i = 32$, and $P_i = 14$. Period T_i is set to 21.

First, we represent DAG τ_i as a *timing diagram* τ_i^∞ (Figure 2(a)) that shows its execution time on infinite number of unit-speed processor cores. Specifically, τ_i^∞ indicates the earliest start time and the earliest finishing time of each node when $m = \infty$. For any node W_i^j that has no parents, the *earliest start time* and the *earliest finishing time* are 0 and E_i^j, respectively. For every other node W_i^j, the *earliest start time* is the latest finishing time among its parents, and the *earliest finishing time* is E_i^j time units after that. For example, in τ_i of Figure 1, nodes W_i^1, W_i^2, and W_i^3 can start execution at time 0, and their earliest finishing times are 4, 2, and 4, respectively. Node W_i^4 can start after W_i^1 and W_i^2 complete, and finish after 5 time units at its earliest, and so on. Thus, Figure 2(a) shows τ_i^∞ of the DAG τ_i of Figure 1.

Next, based on τ_i^∞, the calculation of D_i^j and Φ_i^j (see Figure 2(a)) for each node W_i^j involves the following two steps. In Step 1, for each node, we distribute slack among different parts of the node. In Step 2, the total slack assigned to different parts of the node is assigned as the node's slack.

1) Step 1 (slack distribution): In DAG τ_i, a node can execute with different numbers of nodes in parallel at different time. Such a degree of parallelism can be approximated based on τ_i^∞. For example, in Figure 2(a), node W_i^5 executes with W_i^1 and W_i^3 in parallel for the first 2 time units, and then executes with W_i^4 in parallel for the next time unit. In this way, we first identify the degrees of parallelism at different parts of each node. Intuitively, the parts of a node that may execute with a large number of nodes in parallel demand more slack. Therefore, different parts of a node are assigned different amounts of slack considering their degrees of parallelism and execution requirements. Later, the sum of slack of all parts of a node is assigned to the node itself.

To identify the degree of parallelism for different portions of a node based on τ_i^∞, we assign slack to a node in different (consecutive) segments. In different segments of a node, the task may have different degrees of parallelism. In τ_i^∞, starting from the left, we draw a vertical line at every time instant where a node starts or ends (as shown in Figure 2(b)). This is done in linear time using a breadth-first search over the DAG. The vertical lines now split τ_i^∞ into segments. For example, in Figure 2(b), τ_i is split into 7 segments (numbered in increasing order from left to right).

Once τ_i^∞ is split into segments, each segment consists of an equal amount of execution by the nodes that lie in the segment. Parts of different nodes in the same segment can now be thought of *threads* that can run in parallel, and the threads in a segment can start only after those in the preceding one finish. Such a model is thus similar to the synchronous task model used in [6]. We denote this model by τ_i^{syn}. We first assign slack to the segments, and finally we add all slack assigned to different segments of a node to calculate its overall slack. Note that *τ_i is never converted to a synchronous model; the procedure only identifies segments to determine slack for nodes, and does not decompose the task at this stage.*

We distribute slack among the nodes based on the number of threads and execution requirement of the segments where a node lies in τ_i^{syn}. We first calculate slack for each segment. Let τ_i^{syn} be a sequence of s_i segments, where the j-th segment is represented by $\langle e_i^j, m_i^j \rangle$, with m_i^j being the number of threads in the segment, and e_i^j being the execution requirement of each thread in the segment (see Figure 2(b)). Since τ_i^{syn} has the same critical path and total execution requirements as those of τ_i, we can now define P_i and C_i in terms of τ_i^{syn}:

$$P_i = \sum_{j=1}^{s_i} e_i^j; \qquad C_i = \sum_{j=1}^{s_i} m_i^j . e_i^j$$

For every j-th segment of τ_i^{syn}, we calculate a value d_i^j, called an *intermediate subdeadline*, so that the segment is assigned a slack value of $d_i^j - \frac{e_i^j}{2}$. That is, each thread in the segment gets this "extra time" $d_i^j - \frac{e_i^j}{2}$ beyond its execution time $\frac{e_i^j}{2}$ on 2-speed processor cores. In the rest of Step 1, we calculate the values d_i^j based on the technique used in [6].

The total slack is L_i (Equation 5). For every j-th segment, a fraction f_i^j of L_i is determined so that each thread in the segment is assigned slack $\frac{e_i^j}{2} f_i^j$, and intermediate subdeadline

$$d_i^j = \frac{e_i^j}{2} + \frac{e_i^j}{2} f_i^j = \frac{e_i^j}{2}(1 + f_i^j) \tag{6}$$

The density of each thread on 2-speed cores then becomes

$$\frac{\frac{e_i^j}{2}}{d_i^j} = \frac{\frac{e_i^j}{2}}{\frac{e_i^j}{2}(1 + f_i^j)} = \frac{1}{1 + f_i^j}$$

Since any j-th segment consists of m_i^j threads, the segment's density on 2-speed processor cores is then $\frac{m_i^j}{1+f_i^j}$.

The segments with larger numbers of threads and with longer threads are computation intensive, and demand more slack. Therefore, for each j-th segment, we determine its slack fraction f_i^j by considering both m_i^j and e_i^j. Each j-th segment with $m_i^j > \frac{C_{i,2}}{T_i - P_{i,2}}$ is classified as a *heavy segment* while other segments are called *light segments*. This leads us to two different scenarios: when τ_i^{syn} has no heavy segments, and when τ_i^{syn} has some heavy segments. Therefore, two different approaches are followed for two scenarios to determine f_i^j.

(a) *When τ_i^{syn} has no heavy segments:* Since each segment has a smaller number of threads ($\leq \frac{C_{i,2}}{T_i - P_{i,2}}$), we only consider the length of a thread in each segment, and assign the slack proportionally among all segments. That is, for j-th segment,

$$f_i^j = \frac{L_i}{P_{i,2}} \tag{7}$$

Then, the intermediate subdeadline d_i^j is given by Equation 6.

(b) *When τ_i^{syn} has some (or all) heavy segments:* In this case, no slack is assigned to the light segments. All available slack L_i is distributed among the heavy segments in a way so that each heavy segment can achieve the same density.

Let τ_i^{syn} have a total of s_i^h heavy segments, each k-th heavy segment denoted $\langle e_i^{k,h}, m_i^{k,h} \rangle$, where $1 \leq k \leq s_i^h$ (superscript h standing for 'heavy'). Similarly, let it have a total of s_i^ℓ light segments, each j-th light segment denoted $\langle e_i^{j,\ell}, m_i^{j,\ell} \rangle$, where $1 \leq j \leq s_i^\ell$ (superscript ℓ standing for 'light'). For any j-th light segment, the slack fraction $f_i^{j,\ell} = 0$. For heavy ones, slack fraction $f_i^{j,h}$ is determined so that

$$\frac{m_i^{1,h}}{1 + f_i^{1,h}} = \frac{m_i^{2,h}}{1 + f_i^{2,h}} = \frac{m_i^{3,h}}{1 + f_i^{3,h}} = \cdots = \frac{m_i^{s_i^h,h}}{1 + f_i^{s_i^h,h}} \tag{8}$$

In addition, since all the slack is distributed among the heavy segments, the following equality must hold.

$$\frac{e_i^{1,h}}{2}.f_i^{1,h} + \frac{e_i^{2,h}}{2}.f_i^{2,h} + \frac{e_i^{3,h}}{2}.f_i^{3,h} + \cdots + \frac{e_i^{s_i^h,h}}{2}.f_i^{s_i^h,h} = L_i \tag{9}$$

Solving Equations 8 and 9 gives (see [6] for details):

$$f_i^{j,h} = \frac{m_i^{j,h}(T_i - P_{i,2}^\ell)}{C_{i,2} - C_{i,2}^\ell} - 1, \quad \forall j, 1 \leq j \leq s_i^h, \text{ where}$$

$$P_{i,2}^\ell = \frac{1}{2}\sum_{j=1}^{s_i^\ell} e_{i,j}^\ell \quad \text{and} \quad C_{i,2}^\ell = \frac{1}{2}\sum_{j=1}^{s_i^\ell} m_{i,j}^\ell . e_{i,j}^\ell$$

Thus, for any j-th segment in τ_i^{syn}, the slack fraction is

$$f_i^j = \begin{cases} 0; & \text{if } m_i^j \leq \frac{C_{i,2}}{T_i - P_{i,2}} \\ \frac{m_i^j (T_i - P_{i,2}^\ell)}{C_{i,2} - C_{i,2}^\ell} - 1; & \text{if } m_i^j > \frac{C_{i,2}}{T_i - P_{i,2}} \end{cases} \tag{10}$$

Then, intermediate subdeadline d_i^j is given by Equation 6. Figure 3(a) shows an example for calculating slacks for different segments of τ_i^{syn} when $T_i = 21$.

2) Step 2 (calculating deadline and offset for nodes): We have assigned intermediate subdeadlines to (the threads of) each segment of τ_i^{syn} in Step 1. Since a node may be split into multiple (consecutive) segments in τ_i^{syn}, now we have to remove all intermediate subdeadlines of a node. Namely, we add all intermediate subdeadlines of a node, and assign the total as the node's deadline.

Now let a node W_i^j of τ_i belong to segments k to r ($1 \leq k \leq r \leq s_i$) in τ_i^{syn}. Therefore, the deadline D_i^j of node W_i^j is calculated as follows (as shown in Figure 2(c)).

$$D_i^j = d_i^k + d_i^{k+1} + \cdots + d_i^r \tag{11}$$

Note that the execution requirement E_i^j of node W_i^j is

$$E_i^j = e_i^k + e_i^{k+1} + \cdots + e_i^r \tag{12}$$

Node W_i^j cannot start until all of its parents complete. Hence, its release offset Φ_i^j is determined as follows (Figure 2(c)).

$$\Phi_i^j = \begin{cases} 0; \text{ if } W_i^j \text{ has no parent} \\ \max\{\Phi_i^l + D_i^l | W_i^l \text{ is a parent of } W_i^j\}; \text{ otherwise.} \end{cases}$$

Now that we have assigned appropriate deadline D_i^j and release offset Φ_i^j to each node W_i^j of τ_i, the DAG τ_i is now decomposed into nodes. Each node W_i^j is now an individual (sequential) multiprocessor subtask with an execution requirement E_i^j, a constrained deadline D_i^j, and a release offset Φ_i^j. Figure 3(b) shows an example of decomposition of τ_i.

C. Density Analysis after Decomposition

After decomposition, let τ_i^{dec} denote all subtasks (i.e., nodes) that τ_i generates. Note that the densities of all such subtasks comprise the density of τ_i^{dec}. Now we analyze the density of τ_i^{dec} which will later be used to analyze schedulability (in terms of resource augmentation bound) upon decomposition.

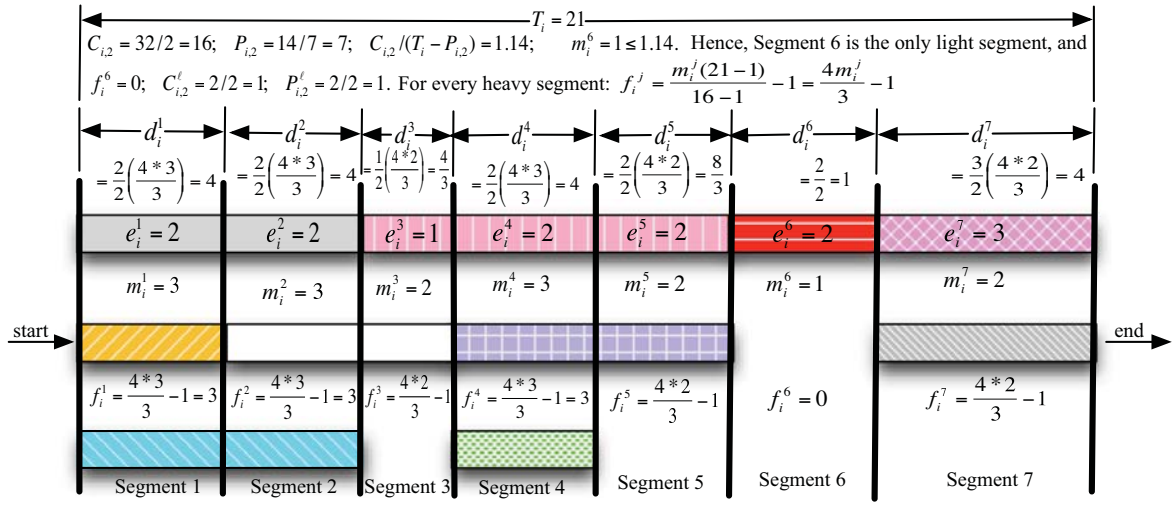
Let node W_i^j of τ_i belong to segments k to r ($1 \leq k \leq r \leq s_i$) in τ_i^{syn}. Since W_i^j has been assigned deadline D_i^j, by Equations 11 and 12, its density $\delta_{i,2}^j$ after decomposition on 2-speed processor cores is

$$\delta_{i,2}^j = \frac{E_i^j/2}{D_i^j} = \frac{(e_i^k + e_i^{k+1} + \cdots + e_i^r)/2}{d_i^k + d_i^{k+1} + \cdots + d_i^r} \tag{13}$$

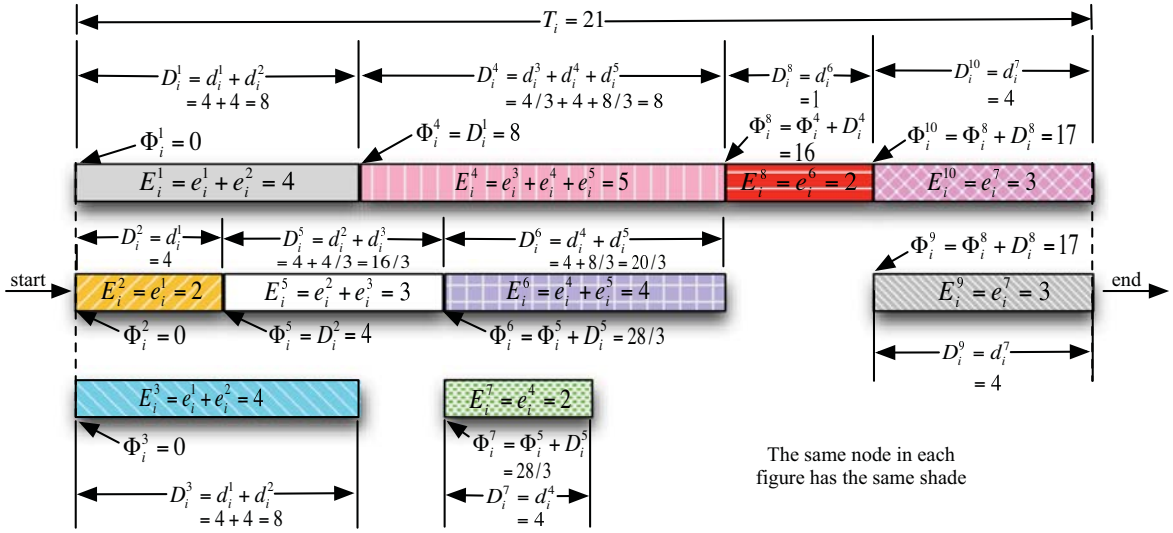
Let τ^{dec} be the set of all generated subtasks of all original DAG tasks, and $\delta_{\max,2}$ be the *maximum density* among all subtasks in τ^{dec} on 2-speed processor cores. By Equations 7 and 10, the value of the slack assigned to each subtask W_l^j in τ^{dec} is non-negative, i.e., $\frac{E_l^j}{2} \leq D_l^j$. Hence,

$$\delta_{\max,2} = \max\{\delta_{l,2}^j | W_l^j \text{ is a subtask in } \tau^{\text{dec}}\} \leq 1 \tag{14}$$

Note that we represent a DAG τ_i as τ_i^{syn} in Step 1. This τ_i^{syn} is a sequence of segments, each segment consisting of a set of equal-length threads (see Figure 2(b)). As noted, τ_i^{syn} is exactly the same as the synchronous task model used in [6]. In Step 1, we assign subdeadlines to different segments of τ_i^{syn} using the same approach as [6]. According to [6], τ_i^{syn} can be decomposed into threads as follows: each thread becomes



(a) Calculating slacks for different segments of τ_i^{syn}



(b) Calculating deadline and offset for nodes of τ_i

Fig. 3. An example for decomposition of τ_i (shown in Figure 1) when $T_i = 21$

a sequential subtask; all threads of j-th segment are assigned execution requirement e_i^j, and deadline d_i^j; all threads of the first segment are assigned release offset 0, and those of any other j-th segment are assigned offset $d_i^1 + d_i^2 + \cdots + d_i^{j-1}$. Theorem 1 states the density of τ_i^{syn} denoted by $\delta_{i,2}^{\text{syn}}$ after such decomposition on 2-speed processor cores as proved in [6].

Theorem 1. *(From [6]) If any τ_i^{syn}, $1 \le i \le n$, is decomposed into threads in all segments, and if $\delta_{i,2}^{\text{syn}}$ is the density of these decomposed threads of τ_i^{syn} on 2-speed processor cores, then $\delta_{i,2}^{\text{syn}} \le \frac{C_i/2}{T_i - P_i/2}$.*

Theorem 2 proves that, after our proposed decomposition of a DAG τ_i into nodes, its density remains no greater than $\delta_{i,2}^{\text{syn}}$ on 2-speed processors cores.

Theorem 2. *Let a DAG τ_i, $1 \le i \le n$, with period T_i, critical path length P_i, and maximum execution requirement C_i be decomposed into subtasks (nodes) denoted τ_i^{dec} using the proposed decomposition. The density of τ_i^{dec} on 2-speed processor cores is at most $\frac{C_i/2}{T_i - P_i/2}$.*

Proof: Since we decompose τ_i into nodes (i.e., subtasks), the densities of all decomposed nodes W_i^j, $1 \le j \le n_i$, comprise the density of τ_i^{dec}. In Step 1, every node W_i^j of τ_i is split into threads in different segments of τ_i^{syn}, and each thread is assigned an intermediate subdeadline. In Step 2, we remove the intermediate subdeadlines in the node, and their total is assigned as the node's deadline. By Theorem 1, if we decompose without removing the intermediate subdeadlines in the nodes, then the density of τ_i after such decomposition on 2-speed processor cores is $\delta_{i,2}^{\text{syn}} \le \frac{C_i/2}{T_i - P_i/2}$. Hence, it is sufficient to prove that removing intermediate subdeadlines in the nodes does not increase the task's overall density. That is, it is sufficient to prove that the density $\delta_{i,2}^j$ (Equation 13) of any node W_i^j after removing its intermediate subdeadlines is no greater than the density $\delta_{i,2}^{j,\text{syn}}$ that it had before removing

its intermediate subdeadlines.

Let node W_i^j of τ_i be split into threads in segments k to r ($1 \leq k \leq r \leq s_i$) in τ_i^{syn}. Since the total density of any set of tasks is an upper bound on its load (proven in [29]), the load of the threads of W_i^j must be no greater than the total density of these threads. Since each of these threads is executed only once in the interval of D_i^j, by Equation 3, the DBF of the thread, $thread_i^l$, in segment l, $k \leq l \leq r$, in the interval D_i^j on 2-speed processor cores is given by

$$DBF(thread_i^l, D_i^j) = \frac{e_i^l}{2}$$

Therefore, using Equation 4, the load, denoted by $\lambda_{i,2}^{j,\text{syn}}$, of the threads of W_i^j in τ_i^{syn} on 2-speed cores for interval D_i^j is

$$\lambda_{i,2}^{j,\text{syn}} \geq \frac{\frac{e_i^k}{2}}{D_i^j} + \frac{\frac{e_i^{k+1}}{2}}{D_i^j} + \cdots + \frac{\frac{e_i^r}{2}}{D_i^j} = \frac{E_i^j/2}{D_i^j} = \delta_{i,2}^j$$

Since $\delta_{i,2}^{\text{j, syn}} \geq \lambda_{i,2}^{\text{j, syn}}$, for any W_i^j, we have $\delta_{i,2}^{\text{j, syn}} \geq \delta_{i,2}^j$. ■

Let $\delta_{\text{sum},2}$ be the *total density* of all subtasks τ^{dec} on 2-speed processor cores. Then, from Theorem 2,

$$\delta_{\text{sum},2} \leq \sum_{i=1}^{n} \frac{C_i/2}{T_i - P_i/2} \tag{15}$$

V. Preemptive Global EDF Scheduling

Once all DAG tasks are decomposed into nodes (i.e., subtasks), we consider scheduling the nodes. Since every node after decomposition becomes a sequential multiprocessor task, we schedule them using traditional multiprocessor scheduling policies. In this section, we consider preemptive global Earliest Deadline First (EDF) scheduling of the decomposed subtasks.

Lemma 3. *For any set of DAG model parallel tasks $\tau = \{\tau_1, \cdots, \tau_n\}$, let τ^{dec} be the decomposed task set. If τ^{dec} is schedulable under some preemptive scheduling, then τ is also preemptively schedulable.*

Proof: In each τ_i^{dec}, a node (i.e., a subtask) is released only after all of its parents finish execution. Hence, the precedence relations in original task τ_i are retained in τ_i^{dec}. Besides, for each τ_i^{dec}, the deadline and the execution requirement are the same as those of original task τ_i. Hence, if τ^{dec} is preemptively schedulable, then a preemptive schedule must exist for τ where each task in τ meets its deadline. ■

To schedule the decomposed subtasks τ^{dec}, the EDF policy is the same as the traditional global EDF policy where jobs with earlier absolute deadlines have higher priorities. Due to the *preemptive* policy, a job can be suspended (preempted) at any time by arriving higher-priority jobs, and is later resumed with (in theory) no cost or penalty. Under preemptive global EDF, we now present a schedulability analysis for τ^{dec} in terms of a resource augmentation bound which, by Lemma 3, is also a sufficient analysis for the original DAG task set τ. For a task set, the *resource augmentation bound* ν of a scheduling policy $\mathbb{A}$ on a multi-core processor with m cores represents a processor speedup factor. That is, if there exists any way to schedule the task set on m identical unit-speed processor cores, then $\mathbb{A}$ is guaranteed to successfully schedule it on an m-core processor with each processor core being ν times as fast as the original.

Our analysis hinges on a result (Theorem 4) for preemptive global EDF scheduling of constrained deadline sporadic tasks on traditional multiprocessor platform [30]. This result is a generalization of the result for implicit deadline tasks [31].

Theorem 4. *(From [30]) Any constrained deadline sporadic task set π with total density $\delta_{\text{sum}}(\pi)$ and maximum density $\delta_{\max}(\pi)$ is schedulable using preemptive global EDF strategy on m unit-speed processor cores if*

$$\delta_{\text{sum}}(\pi) \leq m - (m-1)\delta_{\max}(\pi)$$

Since τ^{dec} also consists of constrained deadline (sub)tasks that are periodic (with offsets), the above result holds for τ^{dec}. We now use the results of density analysis from Subsection IV-C and prove in Theorem 5 that τ^{dec} is guaranteed to be schedulable with a resource augmentation of at most 4. The proof of Theorem 5 is similar to the proof used in [6] .

Theorem 5. *For any set of DAG model parallel tasks $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$, let τ^{dec} be the decomposed task set. If there exists any algorithm that can schedule τ on m unit-speed processor cores, then τ^{dec} is schedulable under preemptive global EDF on m processor cores, each of speed 4.*

Proof: If τ is schedulable on m identical unit-speed processor cores, the following condition must hold.

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq m \tag{16}$$

We decompose tasks considering that each processor core has speed 2. To be able to schedule the decomposed tasks τ^{dec}, suppose we need to increase the speed of each core ν times further. That is, we need each core to be of speed 2ν. On an m-core platform where each core has speed 2ν, let the total density and the maximum density of task set τ^{dec} be denoted by $\delta_{\text{sum},2\nu}$ and $\delta_{\max,2\nu}$, respectively. From 14, we have

$$\delta_{\max,2\nu} = \frac{\delta_{\max,2}}{\nu} \leq \frac{1}{\nu} \tag{17}$$

Based on Equations 2 and 16, when each processor core is of speed 2ν, the total density of τ^{dec} can be written from 15 as

$$\delta_{\text{sum},2}^{\nu} \leq \sum_{i=1}^{n} \frac{\frac{C_i}{2\nu}}{T_i - \frac{P_i}{2}} \leq \frac{\frac{C_i}{2\nu}}{T_i - \frac{T_i}{2}} = \frac{1}{\nu} \sum_{i=1}^{n} \frac{C_i}{T_i} \leq \frac{m}{\nu} \tag{18}$$

Using Equations 17 and 18 in Theorem 4, τ^{dec} is schedulable under preemptive EDF on m cores each of speed 2ν if

$$\frac{m}{\nu} \leq m - (m-1)\frac{1}{\nu} \;\Leftrightarrow\; \frac{2}{\nu} - \frac{1}{m\nu} \leq 1$$

From the above condition, τ^{dec} must be schedulable if

$$\tfrac{2}{\nu} \leq 1 \quad \Leftrightarrow \quad \nu \geq 2 \quad \Leftrightarrow \quad 2\nu \geq 4$$ ■

VI. Non-Preemptive Global EDF Scheduling

We now consider non-preemptive global EDF scheduling. The original task set τ is scheduled based on node-level non-preemption. In *node-level non-preemptive scheduling*, whenever the execution of a node in a DAG starts, the node's execution cannot be preempted by any task. Most parallel languages and libraries have yield points at the ends of threads (nodes of the DAG). Therefore, they allow low cost, user-space preemption at the end of threads. For these languages and libraries, schedulers that require preemption only when threads end can be implemented entirely in user-space (without interaction with the kernel), and therefore have low overheads.

The decomposition converts each node of a DAG to a traditional multiprocessor (sub)task. Therefore, we consider fully non-preemptive global EDF scheduling of the decomposed tasks. Namely, once a job of a decomposed (sub)task starts execution, it cannot be preempted by any other job.

Lemma 6. *For a set of DAG parallel tasks $\tau = \{\tau_1, \cdots, \tau_n\}$, let τ^{dec} be the decomposed task set. If τ^{dec} is schedulable under some fully non-preemptive scheduling, then τ is schedulable under node-level non-preemption.*

Proof: Since the decomposition converts each node of a DAG to an individual task, a fully non-preemptive scheduling of τ^{dec} preserves the node-level non-preemptive behavior of task set τ. The rest of the proof follows from Lemma 3. ■

Under non-preemptive global EDF, we now present a schedulability analysis for τ^{dec} in terms of a resource augmentation bound which, by Lemma 6, is also a sufficient analysis for the DAG task set τ. This analysis exploits Theorem 7 for non-preemptive global EDF scheduling of constrained deadline periodic tasks on traditional multiprocessor. The theorem is a generalization of the result for implicit deadline tasks [32].

For a task set π, let $C_{\max}(\pi)$ and $D_{\min}(\pi)$ be the maximum execution requirement and the minimum deadline among all tasks in π. In non-preemptive scheduling, $C_{\max}(\pi)$ represents the *maximum blocking time* that a task may experience, and plays major role in schedulability. Hence, a *non-preemption overhead* [32] $\rho(\pi) = \frac{C_{\max}(\pi)}{D_{\min}(\pi)}$.

Theorem 7. *(From [32]) Any constrained deadline periodic task set π with total density $\delta_{\text{sum}}(\pi)$, maximum density $\delta_{\max}(\pi)$, and non-preemption overhead $\rho(\pi)$ is schedulable using non-preemptive global EDF on m unit-speed cores if*

$$\delta_{\text{sum}}(\pi) \leq m\big(1 - \rho(\pi)\big) - (m-1)\delta_{\max}(\pi)$$

Let $E_{\max}$ and $E_{\min}$ be the maximum and minimum execution requirement, respectively, among all nodes of all DAG tasks. In non-preemptive scheduling of decomposed subtasks τ^{dec}, the non-preemption overhead ρ on 2-speed processor cores is given by $\rho \leq \frac{E_{\max}}{E_{\min}}$. The overhead on unit-speed processor cores is then 2ρ. Using an analysis similar to Section V, Theorem 8 derives a resource augmentation bound of $4 + 2\rho$ for non-preemptive global EDF scheduling of τ^{dec}.

Theorem 8. *For DAG model parallel tasks $\tau = \{\tau_1, \cdots, \tau_n\}$, let τ^{dec} be the decomposed task set with non-preemption overhead ρ. If there exists any way to schedule τ on m unit-speed processor cores, then τ^{dec} is schedulable under non-preemptive global EDF on m cores, each of speed $4 + 2\rho$.*

Proof: Similar to Theorem 5, suppose we need each processor core to be of speed 2ν to be able to schedule the decomposed tasks τ^{dec}. Since the non-preemption overhead of τ^{dec} on 2-speed cores is ρ, on 2ν-speed cores it is ρ/ν. Using Equations 17 and 18 in Theorem 7, τ^{dec} is schedulable under non-preemptive EDF on m cores each of speed 2ν if

$$\frac{m}{\nu} \leq m(1 - \frac{\rho}{\nu}) - (m-1)\frac{1}{\nu} \;\Leftrightarrow\; \frac{2+\rho}{\nu} - \frac{1}{m\nu} \leq 1$$

From the above condition, task set τ^{dec} is schedulable if $\frac{2+\rho}{\nu} \leq 1 \;\Leftrightarrow\; \nu \geq 2 + \rho \;\Leftrightarrow\; 2\nu \geq 4 + 2\rho$ ■

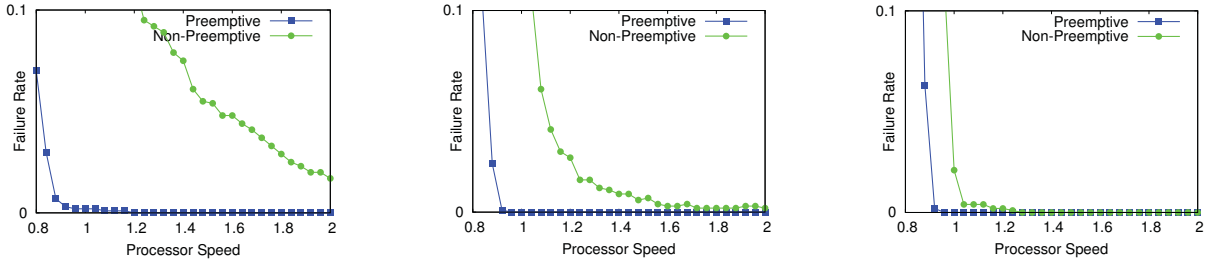
VII. Evaluation

In this section, we describe some preliminary simulation studies we have conducted to validate our bounds. While these are small-scale studies, they seem to indicate that not only are the theoretical bounds easily met, but also they are in fact quite loose, primarily for non-preemptive scheduling. In particular, in our experiments most task sets require augmentation of less than 2 and all require augmentation of less than 3.

In our studies, DAGs are generated by first fixing the number of nodes in the graph and then adding edges until it becomes weakly connected. Nodes are assigned random execution requirements from a given range. Each task is assigned a valid harmonic period. To generate a task set, we keep adding tasks to the set as long as their total utilization upper bound (Equation 16) is still satisfied. Each result is generated using at least 1000 task sets.

For the first set of simulations, execution requirements of the nodes in DAGs are in a range $[50, 100]$ (making the non-preemption overhead $\rho = 2$), and the average parallelism of tasks (C_i/P_i) is about 3.4. We test using 4, 8, and 16 processor cores, and task sets have an average utilization of 3.13, 7.15, and 15.03, respectively. For every case, the decomposed subtasks are scheduled under both preemptive and non-preemptive EDF considering different speeds of the cores. Figure 4 shows the failure rates (i.e., the ratio of the number of unschedulable task sets to the total number of task sets) as the processor speed increases. Under preemptive EDF, all task sets are schedulable at speed 1.20, 0.92, and 0.96 respectively for 4, 8, and 16 processor cores. Under nonpreemptive scheduling, the tasks require an augmentation of 3 (not shown to preserve resolution), 2, and 1.3 respectively.

In the second set of simulations, we set the number of cores to 16 and test the effect of non-preemption overhead (ρ) on our decomposition (results shown in Figure 5). To achieve a value of 1, 2, 5, and 10 for ρ, we assign execution requirements from ranges $[50, 50]$, $[50, 100]$, $[50, 250]$, and $[50, 500]$, respectively. Our results indicate that all tasks are schedulable at speed of just 2, except when $\rho = 1$ where a few test cases required speed more than 2 (up to 3). Surprisingly, contrary to the theoretical bounds, higher values of ρ require a smaller augmentation. We suspect that this might be due to



(a) $m = 4$ (b) $m = 8$ (c) $m = 16$

Fig. 4. Failure rate under varying processor speed-up factor

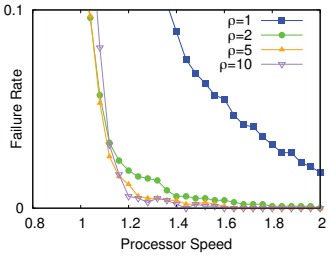


Fig. 5. Failure rate under different non-preemption overheads

the particular method we use to generate DAGs, since in our method, when ρ is smaller, the number of tasks in the task set may be larger, making them more difficult to schedule.

VIII. Conclusions

As multi-core technology becomes mainstream in processor design, real-time scheduling of parallel tasks is crucial to exploit its potential. In this paper, we consider a general task model and through a novel task decomposition, we prove a resource augmentation bound of 4 for preemptive scheduling and 4 plus a non-preemption overhead for non-preemptive EDF scheduling. To our knowledge, these are the first bounds for real-time scheduling of general DAG model tasks. Through simulations, we have observed that bounds in practice are significantly smaller than the theoretical bounds.

These results suggest many directions of future work. First, the simulations indicate that the bounds may be loose, especially for non-preemptive scheduling. We can try to provide better bounds and/or provide lower bound arguments that suggest that the bounds are in fact tight. Second, we can study the effect of caches on scheduling overhead. Requiring non-preemption mitigates this problem to a certain extent, but more can be done to optimize cache-locality. Finally, we have ignored the effects of locks and other forms of non-deterministic synchronization in this paper. Generalizing these bounds to some of those models would be very interesting.

References

[1] http://techresearch.intel.com/ProjectDetails.aspx?Id=151.

[2] www.amd.com/us/products/server/processors.

[3] www.clearspeed.com/newsevents/news/ClearSpeed_Ace_011708.php.

[4] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comp. Surv.*, vol. 43, pp. 35:1–44, 2011.

[5] H.-M. Huang, T. Tidwell, C. Gill, C. Lu, X. Gao, and S. Dyke, "Cyber-physical systems for real-time hybrid structural testing: a case study," in *ICCPS '10*.

[6] A. Saifullah, K. Agrawal, C. Lu, and C. Gill, "Multi-core real-time scheduling for generalized parallel task models," in *RTSS '11*.

[7] K. Lakshmanan, S. Kato, and R. R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *RTSS '10*.

[8] "OpenMP," http://openmp.org.

[9] "Intel CilkPlus," http://software.intel.com/en-us/articles/intel-cilk-plus.

[10] C. D. Polychronopoulos and D. J. Kuck, "Guided self-scheduling: A practical scheduling scheme for parallel supercomputers," *IEEE Transactions on Computers*, vol. C-36, no. 12, pp. 1425–1439, 1987.

[11] M. Drozdowski, "Real-time scheduling of linear speedup parallel tasks," *Inf. Process. Lett.*, vol. 57, no. 1, pp. 35–40, 1996.

[12] X. Deng, N. Gu, T. Brecht, and K. Lu, "Preemptive scheduling of parallel jobs on multiprocessors," in *SODA '96*.

[13] N. S. Arora, R. D. Blumofe, and C. G. Plaxton, "Thread scheduling for multiprogrammed multiprocessors," in *SPAA '98*.

[14] N. Bansal, K. Dhamdhere, J. Konemann, and A. Sinha, "Non-clairvoyant scheduling for minimizing mean slowdown," *Algorithmica*, vol. 40, no. 4, pp. 305–318, 2004.

[15] J. Edmonds, D. D. Chinn, T. Brecht, and X. Deng, "Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics," *Journal of Scheduling*, vol. 6, no. 3, pp. 231–250, 2003.

[16] K. Agrawal, Y. He, W. J. Hsu, and C. E. Leiserson, "Adaptive task scheduling with parallelism feedback," in *PPoPP '06*.

[17] J. M. Calandrino and J. H. Anderson, "On the design and implementation of a cache-aware multicore real-time scheduler," in *ECRTS '09*.

[18] ——, "Cache-aware real-time scheduling on multicore platforms: Heuristics and a case study," in *ECRTS '08*.

[19] J. M. Calandrino, J. H. Anderson, and D. P. Baumberger, "A hybrid real-time scheduling approach for large-scale multicore platforms," in *ECRTS '07*.

[20] J. H. Anderson and J. M. Calandrino, "Parallel real-time task scheduling on multicore platforms," in *RTSS '06*.

[21] Q. Wang and K. H. Cheng, "A heuristic of scheduling parallel tasks and its analysis," *SIAM J. Comput.*, vol. 21, no. 2, pp. 281–294, 1992.

[22] O.-H. Kwon and K.-Y. Chwa, "Scheduling parallel tasks with individual deadlines," *Theor. Comput. Sci.*, vol. 215, no. 1-2, pp. 209–223, 1999.

[23] C.-C. Han and K.-J. Lin, "Scheduling parallelizable jobs on multiprocessors," in *RTSS '89*.

[24] K. Jansen, "Scheduling malleable parallel tasks: An asymptotic fully polynomial time approximation scheme," *Algorithmica*, vol. 39, no. 1, pp. 59–81, 2004.

[25] W. Y. Lee and H. Lee, "Optimal scheduling for real-time parallel tasks," *IEICE Trans. Inf. Syst.*, vol. E89-D, no. 6, pp. 1962–1966, 2006.

[26] S. Collette, L. Cucu, and J. Goossens, "Integrating job parallelism in real-time scheduling theory," *Inf. Process. Lett.*, vol. 106, no. 5, pp. 180–187, 2008.

[27] G. Manimaran, C. S. R. Murthy, and K. Ramamritham, "A new approach for scheduling of parallelizable tasks inreal-time multiprocessor systems," *Real-Time Syst.*, vol. 15, no. 1, pp. 39–60, 1998.

[28] S. Kato and Y. Ishikawa, "Gang EDF scheduling of parallel task systems," in *RTSS '09*.

[29] N. Fisher, T. P. Baker, and S. Baruah, "Algorithms for determining the demand-based load of a sporadic task system," in *RTCSA '06*.

[30] S. Baruah, "Techniques for multiprocessor global schedulability analysis," in *RTSS '07*.

[31] J. Goossens, S. Funk, and S. Baruah, "Priority-driven scheduling of periodic task systems on multiprocessors," *Real-Time Syst.*, vol. 25, no. 2-3, pp. 187–205, 2003.

[32] S. Baruah, "The non-preemptive scheduling of periodic tasks upon multiprocessors," *Real-Time Syst.*, vol. 32, pp. 9–20, 2006.