Multi-core Real-Time Scheduling for Generalized Parallel Task Models

Abusayeed Saifullah, Kunal Agrawal, Chenyang Lu, and Christopher Gill
Department of Computer Science and Engineering
Washington University in St. Louis
{saifullaha, kunal, lu, cdgill}@cse.wustl.edu

***Abstract*—Multi-core processors offer a significant performance increase over single core processors. Therefore, they have the potential to enable real-time applications that must complete large amounts of computation within stringent timing constraints that cannot be met on traditional single-core processors. However, most results in traditional multiprocessor real-time scheduling are limited to sequential programming models and ignore task parallelism. In this paper, we address the problem of scheduling periodic parallel tasks with implicit deadlines on multi-core processors. We first consider a synchronous task model where each task consists of segments containing equal length threads, and all threads of a segment synchronize at the end of the segment. The segments can have an arbitrary number of threads. We propose a new task decomposition method that decomposes each synchronous parallel task into a set of sequential tasks. We prove that our task decomposition achieves a resource augmentation bound of 2.62 when these decomposed tasks are scheduled using global EDF scheduling, and 3.42 when they are scheduled using partitioned deadline monotonic scheduling. Finally, we extend our analysis to directed acyclic graph tasks. We show how these tasks can be converted into synchronous tasks such that the same transformation can be applied and the same augmentation bounds hold.**

I. Introduction

In recent years, multi-core processor technology has improved dramatically as chip manufacturers try to boost performance while minimizing power consumption. This development has shifted the scaling trends from processor clock frequencies to the number of cores per processor. For example, Intel has recently put 80 cores in a Teraflops Research Chip [**?**] with a view to making it generally available, and ClearSpeed has developed a 96-core processor [**?**]. While hardware technology is moving at a rapid pace, software and programming models have failed to keep pace. For example, Intel has set a time frame of 5 years to make their 80-core processor generally available due to the inability of current operating systems and software to exploit the benefits of multi-core processors [?].

As multi-core processors continue to scale, they provide an opportunity for performing more complex and computation-intensive tasks in real time. However, to take full advantage of multi-core processing, these systems must exploit intra-task parallelism, where parallelizable real-time tasks can utilize multiple cores at the same time. By exploiting intra-task parallelism, multi-core processors can achieve significant real-time performance improvement over traditional single-core processors for many computation-intensive real-time applications such as video surveillance, radar tracking, and hybrid real-time structural testing [**?**] where the performance limitations of traditional single-core processors had been a major hurdle.

The growing importance of parallel task models to real-time applications poses new challenges to real-time scheduling theory that has mostly focused on sequential task models. Notably, the state-of-the-art work [**?**] on parallel scheduling for real-time tasks proves a *resource augmentation bound* of 3.42 using partitioned Deadline Monotonic (DM) scheduling. It considers a *restrictive* synchronous task model, where each parallel task consists of a series of sequential or parallel segments. We call this model *synchronous*, since all the threads of a parallel segment must finish before the next segment starts, creating a synchronization point. However, that task model is *restrictive* in that, for every task, all the parallel segments in the task have the *same* number of parallel threads, and the number of parallel threads per segment must be *no greater* than the total number of processor cores.

While the work presented in [?] represents a promising step towards parallel real-time scheduling on multi-core processors, the restrictions on the task model make the solutions unsuitable for many real-time applications that often employ different numbers of threads in different segments of computation. In addition, the analysis presented in [?] only provides the resource augmentation bound under partitioned DM scheduling and does not consider other scheduling policies such as global EDF. To overcome these limitations of the state-of-the-art in parallel real-time scheduling, we consider a more general synchronous task model in this paper. Our tasks still contain segments, and the threads of the segment must still synchronize at the end of each segment. However, in contrast to the restrictive task model addressed in [?], for any task in our model, each segment can contain an *arbitrary* number of parallel threads. That is, different segments of the same parallel task can contain different numbers of threads, and segments can contain more threads than the number of processor cores. Therefore, this model is more portable, since the same task can be executed on machines with small as well as large numbers of cores. Specifically, our work makes the following new contributions to real-time scheduling for generalized parallel task models:

- For the general synchronous task model, we propose a task decomposition algorithm that converts this implicit

deadline parallel task into a set of constrained deadline sequential tasks.
- We derive a resource augmentation bound of 2.62 when these decomposed tasks are scheduled using global EDF scheduling. To our knowledge, this is the first resource augmentation bound for global EDF scheduling of parallel tasks..
- Using the proposed task decomposition, we also derive a resource augmentation bound of 3.42 for partitioned DM scheduling. This bound is the same as the one given in [?], but applies to a more general synchronous task model.
- Finally, we extend our analyses for a Directed Acyclic Graph (DAG) task model, an even more general model for parallel tasks. In particular, we show that we can transform DAG-tasks into synchronous tasks, and then use our proposed decomposition to get the same resource augmentation bounds for DAG tasks.

In the rest of the paper, Section II describes the parallel synchronous task model. Section III presents the proposed task decomposition. Section IV presents the global EDF scheduling, and analysis. Section V presents the analysis for partitioned DM scheduling. Section VI extends our results and analyses for DAG task models. Section VII reviews the related works. Finally, we conclude in Section VIII.

II. Parallel Synchronous Task Model

In this paper, we primarily consider a synchronous task model, where each parallel job consists of many segments, and each segment may contain many parallel threads which synchronize at the end of the segment. Such tasks are generated by parallel *for* loops, a construct common to many parallel languages such as OpenMP [**?**], Intel's CilkPlus [**?**], etc.

More precisely, we consider a set of n synchronous, implicit-deadline, periodic, parallel tasks denoted by $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$. Each task τ_i, $1 \leq i \leq n$, is a sequence of s_i segments, where the j-th segment is represented by $\langle P_{i,j}, m_{i,j} \rangle$, with $m_{i,j}$ being the number of threads in the segment, and $P_{i,j}$ being the worst case execution requirement of each of its threads. When $m_{i,j} > 1$, the threads in the j-th segment can be executed in parallel on different cores. The j-th segment starts only after all threads of $(j-1)$-th segment have completed. Thus, each parallel task τ_i is represented as follows (Figure 1):

$\tau_i : (\langle P_{i,1}, m_{i,1} \rangle, \langle P_{i,2}, m_{i,2} \rangle, \cdots, \langle P_{i,s_i}, m_{i,s_i} \rangle)$

where

- s_i is the total number of computation segments (both parallel and sequential) in task τ_i.
- In a segment $\langle P_{i,j}, m_{i,j} \rangle$, where $1 \leq j \leq s_i$, $P_{i,j}$ is the worst case execution requirement of each thread, and $m_{i,j}$ is the number of threads. Therefore, any segment $\langle P_{i,j}, m_{i,j} \rangle$ with $m_{i,j} > 1$ is a *parallel segment* with a total of $m_{i,j}$ parallel threads, and any segment $\langle P_{i,j}, m_{i,j} \rangle$ with $m_{i,j} = 1$ is a *sequential segment* since it has only one thread. A task τ_i with $s_i = 1$ and $m_{i,s_i} = 1$ is a sequential task.

The period of task τ_i is denoted by T_i. The deadline D_i of task τ_i is equal to its period T_i. Each task τ_i generates an infinite sequence of jobs, with arrival times of successive jobs separated by T_i time units. Jobs are fully independent and preemptive: any job can be suspended (preempted) at any time instant, and it is later resumed with no cost or penalty. The total number of processor cores is represented by q.

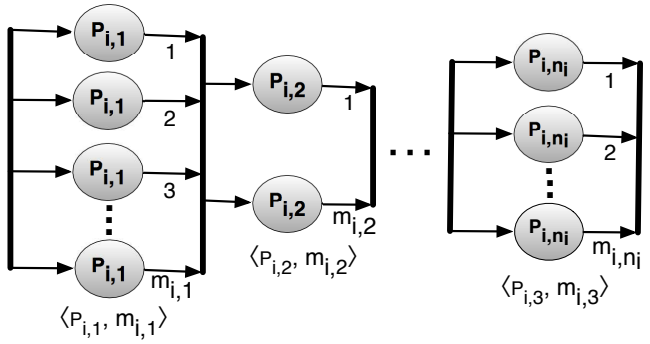


Fig. 1. A parallel synchronous task τ_i

III. Task Decomposition

In this section, we present a decomposition of our parallel tasks into a set of sequential tasks. In particular, we propose a strategy that converts the implicit deadline parallel task into a set of constrained deadline sequential tasks by converting each thread of the parallel task into its own sequential task and assigning appropriate deadlines to these tasks. This strategy then allows us to use existing bounds on multiprocessor scheduling (both global and partitioned) to prove schedulability bounds for parallel tasks, which we discuss in Sections IV and V. In this section, we first present some useful terminology. We then present our decomposition, and a density analysis for it. Finally, we compare our decomposition with the one presented by Lakshmanan et al. [?].

A. Terminology

Definition 1: The *minimum execution time (critical path length)* P_i of task τ_i is defined as follows:

$$P_i = \sum_{j=1}^{s_i} P_{i,j} \tag{1}$$

Observation 1: Any task τ_i requires at least P_i units of time even when the number of cores q is infinite.

Definition 2: The *maximum execution time (work)* C_i of task τ_i defined as follows:

$$C_i = \sum_{j=1}^{s_i} m_{i,j} P_{i,j} \tag{2}$$

That is, C_i is execution time of τ_i on a single core processor if it is never preempted.

Definition 3: The *utilization* of task τ_i is denoted by u_i and is defined as the ratio of its maximum execution requirement to its period:

$$u_i = \frac{C_i}{T_i} \tag{3}$$

Definition 4: For the set of n tasks τ, the *total utilization* is denoted by $u_{sum}(\tau)$ and is defined as follows:

$$u_{sum}(\tau) = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{4}$$

Observation 2: If the total utilization u_{sum} is greater than q, then no scheduling algorithm exists that can schedule τ on q identical unit speed processor cores.

Definition 5: The *density* [**?**] of a task τ_i, denoted by δ_i, is the ratio of its maximum execution requirement to its deadline:

$$\delta_i = \frac{C_i}{D_i} \tag{5}$$

For an implicit-deadline task τ_i, $\delta_i = u_i$.

Definition 6: For the set of n tasks τ, the *total density* is denoted by $\delta_{sum}(\tau)$ and is defined as follows:

$$\delta_{sum}(\tau) = \sum_{i=1}^{n} \delta_i \tag{6}$$

Definition 7: For the set of n tasks τ, the *maximum density* is denoted by $\delta_{max}(\tau)$ and is defined as follows:

$$\delta_{max}(\tau) = \max\{\delta_i | 1 \leq i \leq n\} \tag{7}$$

B. Decomposition

The high-level idea of our decomposition is as follows.

1) In our decomposition, each thread of the task becomes its own sequential subtask. These individual subtasks are assigned release times and deadlines. Since each thread of a segment is identical (with respect to its execution time), we consider each segment one at a time, and assign the same release times and deadlines to all subtasks generated from threads of the same segment.
2) Since a segment $\langle P_{i,j}, m_{i,j} \rangle$ has to complete before segment $\langle P_{i,j+1}, m_{i,j+1} \rangle$ can start, the release time of the subtasks of segment $\langle P_{i,j+1}, m_{i,j+1} \rangle$ is equal to the absolute deadline of the subtasks of segment $\langle P_{i,j}, m_{i,j} \rangle$.
3) We calculate the slack for each task. For task τ_i, the slack L_i is defined as $L_i = T_i - P_i$, the difference between the period (i.e. deadline) and the critical path length. This slack is then distributed among the segments according to a principle of "equitable density"; that is, we try to keep the density of each segment approximately rather than exactly equal by maintaining a uniform upper bound on the densities. To maintain this equitable density, we take both the number of threads in segments and the computation requirement of each thread into consideration while distributing slack.

In order to take the computation requirement of the threads of segments into consideration, we assign proportional slack fractions instead of absolute slacks. We now formalize the notion of *slack fraction*, $f_{i,j}$, for the j-th segment (i.e. segment $\langle P_{i,j}, m_{i,j} \rangle$) of task τ_i. *Slack fraction* $f_{i,j}$ is the fraction of L_i (i.e. the total slack) to be allotted to segment $\langle P_{i,j}, m_{i,j} \rangle$ proportionally to its minimum computation requirement. Specifically, each thread in segment $\langle P_{i,j}, m_{i,j} \rangle$ is assigned a slack of $f_{i,j} P_{i,j}$. Each thread gets this "extra time" beyond its execution requirement. Thus, for each thread in segment $\langle P_{i,j}, m_{i,j} \rangle$, the relative deadline is assigned as

$$d_{i,j} = P_{i,j} + f_{i,j} P_{i,j} \tag{8}$$

For example, if a segment has $P_{i,j} = 2$ and it is assigned slack of 1.5, then its relative deadline is $2(1+1.5) = 5$. Since a segment cannot start before all previous segments complete, the release offset of a segment $\langle P_{i,j}, m_{i,j} \rangle$ is assigned as

$$\phi_{i,j} = \sum_{k=1}^{j-1} d_{i,k} \tag{9}$$

Thus, the density of each thread in segment $\langle P_{i,j}, m_{i,j} \rangle$ is:

$$\frac{P_{i,j}}{d_{i,j}} = \frac{P_{i,j}}{P_{i,j} + f_{i,j} P_{i,j}} = \frac{1}{1 + f_{i,j}}$$

Since a segment $\langle P_{i,j}, m_{i,j} \rangle$ consists of $m_{i,j}$ threads, the segment's density, $\delta_{i,j}$, is defined as follows:

$$\delta_{i,j} = \frac{m_{i,j}}{1 + f_{i,j}} \tag{10}$$

Note that to meet the deadline of the parallel task, the segment slack should be assigned so that

$$f_{i,1}.P_{i,1} + f_{i,2}.P_{i,2} + f_{i,3}.P_{i,3} + \cdots + f_{i,s_i}.P_{i,s_i} \leq L_i. \tag{11}$$

In our decomposition, we always assign the maximum possible segment slack and, therefore, for our decomposition, the above inequality is in fact an equality.

Since after assigning slacks, we want to keep the density of each segment about equal, we must take the number of threads of the segment into consideration while assigning slack fractions. In our decomposition, we calculate a threshold based on task parameters. The segments whose number of threads is greater than this threshold are assigned slack. The other segments are not assigned any slack, since they are deemed to be less computation intensive. Hence, to calculate segment slack according to equitable density, we classify segments into two categories:

- *Heavy segments* are those which have $m_{i,j} > \frac{C_i}{T_i - P_i}$. That is, they have many parallel threads.
- *Light segments* are those which have $m_{i,j} \leq \frac{C_i}{T_i - P_i}$.

Using these categorization, we also classify parallel tasks into two categories: tasks that have some or all heavy segments versus tasks that have only light segments.

Tasks with some (or all) heavy segments: For the tasks which have some heavy segments, we treat heavy and light segments differently while assigning slack. In particular, we assign no slack to the light segments; that is, segments with $m_{i,j} \leq \frac{C_i}{T_i - P_i}$ are assigned $f_{i,j} = 0$. The total available slack L_i is distributed among the heavy segments (segments with $m_{i,j} > \frac{C_i}{T_i - P_i}$) in a way so that each of these segments has the same density.

For simplicity of presentation, we first distinguish notation between the heavy and light segments. Let the heavy segments (i.e., segment with $m_{i,j} > \frac{C_i}{T_i - P_i}$) be denoted by : $\{\langle P'_{i,1}, m'_{i,1}\rangle, \langle P'_{i,2}, m'_{i,2}\rangle, \cdots, \langle P'_{i,s'_i}, m'_{i,s'_i}\rangle\}$, where $s'_i \leq s_i$. Then, let

$$P'_i = \sum_{j=1}^{s'_i} P'_{i,j} \tag{12}$$

$$C'_i = \sum_{j=1}^{s'_i} m'_{i,j} P'_{i,j} \tag{13}$$

The light segments (i.e., segments with $m_{i,j} \leq \frac{C_i}{T_i - P_i}$) are represented by : $\{\langle P''_{i,1}, m''_{i,1}\rangle, \langle P''_{i,2}, m''_{i,2}\rangle, \cdots, \langle P''_{i,s''_i}, m''_{i,s''_i}\rangle\}$, where $s''_i \leq s_i$. Then, let

$$P''_i = \sum_{j=1}^{s''_i} P''_{i,j} \tag{14}$$

$$C''_i = \sum_{j=1}^{s''_i} m''_{i,j} P''_{i,j} \tag{15}$$

Now, the following relations hold:

$$s_i = s'_i + s''_i \tag{16}$$

$$P_i = P'_i + P''_i \tag{17}$$

$$C_i = C'_i + C''_i \tag{18}$$

Now we calculate $f'_{i,j}$ for all the heavy segments (i.e., segments $\langle P'_{i,j}, m'_{i,j}\rangle$, where $1 \leq j \leq s'_i$ and $m'_{i,j} > \frac{C_i}{T_i - P_i}$) so that they all have equal density. Therefore, we can say that:

$$\frac{m'_{i,1}}{1 + f'_{i,1}} = \frac{m'_{i,2}}{1 + f'_{i,2}} = \frac{m'_{i,3}}{1 + f'_{i,3}} = \cdots = \frac{m'_{i,s'_i}}{1 + f'_{i,s'_i}} \tag{19}$$

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

$$f'_{i,1}.P'_{i,1} + f'_{i,2}.P'_{i,2} + f^3_i.P'_{i,3} + \cdots + f'_{i,s'_i}.P'_{i,s'_i} = L_i \tag{20}$$

It follows that the value of $f'_{i,j}$, for each j, $1 \leq j \leq s'_i$, can be determined by solving Equations 19 and 20 as shown below.

From Equation 19 , each $f'_{i,j}$, $2 \leq j \leq s'_i$, can be expressed in terms of $f'_{i,1}$ as follows

$$f'_{i,j} = (1 + f'_{i,1})\frac{m'_{i,j}}{m'_{i,1}} - 1 \tag{21}$$

Putting the above value of each $f'_{i,j}$, $2 \leq j \leq s'_i$, into Equation 20 gives

$$f'_{i,1}P'_{i,1} + \sum_{j=2}^{s'_i}\left(\Big((1 + f'_{i,1})\frac{m'_{i,j}}{m'_{i,1}} - 1\Big)P'_{i,j}\right) = L_i$$

$$\iff f'_{i,1}P'_{i,1} + \sum_{j=2}^{s'_i}\left(\frac{m'_{i,j}}{m'_{i,1}}P'_{i,j} + f'_{i,1}\frac{m'_{i,j}}{m'_{i,1}}P'_{i,j} - P'_{i,j}\right) = L_i$$

$$\iff f'_{i,1}P'_{i,1} + \frac{1}{m'_{i,1}}\sum_{j=2}^{s'_i} m'_{i,j}P'_{i,j} + \frac{f'_{i,1}}{m'_{i,1}}\sum_{j=2}^{s'_i} m'_{i,j}P'_{i,j} - \sum_{j=2}^{s'_i} P'_{i,j} = L_i$$

$$\iff f'_{i,1}\big(P'_{i,1} + \frac{1}{m'_{i,1}}\sum_{j=2}^{s'_i} m'_{i,j}P'_{i,j}\big) = L_i + \sum_{j=2}^{s'_i} P'_{i,j} - \frac{1}{m'_{i,1}}\sum_{j=2}^{s'_i} m'_{i,j}P'_{i,j}$$

$$\iff f'_{i,1} = \frac{L_i + \sum_{j=2}^{s'_i} P'_{i,j} - \frac{1}{m'_{i,1}}\sum_{j=2}^{s'_i} m'_{i,j}P'_{i,j}}{P'_{i,1} + \frac{1}{m'_{i,1}}\sum_{j=2}^{s'_i} m'_{i,j}P'_{i,j}}$$

$$\iff f'_{i,1} = \frac{L_i + (\sum_{j=2}^{s'_i} P'_{i,j} + P'_{i,1}) - (P'_{i,1} + \frac{1}{m'_{i,1}}\sum_{j=2}^{s'_i} m'_{i,j}P'_{i,j})}{P'_{i,1} + \frac{1}{m'_{i,1}}\sum_{j=2}^{s'_i} m'_{i,j}P'_{i,j}}$$

$$\iff f'_{i,1} = \frac{L_i + \sum_{j=1}^{s'_i} P'_{i,j}}{P'_{i,1} + \frac{1}{m'_{i,1}}\sum_{j=2}^{s'_i} m'_{i,j}P'_{i,j}} - 1$$

$$\iff f'_{i,1} = \frac{L_i + P'_i}{P'_{i,1} + \frac{1}{m'_{i,1}}\sum_{j=2}^{s'_i} m'_{i,j}P'_{i,j}} - 1 \qquad \text{(From 12)}$$

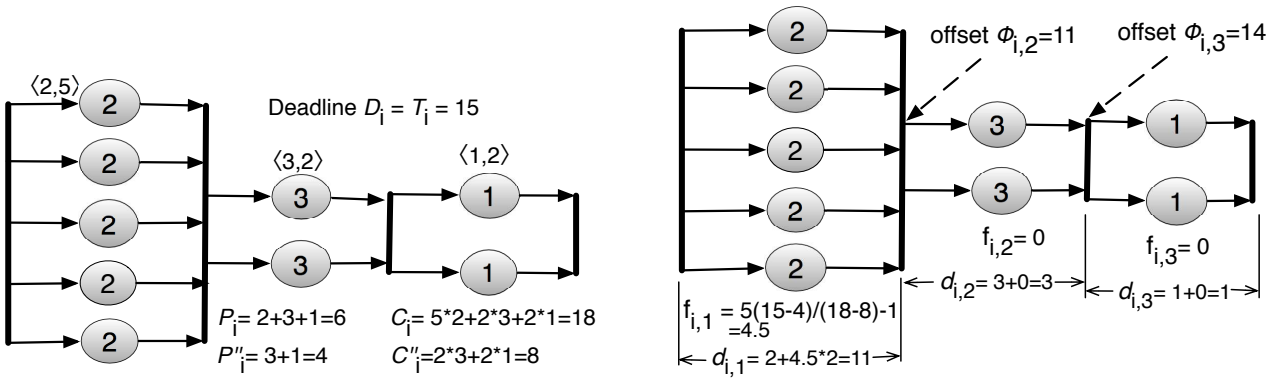
$$\iff f'_{i,1} = \frac{m'_{i,1}(L_i + P'_i)}{m'_{i,1}P'_{i,1} + \sum_{j=2}^{s'_i} m'_{i,j}P'_{i,j}} - 1$$

$$\iff f'_{i,1} = \frac{m'_{i,1}(L_i + P'_i)}{\sum_{j=1}^{s'_i} m'_{i,j}P'_{i,j}} - 1$$

$$\iff f'_{i,1} = \frac{m'_{i,1}(L_i + P'_i)}{C'_i} - 1 \qquad \text{(From 13)}$$

$$\iff f'_{i,1} = \frac{m'_{i,1}(L_i + P'_i)}{C_i - C''_i} - 1 \qquad \text{(From 18)}$$

$$\iff f'_{i,1} = \frac{m'_{i,1}((T_i - P_i) + P'_i)}{C_i - C''_i} - 1$$



(a) A parallel synchronous task τ_i

(b) Decomposed task τ_i^{decom}

Fig. 2. An example of decomposition

$$\iff f'_{i,1} = \frac{m'_{i,1}(T_i - (P'_i + P''_i) + P'_i)}{C_i - C''_i} - 1 \quad \text{(From 17)}$$
$$\iff f'_{i,1} = \frac{m'_{i,1}(T_i - P''_i)}{C_i - C''_i} - 1$$

Now putting the above value of $f'_{i,1}$ in Equation 19, for any segment $\langle P'_{i,j}, m'_{i,j}\rangle$:

$$f'_{i,j} = \frac{m'_{i,j}(T_i - P''_i)}{C_i - C''_i} - 1 \tag{22}$$

Intuitively, the slack never should be negative, since the deadline should be no less than the computation requirement of the thread. Since $m'_{i,j} > \frac{C_i}{T_i - P_i}$, according to Equation 22, the quantity $\frac{m'_{i,1}(T_i - P''_i)}{C_i - C''_i} \geq 1$. This implies that $f'_{i,j} \geq 0$. As mentioned above, all light segments have $f_{i,j} = 0$. Figure 2 shows a simple example of decomposition for a task τ_i consisting of 3 segments.

Now, using Equation 10, the density of every segment $\langle P'_{i,j}, m'_{i,j}\rangle$ is :

$$\frac{m'_{i,j}}{1 + f'_{i,j}} = \frac{m'_{i,j}}{1 + \frac{m'_{i,j}(T_i - P''_i)}{C_i - C''_i} - 1} = \frac{C_i - C''_i}{T_i - P''_i} \tag{23}$$

In addition, the light segments have density $m_{i,j} \leq C_i/(T_i - P_i)$, since they have no slack.

Tasks with no heavy segments: When the parallel task does not contain any heavy segments, we just assign the slack proportionally (according to length of $P_{i,j}$) among all segments. That is,

$$f_{i,j} = \frac{L_i}{P_i} \tag{24}$$

Now, using Equation 10, the density of each segment $\langle P_{i,j}, m_{i,j}\rangle$ is:

$$\frac{m_{i,j}}{1 + f_{i,j}} = \frac{m_{i,j}}{1 + \frac{L_i}{P_i}} = m_{i,j}\frac{P_i}{L_i + P_i} = m_{i,j}\frac{P_i}{T_i} \tag{25}$$

C. Density Analysis

Once the above decomposition is done on task τ_i : $(\langle P_{i,1}, m_{i,1}\rangle, \langle P_{i,2}, m_{i,2}\rangle, \cdots, \langle P_{i,s_i}, m_{i,s_i}\rangle)$, each thread of each segment $\langle P_{i,j}, m_{i,j}\rangle$, $1 \leq j \leq s_i$, is considered as a sequential multiprocessor task. Since $f_{i,j} \geq 0,\ \forall 1 \leq j \leq s_i$, $\forall 1 \leq i \leq n$, the maximum density δ_{max} of any thread is:

$$\delta_{max} = \frac{1}{1 + f_{i,j}} \leq 1 \tag{26}$$

Lemma 1 shows that the density of every segment is at most $\frac{C_i}{T_i - P_i}$ for both tasks with and without heavy segments.

Lemma 1: After the decomposition, the density $\delta_{i,j}$ of every segment $\langle P_{i,j}, m_{i,j}\rangle$, where $1 \leq j \leq s_i$, of every task τ_i is upper bounded by $\frac{C_i}{T_i - P_i}$.

Proof: First, we analyze the case when the task contains some heavy segments, i.e., there is some segment $\langle P_{i,j}, m_{i,j}\rangle$ such that $m_{i,j} > \frac{C_i}{T_i - P_i}$. According to Equation 23, for every heavy segment,

$$\begin{aligned}
\delta_{i,j} &= \frac{C_i - C''_i}{T_i - P''_i} \\
&\leq \frac{C_i}{T_i - P''_i} && \text{(since } C''_i \geq 0\text{)} \\
&\leq \frac{C_i}{T_i - P_i} && \text{(since } P_i \geq P''_i\text{)}
\end{aligned}$$

For every light segment $\langle P_{i,j}, m_{i,j}\rangle$ with $m_{i,j} \leq \frac{C_i}{T_i - P_i}$, $f_{i,j} = 0$. That is, its deadline is equal to its computation requirement $P_{i,j}$. Therefore, the density of each of such segments is:

$$\delta_{i,j} = \frac{m_{i,j}}{1 + f_{i,j}} = m_{i,j} \leq \frac{C_i}{T_i - P_i} \tag{27}$$

Now we analyze the case when there are no heavy segments in the task. That is, for every segment $\langle P_{i,j}, m_{i,j}\rangle$ of τ_i, $m_{i,j} \leq \frac{C_i}{T_i - P_i}$. From Equation 25, the density of each segment $\langle P_{i,j}, m_{i,j}\rangle$ of τ_i is:

$$\begin{aligned}
\delta_{i,j} &= m_{i,j}\frac{P_i}{T_i} \\
&\leq m_{i,j} && \text{(since } T_i \geq P_i, \text{ by Observation 1)} \\
&\leq \frac{C_i}{T_i - P_i}
\end{aligned}$$

Hence, follows the lemma. ■

Therefore, our decomposition distributes the slack so that each segment has a density that is bounded above. We use

τ_i^{decom} to denote task τ_i after decomposition. That is, τ_i^{decom} denotes the set of subtasks generated from τ_i through decomposition. Similarly, we use τ^{decom} to denote the entire task set τ after decomposition. That is, τ^{decom} is the set of all subtasks that our decomposition generates. Theorem 2 establishes an upper bound on the density of every synchronous task after decomposition.

Theorem 2: The density δ_i of every τ_i^{decom}, $1 \leq i \leq n$, (i.e. the density of every task τ_i after decomposition) is upper bounded by $\frac{C_i}{T_i - P_i}$.

Proof: After the decomposition, the densities of all segments of τ_i comprise the density of τ_i^{decom}. However, no two segments are simultaneous active, and each segment occurs exactly once during the activation time of task τ_i. Therefore, we can replace each segment with the segment that has the maximum density. Thus, task τ_i^{decom} can be considered as s_i occurances of the segment that has the maximum density, and therefore, the density of the entire task set τ_i^{decom} is equal to the density of the maximum density segment, which is at most $\frac{C_i}{T_i - P_i}$ (Lemma 1). Therefore, $\delta_i \leq \frac{C_i}{T_i - P_i}$. ■

Lemma 3: If τ^{decom} is schedulable, then τ is also schedulable.

Proof: For each τ_i^{decom}, $1 \leq i \leq n$, its deadline (i.e. period) and execution requirement are the same as those of original task τ_i. Besides, in each τ_i^{decom}, a subtask is released only after all its preceding segments are complete. Hence, the precedence relations in original task τ_i are retained in τ_i^{decom}. Therefore, If τ^{decom} is schedulable under an algorithm, then τ is also schedulable under that algorithm. ■

IV. Global EDF Scheduling

After our proposed decomposition, we consider the scheduling of synchronous parallel tasks. Lakshmanan et al. [?] show that there exist task sets with total utilization slightly greater than (and arbitrarily close to 1) that are unschedulable with q processor cores. Since our model is a generalization of theirs, this lower bound still holds for our tasks. Since conventional utilization bound approaches are not useful for schedulability analysis of parallel tasks, we (like [?]) use the *resource augmentation bound approach*, originally introduced in [**?**]. We first consider *global scheduling* where tasks are allowed to migrate among processor cores. We then analyze schedulability in terms of a resource augmentation bound. Since the synchronous parallel tasks are now split into individual sequential subtasks, we can use global EDF (Earliest Deadline First) scheduling for them. The EDF policy for subtask scheduling is basically the same as the traditional global EDF where jobs with earlier deadlines are assigned higher priorities.

Under global EDF scheduling, we now present a schedulability analysis in terms of a resource augmentation bound for our decomposed tasks. For any task set, the *resource augmentation bound* ν of a scheduling policy $\mathbb{A}$ on a multicore processor with q cores represents a processor speedup factor. That is, if there exists any (optimal) algorithm under which a task set is feasible on q identical unit speed processor cores, then $\mathbb{A}$ is guaranteed to successfully schedule this task set on a q-core processor, where each processor core is ν times as fast as the original. In Theorem 5, we show that, after our proposed decomposition, global EDF scheduling has a resource augmentation bound of 2.62.

Our analysis uses a result for constrained-deadline sporadic sequential tasks on q processors [?], stated in Theorem 4. This result is a generalization of the result for implicit deadline sporadic tasks [**?**].

Theorem 4: (From [?]) Any constrained-deadline sporadic task set with total density δ_{sum} and maximum density δ_{max} is schedulable using global EDF strategy on q unit-speed processor cores if

$$\delta_{sum} \leq q - (q-1)\delta_{max} \tag{28}$$

Since we converted our synchronous parallel tasks into sequential tasks with constrained deadlines, this result applies to our transformed task set τ^{decom}. If we can schedule τ^{decom}, then we can schedule τ (Lemma 3).

Theorem 5: If a synchronous parallel task set τ is schedulable on q unit speed processor cores using any (optimal) algorithm, then the transformed task set τ^{decom} is schedulable using global EDF on q processor cores of speed 2.62.

Proof: When each processor core is ν times faster,

$$\text{The maximum execution time of task } \tau_i\text{: } C_i^\nu = \frac{C_i}{\nu} \tag{29}$$

$$\text{The minimum execution time of task } \tau_i\text{: } P_i^\nu = \frac{P_i}{\nu} \tag{30}$$

Let the total density and the maximum density of task set τ^{decom} be denoted by δ_{sum}^ν and δ_{max}^ν, respectively, when each processor core is ν times faster. We have

$$\delta_{max}^\nu = \frac{\delta_{max}}{\nu} \leq \frac{1}{\nu} \qquad \text{(From 26)} \tag{31}$$

The value δ_{sum}^ν turns out to be the total density of all decomposed tasks. By Theorem 2 and Equations 29 and 30, the density of every task τ_i^{decom} on q identical processors each of which is ν times faster is:

$$\begin{aligned}
\delta_i^\nu \leq \frac{C_i^\nu}{T_i - P_i^\nu} &= \frac{\frac{C_i}{\nu}}{T_i - \frac{P_i}{\nu}} \\
&\leq \frac{\frac{C_i}{\nu}}{T_i - \frac{T_i}{\nu}} \qquad \text{(since } P_i \leq T_i\text{)} \\
&= \frac{\frac{C_i}{\nu}}{T_i(1 - \frac{1}{\nu})} \\
&= \frac{1}{\nu - 1}.\frac{C_i}{T_i}
\end{aligned}$$

Thus,

$$\delta_{sum}^\nu = \sum_{i=1}^{n} \delta_i^\nu \leq \frac{1}{\nu - 1} \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{32}$$

Now, let there exists some algorithm $\mathbb{A}$ under which the original task set τ is feasible on q identical unit-speed processor cores. For τ to be schedulable under $\mathbb{A}$, the following

condition must hold (Observation 2).

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq q \tag{33}$$

Therefore, we have

$$\delta_{sum}^{\nu} \leq \frac{q}{\nu - 1} \tag{34}$$

Note that, in the decomposed task set, every thread of the original task is a sequential multiprocessor task. Therefore, δ_{sum}^{ν} is the total density of all threads (decomposed to multiprocessor tasks), and δ_{max}^{ν} is the maximum density among all threads. Therefore, by Theorem 4, following is the sufficient condition for EDF schedulability of the decomposed task set on q ν speed processors:

$$\delta_{sum}^{\nu} \leq q - (q-1)\delta_{max}^{\nu} \tag{35}$$

We can substitute the values of δ_{sum}^{ν} (Equation 34) and δ_{max}^{ν} (Equation 31) into this equation, and say that the task set is schedulable if

$$\begin{aligned} & \frac{q}{\nu - 1} \leq q - (q-1)\frac{1}{\nu} \\ \equiv \; & \frac{1}{\nu - 1} + \frac{1}{\nu} - \frac{1}{q\nu} \leq 1 \end{aligned}$$

From the above Equation, the task set must be schedulable if

$$\begin{aligned} & \frac{1}{\nu - 1} + \frac{1}{\nu} \leq 1 \\ \equiv \; & \nu^2 - 3\nu + 1 \geq 0 \end{aligned}$$

Therefore, the task set is schedulable if $\nu \geq \frac{3+\sqrt{5}}{2} \approx 2.62$. ■

V. Partitioned Deadline Monotonic Scheduling

Using the same decomposition described in Section III, we now derive the resouce augmentation bound required to schedule task sets under FBB-FDD partitioned Deadline Monotonic (DM) scheduling [**?**] which the previous work on parallel task scheduling [?] uses as its underlying scheduling strategy. However, as was explained earlier, we consider a more general task model and a different task decomposition strategy, and as we explain in this section still are able to obtain the same resource augmentation bound of 3.42.

In *partitioned scheduling*, tasks are executed only on their assigned processor cores, and are not allowed to migrate between processor cores. We schedule the decomposed subtasks using FBB-FFD considering each subtask as a traditional multiprocessor task. Specifically, the decomposed subtasks are considered for processor core allocation in decreasing order of deadline monotonic priorities, and each subtask is allocated to the first available core that satisfies the condition specified in FBB-FFD algorithm for allocation.

We use an analysis similar to the one used in [?] to derive the resource augmentation bound as shown in Theorem 6. The analysis is based on the demand bound function of the tasks after decomposition. We first define the demand bound function and load as follows.

Definition 8: The *demand bound function* (DBF), originally introduced in [**?**], of a task τ_i is the largest cumulative execution requirement of all jobs generated by τ_i that have both their arrival times and their deadlines within a contiguous interval of t time units. For a task τ_i with a maximum computation requirement of C_i, period of T_i, and a deadline of D_i, its DBF is given by:

$$DBF(\tau_i, t) = \max\left(0, \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) C_i\right) \tag{36}$$

For implicit deadline task τ_i:

$$DBF(\tau_i, t) = \max\left(0, \left\lfloor \frac{t}{T_i} \right\rfloor C_i\right) \tag{37}$$

Definition 9: Based upon the DBF function, the *load* of task system τ, denoted by $\lambda(\tau)$, is defined as follows:

$$\lambda(\tau) = \max_{t>0} \left(\frac{\sum_{i=1}^{n} DBF(\tau_i, t)}{t} \right) \tag{38}$$

Since, for every τ_i^{decom}, $1 \leq i \leq n$, its total execution requirement, period, and deadline are the same as that of the original task τ_i, the demand bound function $DBF(\tau_i^{decom}, t)$ of every τ_i^{decom} over any time interval of t units remains unchanged through decomposition (Lemma 6 in [?]). That is,

$$DBF(\tau_i^{decom}, t) = DBF(\tau_i, t), \quad \forall\ 1 \leq i \leq n$$

The constrained-deadline subtasks in τ^{decom} are offset to ensure that subtasks belonging to different parallel execution segments of the same task τ_i are never simultaneously active. For every task τ_i, the release offset of each subtask corresponding to segment $\langle P_{i,j}, m_{i,j} \rangle$ is:

$$\phi_{i,j} = \sum_{k=1}^{j-1} d_{i,k} \tag{39}$$

This implies that each segment (and each of its subtasks) happens only once during the activation time of task τ_i, and no two segments are simultaneously active. That is, the total number of occurrences of all of these segments during the activation time of task τ_i is s_i (the number of total segments in τ_i). Therefore, to calculate an upper bound on the total density of all subtasks of τ_i^{decom}, we can consider the segment having the maximum density in place of every segment.

According to Lemma 1, after decomposition the density of every segment of task τ_i is upper-bounded by $\frac{C_i}{T_i - P_i}$. Therefore, after decomposition of τ_i, its demand bound function over any interval of length t does not exceed $\frac{C_i}{T_i - P_i} t$ due to the offsets (see [?] for details). That is,

$$DBF(\tau_i^{decom}, t) \leq \frac{C_i}{T_i - P_i} . t$$

and the load of the task system τ, denoted by $\lambda(\tau)$ is:

$$\lambda(\tau^{decom}) = \max_{t>0} \left(\frac{\sum_{i=1}^{n} DBF(\tau_i^{decom}, t)}{t} \right) \leq \sum_{i=1}^{n} \frac{C_i}{T_i - P_i} \tag{40}$$

Theorem 6: If a synchronous parallel task set τ is schedulable using the optimal scheduling strategy on q unit speed processors, then its decomposed task set τ^{decom} is schedulable using FBB-FDD on q identical processors of speed 3.42.

Proof: According to [?], any constrained sporadic task set with total utilization u_{sum}, maximum density δ_{max}, and load λ is schedulable by FBB-FFD on q unit-capacity processors if

$$q \geq \frac{\lambda + u_{sum} - \delta_{max}}{1 - \delta_{max}} \tag{41}$$

Let the maximum density, total utilization, and load of task set τ^{decom} be denoted by δ_{max}^{ν}, u_{sum}^{ν}, and λ^{ν} respectively, when each processor core is ν times faster. From Equation 29:

$$u_{sum}^{\nu} = \sum_{i=1}^{n} \frac{C_i^{\nu}}{T_i} = \frac{1}{\nu} \sum_{i=1}^{n} \frac{C_i}{T_i} = \frac{u_{sum}}{\nu} \tag{42}$$

From Equations 29, 30, and 40,

$$\lambda^{\nu} \leq \sum_{i=1}^{n} \frac{C_i^{\nu}}{T_i - P_i^{\nu}} \leq \sum_{i=1}^{n} \frac{1}{\nu - 1} . \frac{C_i}{T_i} = \frac{u_{sum}}{\nu - 1} \tag{43}$$

Since the task set τ^{decom} contains sequential tasks, due to Equation 41, the following is the sufficient condition for FBB-FFD schedulability of the decomposed task set on q identical processors each of which is ν times faster:

$$q \geq \frac{\lambda^{\nu} + u_{sum}^{\nu} - \delta_{max}^{\nu}}{1 - \delta_{max}^{\nu}} \tag{44}$$

Using, Equations 43, 42, 31 in Equation 44, the task set is schedulable if

$$q \geq \frac{\frac{u_{sum}}{\nu - 1} + \frac{u_{sum}}{\nu} - \frac{1}{\nu}}{1 - \frac{1}{\nu}} \tag{45}$$

Even for the best possible algorithm, u_{sum} can be at most q for unit speed processors. Therefore, the task set is schedulable if

$$\begin{aligned} & q \geq \frac{\frac{q}{\nu - 1} + \frac{q}{\nu} - \frac{1}{\nu}}{1 - \frac{1}{\nu}} \\ \equiv & \ \nu - \frac{1}{\nu - 1} \geq 3 - \frac{1}{q} \\ \equiv & \ \nu \geq 2 + \sqrt{2} \approx 3.42 \end{aligned}$$

■

VI. Generalizing to a DAG Task Model

In the analysis presented so far, we assume that we have synchronous parallel tasks. That is, there is a synchronization point at the end of each segment, and the next segment starts only after all the threads of the previous segment have completed. In this section, we argue that even more general parallel tasks that can be represented as directed acyclic graphs (DAGs) with unit time nodes, can be easily converted into synchronous tasks. Therefore, the above analysis holds for these tasks as well.

In the DAG model of tasks, each job is made up of nodes that represent work, and edges that represent dependences between nodes. Therefore, a node can execute only after all of its predecessors have been executed. We consider the case where each node represents unit-time work.

Given this limitation of unit-time nodes, a DAG can be converted into a synchronous task by simply adding new dependence edges. The conversion can be done as follows: If there is an edge from node u to node v, then we say that u is the parent of v. Then we calculate the depth of each node v, $h(v)$. If v has no parents, then it is assigned depth 1. Otherwise, we calculate the depth of v as follows:

$$h(v) = \max_{u \text{ parent of } v} h(u) + 1 \tag{46}$$

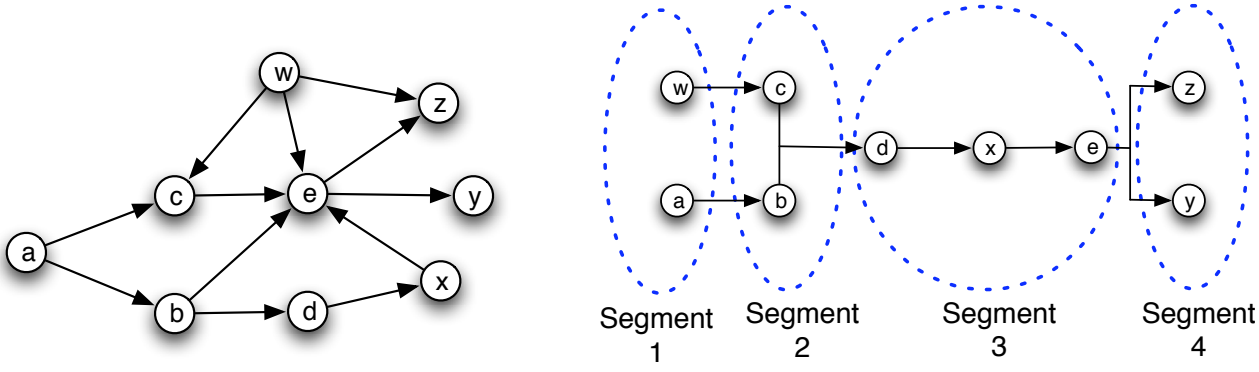
Then each node with depth t is assigned to segment t. After this, every node of the DAG is now considered as a parallel thread in the corresponding segment. The threads (i.e. nodes in the DAG) in the segment can happen in parallel, and the segment is considered as a parallel segment of a bulk-synchronous task. If there are $k > 1$ consecutive segments such that each segment consists of just one thread, then all these k segments are considered as one sequential segment of execution requirement k (by preserving the sequence). Figure 3 shows a simple example, where a DAG in Figure 3(a) is represented as a synchronous task in Figure 3(b). This transformation is valid since it preserves all the dependences of the DAG, and in fact only adds extra dependences.

Upon representing a DAG-task as a sunchronous task, we perform the same decomposition proposed in Section III, and then this decomposed task set can be scheduled using either global EDF or partitioned DM scheduling. Note that the transformation from the DAG to the synchronous task preserves the work C_i of the task i. Since the lower bound we use for our analysis is $\sum C_i/T_i < q$, this lower bound still holds. In addition, the transformation also preserves the critical path length P_i of the task, and therefore the rest of the analysis also holds. Therefore, our analysis proves that a set of DAG-tasks can be scheduled with resource augmentation of 2.62 under global EDF scheduling, and of 3.42 under partitioned DM scheduling.

VII. Related Work

There has been extensive work on traditional multiprocessor real-time scheduling [**?**], [?], [?], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [?], [**?**] (see also [**?**] for a survey of results from 1960 to 2009). Most of this work focuses on scheduling sequential tasks on multiprocessor or multi-core systems. In the parallel computing community, there has also been extensive work on scheduling of one or more parallel jobs on multiprocessors or multi-cores [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**]. However, this work does not consider task deadlines.

There has been relatively little work that considers parallel tasks with real-time constraints. Preemptive fixed-priority scheduling of parallel task systems was shown to be NP-hard by Han et al. [**?**]. Kwon et al. [**?**] explored preemptive



(a) DAG

(b) Parallel synchronous model

Fig. 3. DAG to parallel synchronous model

EDF scheduling of parallel task systems with linear-speedup parallelism. Wang et al. [**?**] consider a heuristic nonpreemptive scheduling. However, this work just concentrates on metrics like throughput (total works that meet deadline) [?] or makespan [?], and considers a task model where each task is executed on up to a given number of processors. Following is a brief review of recent results on real-time scheduling of parallel tasks.

Our work is most related to, and is inspired by, recent work by Lakshmanan et al. on real-time scheduling for a more constrained parallel task model using partitioned DM scheduling [?]. They consider tasks that are an alternate sequence of parallel and sequential tasks, and each parallel segment has the same number of threads. They also convert each parallel task into a set of sequential tasks, and then derive a resource augmentation bound of 3.42 for partitioned deadline monotonic scheduling. However, their strategy of task decompostion is quite different from ours. They create a master thread of execution requirement equal to period T_i (i.e., the deadline) by stretching task τ_i when $C_i > T_i$. In partitioned DM scheduling, one processor core is assigned exclusively to the master thread. Other threads some of whose execution requirements have been added to the master thread, remain as constrained deadline subtasks, and are scheduled using partitioned DM policy without modification. Their transformation and analysis does not apply to the more general task model where the number of threads can vary by segment, and can exceed the number of cores in the machine. In addition, any task that can be expressed as a DAG of unit time nodes can be converted into our general synchronous task model in a work and critical-path length conserving manner. In contrast, if we convert a DAG into their more restricted synchronous task model, we may have to increase the work or the critical-path length. Therefore, unlike our analysis, their analysis does not directly apply to these more general DAG tasks.

Most of the other work on real time scheduling of parallel tasks address more simplistic task models. Jansen [**?**], Lee et al. [**?**], and Collette et al. [**?**] studied the scheduling of *malleable tasks*, where each task is assumed to execute on a fixed number of cores or processors that are dedicated to the task. For *moldable tasks*, where the number of processors simultaneously used by the task is not fixed a $priori$ but is determined before execution, Manimaran et al. [**?**] addressed non-preemptive EDF scheduling. Kato et al. [**?**] studied the Gang EDF scheduling of moldable parallel task systems. They require the users to select (at the submission time) the number of processors upon which a parallel task will run. They further assume that a parallel task generates the same number of threads as processors selected before the execution. Notably, both the malleable and moldable task models addressed in the aforementioned work assume the number of processors occupied by a task to be fixed throughout its execution. In contrast, the parallel task model addressed in this paper allows tasks to have different numbers of threads in different stages, which make our solution applicable to a much broader range of applications.

VIII. Conclusions

With the advent of the era of multi-core computing, real-time scheduling of parallel tasks is crucial for real-time applications to exploit the power of multi-core processors. While recent research on real-time scheduling of parallel tasks has shown promise, the efficacy of existing approaches is limited by their restrictive parallel task models. To overcome these limitations, this paper presents new results on real-time scheduling for generalized parallel task models. We first consider a general synchronous parallel task model where the number of parallel threads may vary over time and exceed the number of processor cores. We then propose a novel task decomposition algorithm that converts parallel tasks with implicit deadlines into a set of sequential tasks with constrained deadlines. We derive a resource augmentation bound of 2.62 under global EDF scheduling, which to our knowledge is the first resource augmentation bound for global EDF scheduling of parallel tasks. Furthermore, we derive a resource augmentation bound of 3.42 for partitioned DM scheduling. While the same as the previous bound in the literature [4] for partitioned DM scheduling, our algorithm and analysis are applied to a more general synchronous task model. Finally, we

show how to convert a task in the form of a Directed Acyclic Graph (DAG) with unit time nodes into a synchronous task. Therefore, our analysis holds for this more general task model as well.

In the future, we plan to consider even more general DAG models, where each node may have more than one unit of work, and to provide analysis without requiring any transformation to a synchronous task model. We also plan to extend our results to even more general task models and system issues such as cache effects, preemption penalties, and resource contention.