2011 32nd IEEE Real-Time Systems Symposium

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 computation-intensive real-time applications with 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 intra-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, each segment having an arbitrary number of parallel threads that synchronize at the end of the segment. We propose a new task decomposition method that decomposes each parallel task into a set of sequential tasks. We prove that our task decomposition achieves a resource augmentation bound of 4 and 5 when the decomposed tasks are scheduled using global EDF and partitioned deadline monotonic scheduling, respectively. Finally, we extend our analysis to directed acyclic graph (DAG) task model where each node in the DAG has unit execution requirement. 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.**

***Keywords*-parallel task; multi-core processor; real-time scheduling; resource augmentation bound.**

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 [1] with a view to making it generally available, and ClearSpeed has developed a 96-core processor [2]. 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 [1].

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 [3] where the performance limitations of traditional single-core processors have been a major hurdle.

The growing importance of parallel task models for 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 [4] on parallel scheduling for real-time tasks analyzes the *resource augmentation bound* using partitioned Deadline Monotonic (DM) scheduling. It considers a 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 segments have an *equal* number of parallel threads, and that number must be *no greater* than the total number of processor cores.

While the work presented in [4] 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, its analyzes the resource augmentation bound under partitioned DM scheduling only, and does not consider other scheduling policies such as global EDF. To overcome these limitations, we consider a more general synchronous task model in this paper. Our tasks still contain segments where the threads of each segment synchronize at its end. However, in contrast to the restrictive task model addressed in [4], 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. 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:

1052-8725/11 $26.00 © 2011 IEEE
DOI 10.1109/RTSS.2011.27

- For the general synchronous task model, we propose a task decomposition algorithm that converts each implicit deadline parallel task into a set of constrained deadline sequential tasks.
- We derive a resource augmentation bound of 4 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 5 for our more general task model under partitioned DM scheduling.
- 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 related work. Finally, we conclude in Section VIII.

II. Parallel Synchronous Task Model

We primarily consider a synchronous task model, where each parallel job consists of many computation 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 [5] and Intel's CilkPlus [6].

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 $\langle e_{i,j}, m_{i,j} \rangle$, with $m_{i,j}$ being the number of threads in the segment, and $e_{i,j}$ being the 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): $\tau_i : (\langle e_{i,1}, m_{i,1} \rangle, \langle e_{i,2}, m_{i,2} \rangle, \cdots, \langle e_{i,s_i}, m_{i,s_i} \rangle)$ where

- s_i is the total number of segments in task τ_i.
- In a segment $\langle e_{i,j}, m_{i,j} \rangle$, where $1 \leq j \leq s_i$, $e_{i,j}$ is the worst case execution requirement of each thread, and $m_{i,j}$ is the number of threads. Therefore, any segment $\langle e_{i,j}, m_{i,j} \rangle$ with $m_{i,j} > 1$ is a *parallel segment* with a total of $m_{i,j}$ parallel threads, and any segment $\langle e_{i,j}, m_{i,j} \rangle$ 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 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 is later resumed with no cost or penalty. The total number of processor cores is denoted by m.

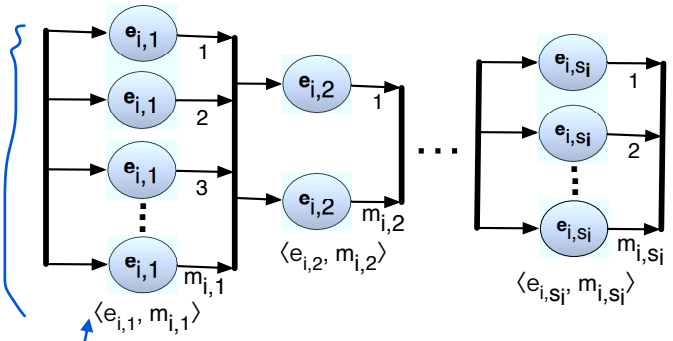


Figure 1. A parallel synchronous task τ_i

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 decomposes each 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 allows us to use existing schedulability analysis for multiprocessor scheduling (both global and partitioned) to prove the resource augmentation bounds for parallel tasks (to be discussed in Sections IV and V). Here, we first present some useful terminology. We then present our decomposition and a density analysis for it.

A. Terminology

Definition 1. *The minimum execution time (i.e. the critical path length) P_i of task τ_i on a multi-core platform where each processor core has unit speed is defined as follows:*

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

Observation 1. *On a unit-speed multi-core platform, any task τ_i requires at least P_i units of time even when the number of cores m is infinite.*

On a multi-core platform where each processor core has speed ν, the critical path length of task τ_i is denoted by $P_{i,\nu}$ and is expressed as follows:

$$P_{i,\nu} = \frac{1}{\nu} \sum_{j=1}^{s_i} e_{i,j} = \frac{P_i}{\nu}$$

Definition 2. *The maximum execution time (i.e. the work) C_i of task τ_i on a multi-core platform where each processor core has unit speed is defined as follows:*

$$C_i = \sum_{j=1}^{s_i} m_{i,j}.e_{i,j}$$

That is, C_i is the execution time of τ_i on a unit-speed single core processor if it is never preempted. On a multi-core platform where each processor core has speed ν, the maximum execution time of task τ_i is denoted by $C_{i,\nu}$ and is expressed as follows:

$$C_{i,\nu} = \frac{1}{\nu}\sum_{j=1}^{s_i} m_{i,j}.e_{i,j} = \frac{C_i}{\nu} \tag{1}$$

Definition 3. *The utilization u_i of task τ_i, and the total utilization $u_{\text{sum}}(\tau)$ for the set of n tasks τ on a unit-speed multi-core platform are defined as follows:*

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

Observation 2. *If the total utilization u_{sum} is greater than m, then no algorithm can schedule τ on m identical unit speed processor cores.*

Definition 4. *The density δ_i of task τ_i, the maximum density $\delta_{\max}(\tau)$ and the total density $\delta_{\text{sum}}(\tau)$ of the set of n tasks τ on a unit-speed multi-core platform are defined as follows:*

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

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

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 $\langle e_{i,j}, m_{i,j}\rangle$ has to complete before segment $\langle e_{i,j+1}, m_{i,j+1}\rangle$ can start, the release time of the subtasks of segment $\langle e_{i,j+1}, m_{i,j+1}\rangle$ is equal to the absolute deadline of the subtasks of segment $\langle e_{i,j}, m_{i,j}\rangle$.
3) 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 i.e.

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

This slack is distributed among the segments according to a principle of "equitable density" meaning that we try to keep the density of each segment approximately rather than exactly equal by maintaining a uniform upper bound on the densities. To do this, we take both the number of threads in each segment and the computation requirement of the threads in each segment into consideration while distributing the slack.

In order to take the computation requirement of the threads in each segment into consideration, we assign proportional slack fractions instead of absolute slack. We now formalize the notion of *slack fraction*, $f_{i,j}$, for the j-th segment (i.e. segment $\langle e_{i,j}, m_{i,j}\rangle$) 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 $\langle e_{i,j}, m_{i,j}\rangle$ proportionally to its minimum computation requirement. Each thread in segment $\langle e_{i,j}, m_{i,j}\rangle$ has a minimum execution time of $\frac{e_{i,j}}{2}$ on 2-speed processor cores, and is assigned a slack value of $f_{i,j}\frac{e_{i,j}}{2}$. Each thread gets this "extra time" beyond its execution requirement on 2-speed processor cores. Thus, for each thread in segment $\langle e_{i,j}, m_{i,j}\rangle$, the relative deadline is assigned as

$$d_{i,j} = \frac{e_{i,j}}{2} + f_{i,j}.\frac{e_{i,j}}{2} = \frac{e_{i,j}}{2}(1 + f_{i,j})$$

For example, if a segment has $e_{i,j} = 4$ and it is assigned a slack fraction 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 $\langle e_{i,j}, m_{i,j}\rangle$ is assigned as

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

Thus, the density of each thread in segment $\langle e_{i,j}, m_{i,j}\rangle$ on 2-speed processor cores is:

$$\frac{\frac{e_{i,j}}{2}}{d_{i,j}} = \frac{\frac{e_{i,j}}{2}}{\frac{e_{i,j}}{2}(1 + f_{i,j})} = \frac{1}{1 + f_{i,j}}$$

Since a segment $\langle e_{i,j}, m_{i,j}\rangle$ consists of $m_{i,j}$ threads, the segment's density on 2-speed processor cores is as follows:

$$\frac{m_{i,j}}{1 + f_{i,j}} \tag{3}$$

Note that to meet the deadline of the parallel task on 2-speed processor cores, the segment slack should be assigned so that

$$f_{i,1}.\frac{e_{i,1}}{2} + f_{i,2}.\frac{e_{i,2}}{2} + f_{i,3}.\frac{e_{i,3}}{2} + \cdots + f_{i,s_i}.\frac{e_{i,s_i}}{2} \le L_i.$$

In our decomposition, we always assign the maximum possible segment slack on 2-speed processor cores and,

therefore, for our decomposition, the above inequality is in fact an equality.

Since after assigning slack, 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. Also, we want to keep the density of any thread at most 1 on 2-speed processor cores. Hence, 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 $m_{i,j} > \frac{C_{i,2}}{T_i - P_{i,2}}$. That is, they have many parallel threads.
- *Light segments* are those which have $m_{i,j} \leq \frac{C_{i,2}}{T_i - P_{i,2}}$.

Using that 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 $m_{i,j} \leq \frac{C_{i,2}}{T_i - P_{i,2}}$ are assigned $f_{i,j} = 0$. The total available slack L_i is distributed among the heavy segments (segments with $m_{i,j} > \frac{C_{i,2}}{T_i - P_{i,2}}$) 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 be represented by: $\{\langle e^h_{i,1}, m^h_{i,1}\rangle, \langle e^h_{i,2}, m^h_{i,2}\rangle, \cdots, \langle e^h_{i,s^h_i}, m^h_{i,s^h_i}\rangle\}$, where $s^h_i \leq s_i$ (superscript h standing for 'heavy'). Then, let

$$P^h_{i,2} = \frac{1}{2}\sum_{j=1}^{s^h_i} e^h_{i,j}; \qquad C^h_{i,2} = \frac{1}{2}\sum_{j=1}^{s^h_i} m^h_{i,j}.e^h_{i,j} \tag{4}$$

Now, let the light segments be represented by: $\{\langle e^\ell_{i,1}, m^\ell_{i,1}\rangle, \langle e^\ell_{i,2}, m^\ell_{i,2}\rangle, \cdots, \langle e^\ell_{i,s^\ell_i}, m^\ell_{i,s^\ell_i}\rangle\}$, where $s^\ell_i = s_i - s^h_i$ (superscript ℓ standing for 'light'). Then, let

$$P^\ell_{i,2} = \frac{1}{2}\sum_{j=1}^{s^\ell_i} e^\ell_{i,j}; \qquad C^\ell_{i,2} = \frac{1}{2}\sum_{j=1}^{s^\ell_i} m^\ell_{i,j}.e^\ell_{i,j} \tag{5}$$

Now, the following relations hold:

$$P_{i,2} = \frac{P_i}{2} = P^h_{i,2} + P^\ell_{i,2}; \qquad C_{i,2} = \frac{C_i}{2} = C^h_{i,2} + C^\ell_{i,2} \tag{6}$$

Now we calculate slack fraction $f^h_{i,j}$ for all heavy segments (i.e. segments $\langle e^h_{i,j}, m^h_{i,j}\rangle$, where $1 \leq j \leq s^h_i$ and $m^h_{i,j} > \frac{C_{i,2}}{T_i - P_{i,2}}$) so that they all have equal density on 2-speed processor cores. That is,

$$\frac{m^h_{i,1}}{1 + f^h_{i,1}} = \frac{m^h_{i,2}}{1 + f^h_{i,2}} = \frac{m^h_{i,3}}{1 + f^h_{i,3}} = \cdots = \frac{m^h_{i,s^h_i}}{1 + f^h_{i,s^h_i}} \tag{7}$$

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

$$f^h_{i,1}.e^h_{i,1} + f^h_{i,2}.e^h_{i,2} + f^h_{i,3}.e^h_{i,3} + \cdots + f^h_{i,s^h_i}.e^h_{i,s^h_i} = 2.L_i \tag{8}$$

It follows that the value of each $f^h_{i,j}$, $1 \leq j \leq s^h_i$, can be determined by solving Equations 7 and 8 as shown below.

From Equation 7, each $f^h_{i,j}$, $2 \leq j \leq s^h_i$, can be expressed in terms of $f^h_{i,1}$ as follows

$$f^h_{i,j} = (1 + f^h_{i,1})\frac{m^h_{i,j}}{m^h_{i,1}} - 1 \tag{9}$$

Putting this value of each $f^h_{i,j}$, $2 \leq j \leq s^h_i$, into Equation 8:

$$f^h_{i,1}e^h_{i,1} + \sum_{j=2}^{s^h_i}\left(\left((1 + f^h_{i,1})\frac{m^h_{i,j}}{m^h_{i,1}} - 1\right)e^h_{i,j}\right) = 2L_i$$

$$\Leftrightarrow f^h_{i,1}e^h_{i,1} + \sum_{j=2}^{s^h_i}\left(\frac{m^h_{i,j}}{m^h_{i,1}}e^h_{i,j} + f^h_{i,1}\frac{m^h_{i,j}}{m^h_{i,1}}e^h_{i,j} - e^h_{i,j}\right) = 2L_i$$

$$\Leftrightarrow f^h_{i,1}e^h_{i,1} + \frac{1}{m^h_{i,1}}\sum_{j=2}^{s^h_i} m^h_{i,j}e^h_{i,j} + \frac{f^h_{i,1}}{m^h_{i,1}}\sum_{j=2}^{s^h_i} m^h_{i,j}e^h_{i,j} - \sum_{j=2}^{s^h_i} e^h_{i,j} = 2L_i$$

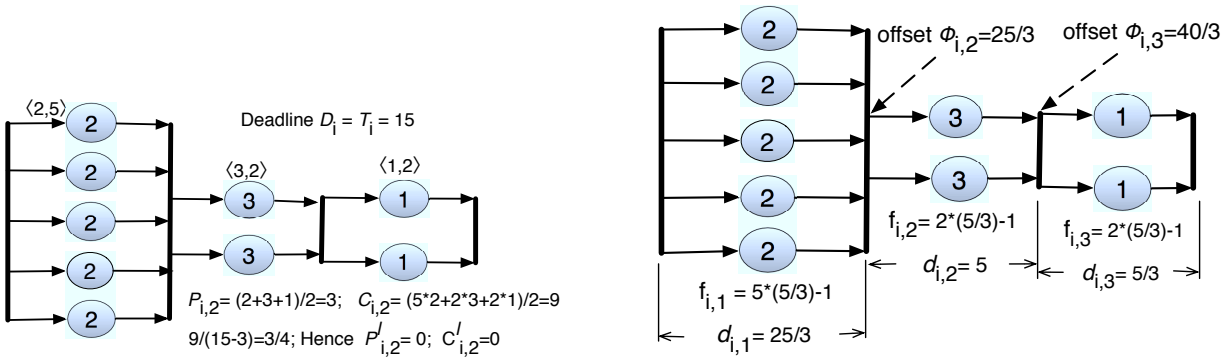
$$\Leftrightarrow f^h_{i,1} = \frac{2L_i + \sum_{j=2}^{s^h_i} e^h_{i,j} - \frac{1}{m^h_{i,1}}\sum_{j=2}^{s^h_i} m^h_{i,j}e^h_{i,j}}{e^h_{i,1} + \frac{1}{m^h_{i,1}}\sum_{j=2}^{s^h_i} m^h_{i,j}e^h_{i,j}}$$

$$\Leftrightarrow f^h_{i,1} = \frac{2L_i + (\sum_{j=2}^{s^h_i} e^h_{i,j} + e^h_{i,1}) - (e^h_{i,1} + \frac{1}{m^h_{i,1}}\sum_{j=2}^{s^h_i} m^h_{i,j}e^h_{i,j})}{e^h_{i,1} + \frac{1}{m^h_{i,1}}\sum_{j=2}^{s^h_i} m^h_{i,j}e^h_{i,j}}$$

$$\Leftrightarrow f^h_{i,1} = \frac{2L_i + \sum_{j=1}^{s^h_i} e^h_{i,j}}{e^h_{i,1} + \frac{1}{m^h_{i,1}}\sum_{j=2}^{s^h_i} m^h_{i,j}e^h_{i,j}} - 1$$

$$\Leftrightarrow f^h_{i,1} = \frac{2L_i + 2P^h_{i,2}}{e^h_{i,1} + \frac{1}{m^h_{i,1}}\sum_{j=2}^{s^h_i} m^h_{i,j}e^h_{i,j}} - 1 \qquad \text{(From 4)}$$

$$\Leftrightarrow f^h_{i,1} = \frac{m^h_{i,1}(2L_i + 2P^h_{i,2})}{m^h_{i,1}e^h_{i,1} + \sum_{j=2}^{s^h_i} m^h_{i,j}e^h_{i,j}} - 1$$



(a) A parallel synchronous task τ_i

(b) Decomposed task τ_i^{decom}

Figure 2. An example of decomposition

$$\Leftrightarrow f^h_{i,1} = \frac{m^h_{i,1}(2L_i + 2P^h_{i,2})}{2C^h_{i,2}} - 1 \qquad \text{(From 4)}$$

$$\Leftrightarrow f^h_{i,1} = \frac{m^h_{i,1}(L_i + P^h_{i,2})}{C_{i,2} - C^\ell_{i,2}} - 1 \qquad \text{(From 6)}$$

$$\Leftrightarrow f^h_{i,1} = \frac{m^h_{i,1}((T_i - P_{i,2}) + P^h_{i,2})}{C_{i,2} - C^\ell_{i,2}} - 1 \qquad \text{(From 2)}$$

$$\Leftrightarrow f^h_{i,1} = \frac{m^h_{i,1}(T_i - (P^h_{i,2} + P^\ell_{i,2}) + P^h_{i,2})}{C_{i,2} - C^\ell_{i,2}} - 1 \qquad \text{(From 6)}$$

$$\Leftrightarrow f^h_{i,1} = \frac{m^h_{i,1}(T_i - P^\ell_{i,2})}{C_{i,2} - C^\ell_{i,2}} - 1$$

Now putting the above value of $f^h_{i,1}$ in Equation 7, for any heavy segment $\langle e^h_{i,j}, m^h_{i,j}\rangle$:

$$f^h_{i,j} = \frac{m^h_{i,j}(T_i - P^\ell_{i,2})}{C_{i,2} - C^\ell_{i,2}} - 1 \qquad (10)$$

Intuitively, the slack never should be negative, since the deadline should be no less than the computation requirement of the thread. Since $m^h_{i,j} > \frac{C_{i,2}}{T_i - P_{i,2}}$, according to Equation 10, the quantity $\frac{m^h_{i,j}(T_i - P^\ell_{i,2})}{C_{i,2} - C^\ell_{i,2}} \geq 1$. This implies that $f^h_{i,j} \geq 0$. Now, using Equation 3, the density of every segment $\langle e^h_{i,j}, m^h_{i,j}\rangle$ is:

$$\frac{m^h_{i,j}}{1 + f^h_{i,j}} = \frac{m^h_{i,j}}{1 + \frac{m^h_{i,j}(T_i - P^\ell_{i,2})}{C_{i,2} - C^\ell_{i,2}} - 1} = \frac{C_{i,2} - C^\ell_{i,2}}{T_i - P^\ell_{i,2}} \qquad (11)$$

Figure 2 shows a simple example of decomposition for a task τ_i consisting of 3 segments.

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

$$f_{i,j} = \frac{L_i}{P_{i,2}} \qquad (12)$$

By Equation 3, the density of each segment $\langle e_{i,j}, m_{i,j}\rangle$ is:

$$\frac{m_{i,j}}{1 + f_{i,j}} = \frac{m_{i,j}}{1 + \frac{L_i}{P_{i,2}}} = m_{i,j}\frac{P_{i,2}}{L_i + P_{i,2}} = m_{i,j}\frac{P_{i,2}}{T_i} \qquad (13)$$

C. Density Analysis

Once the above decomposition is done on task τ_i: $(\langle e_{i,1}, m_{i,1}\rangle, \langle e_{i,2}, m_{i,2}\rangle, \cdots, \langle e_{i,s_i}, m_{i,s_i}\rangle)$, each thread of each segment $\langle e_{i,j}, m_{i,j}\rangle$, $1 \leq j \leq s_i$, is considered as a sequential multiprocessor subtask. 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. Since $f_{i,j} \geq 0$, $\forall 1 \leq j \leq s_i$, $\forall 1 \leq i \leq n$, the maximum density $\delta_{\max,2}$ of any subtask (thread) among τ^{decom} on 2-speed processor cores:

$$\delta_{\max,2} = \max\{\frac{1}{1 + f_{i,j}}\} \leq 1 \qquad (14)$$

Lemma 1 shows that the density of every segment is at most $\frac{C_i/2}{T_i - P_i/2}$ for any task with or without heavy segments.

Lemma 1. *After the decomposition, the density of every segment $\langle e_{i,j}, m_{i,j}\rangle$, where $1 \leq j \leq s_i$, of every task τ_i on 2-speed processor cores is upper bounded by $\frac{C_i/2}{T_i - P_i/2}$.*

Proof: First, we analyze the case when the task contains some heavy segments. According to Equation 11, for every heavy segment $\langle e_{i,j}, m_{i,j}\rangle$, the density is:

$$\frac{C_{i,2} - C^\ell_{i,2}}{T_i - P^\ell_{i,2}} \leq \frac{C_{i,2}}{T_i - P^\ell_{i,2}} \qquad (\text{since } C^\ell_{i,2} \geq 0)$$

$$\leq \frac{C_{i,2}}{T_i - P_{i,2}} \qquad (\text{since } P_{i,2} \geq P^\ell_{i,2})$$

$$= \frac{C_i/2}{T_i - P_i/2}$$

For every light segment $\langle e_{i,j}, m_{i,j}\rangle$, $f_{i,j} = 0$. That is, its deadline is equal to its computation requirement $\frac{e_{i,j}}{2}$ on 2-

speed processor cores. Therefore, its density is:

$$\frac{m_{i,j}}{1+f_{i,j}} = m_{i,j} \leq \frac{C_{i,2}}{T_i - P_{i,2}} = \frac{C_i/2}{T_i - P_i/2}$$

For the case when there are no heavy segments in τ_i, for every segment $\langle e_{i,j}, m_{i,j} \rangle$ of τ_i, $m_{i,j} \leq \frac{C_{i,2}}{T_i - P_{i,2}}$. Since $T_i \geq P_{i,2}$ (Observation 1), the density of each segment $\langle e_{i,j}, m_{i,j} \rangle$ (Equation 13) of τ_i:

$$m_{i,j}\frac{P_{i,2}}{T_i} \leq m_{i,j} \leq \frac{C_{i,2}}{T_i - P_{i,2}} = \frac{C_i/2}{T_i - P_i/2}$$

Hence, follows the lemma. ■

Thus, our decomposition distributes the slack so that each segment has a density that is bounded above. Theorem 2 establishes an upper bound on the density of every task after decomposition.

Theorem 2. *The density $\delta_{i,2}$ of every τ_i^{decom}, $1 \leq i \leq n$, (i.e. the density of every task τ_i after decomposition) on 2-speed processor cores is upper bounded by $\frac{C_i/2}{T_i - P_i/2}$.*

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 occurences of the segment that has the maximum density, and therefore, the density of the entire task set τ_i^{decom} is equal to that of the segment having the maximum density which is at most $\frac{C_i/2}{T_i - P_i/2}$ (Lemma 1). Therefore, $\delta_{i,2} \leq \frac{C_i/2}{T_i - P_i/2}$. ■

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 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, then a schedule must exist for τ where each task in τ can meet its deadline. ■

IV. Global EDF Scheduling

After our proposed decomposition, we consider the scheduling of synchronous parallel tasks. Lakshmanan et al. [4] show that there exist task sets with total utilization slightly greater than (and arbitrarily close to) 1 that are unschedulable with m 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 [4]) use the *resource augmentation bound approach*, originally introduced in [7]. 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 m cores represents a processor speedup factor. That is, if there exists any (optimal) algorithm under which a task set is feasible on m identical unit-speed processor cores, then $\mathbb{A}$ is guaranteed to successfully schedule this task set on a m-core processor, where each processor core is ν times as fast as the original. In Theorem 5, we show that, our proposed decomposition needs a resource augmentation bound of 4 under global EDF scheduling.

Our analysis uses a result for constrained deadline sporadic sequential tasks on m processors [8], stated in Theorem 4. This result is a generalization of the result for implicit deadline sporadic tasks [9].

Theorem 4. *(From [8]) Any constrained deadline sporadic task set π with total density $\delta_{\text{sum}}(\pi)$ and maximum density $\delta_{\max}(\pi)$ is schedulable using global EDF strategy on m unit-speed processor cores if*

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

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 m unit-speed processor cores using any (optimal) algorithm, then the transformed task set τ^{decom} is schedulable using global EDF on m processor cores of speed 4.*

Proof: Let there exist some algorithm $\mathbb{A}$ under which the original task set τ is feasible on m identical unit-speed processor cores. If τ is schedulable under $\mathbb{A}$, the following condition must hold (Observation 2):

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq m \tag{15}$$

We decompose tasks considering that each processor core has speed 2. To be able to schedule the decomposed tasks τ^{decom}, let we need to increase the speed of each processor core ν times further. That is, we need each processor core to be of speed 2ν.

On an m-core platform where each processor core has speed 2ν, let the total density and the maximum density

of task set τ^{decom} be denoted by $\delta_{\text{sum},2\nu}$ and $\delta_{\max,2\nu}$, respectively. From 14, we have

$$\delta_{\max,2\nu} = \frac{\delta_{\max,2}}{\nu} \leq \frac{1}{\nu} \tag{16}$$

The value $\delta_{\text{sum},2\nu}$ turns out to be the total density of all decomposed tasks. By Theorem 2 and Equation 1, the density of every task τ_i^{decom} on m identical processors each of speed 2ν is:

$$\delta_{i,2\nu} \leq \frac{\frac{C_i}{2\nu}}{T_i - \frac{P_i}{2}} \leq \frac{\frac{C_i}{2\nu}}{T_i - \frac{T_i}{2}} = \frac{\frac{C_i}{2\nu}}{\frac{T_i}{2}} = \frac{1}{\nu}.\frac{C_i}{T_i} \quad (\text{since } P_i \leq T_i)$$

$$\text{Thus, from 15, } \delta_{\text{sum},2\nu} = \sum_{i=1}^{n} \delta_{i,2\nu} \leq \frac{1}{\nu}\sum_{i=1}^{n}\frac{C_i}{T_i} \leq \frac{m}{\nu} \tag{17}$$

Note that, in the decomposed task set, every thread of the original task is a sequential task on a multiprocessor platform. Therefore, $\delta_{\text{sum},2\nu}$ is the total density of all threads (i.e. subtasks), and $\delta_{\max,2\nu}$ is the maximum density among all threads. Thus, by Theorem 4, following is the sufficient condition for EDF schedulability of the decomposed task set on m processor cores each of speed 2ν:

$$\delta_{\text{sum},2\nu} \leq m - (m-1)\delta_{\max,2\nu}$$

We can substitute the values of $\delta_{\text{sum},2\nu}$ (Equation 17) and $\delta_{\max,2\nu}$ (Equation 16) into this equation, and say that the task set τ^{decom} is schedulable if

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

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

$$\frac{2}{\nu} \leq 1 \ \Leftrightarrow \ \nu \geq 2 \ \Leftrightarrow \ 2\nu \geq 4$$

Hence follows the theorem. ■

V. Partitioned Deadline Monotonic Scheduling

Using the same decomposition described in Section III, we now derive a resource augmentation bound required to schedule task sets under FBB-FFD (Fisher Baruah Baker - First-Fit Decreasing) partitioned Deadline Monotonic (DM) scheduling [10] which the previous work on parallel task scheduling [4] 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 are able to obtain a resource augmentation bound of 5 as shown below.

In *partitioned scheduling*, tasks are executed only on their assigned processor cores, and are not allowed to migrate among 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 DM priorities, and each subtask is allocated to the first available core that satisfies the condition specified in the FBB-FFD algorithm for allocation.

We use an analysis similar to the one used in [4] 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.

Definition 5. *The demand bound function (DBF), originally introduced in [11], 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:*

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

Definition 6. *Based upon the DBF function, the load of task system τ, denoted by $\lambda(\tau)$, is defined as follows:*

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

From Definition 5, for a constrained deadline task τ_i:

$$DBF(\tau_i, t) \leq \max\left(0, \left(\left\lfloor \frac{t - D_i}{D_i} \right\rfloor + 1\right) C_i\right)$$
$$\leq \left\lfloor \frac{t}{D_i} \right\rfloor C_i \ \leq \frac{t}{D_i}.C_i \ = \delta_i.t$$

Based on the above analysis, we now derive an upper bound of DBF for every task after decomposition. Every segment of task τ_i consists of a set of constrained deadline subtasks after decomposition and, by Lemma 1, the total density of all subtasks in a segment is at most $\frac{C_i/2}{T_i - P_i/2}$. The constrained deadline subtasks are offset to ensure that those belonging to different segments of the same task are never simultaneously active. That is, for each task τ_i, each segment (and each of its subtasks) happens only once during the activation time of τ_i. Therefore, for decomposed task τ_i^{decom}, considering the segment having the maximum density in place of every segment gives an upper bound on the total density of all subtasks of τ_i^{decom}. Since, the density $\delta_{i,j}$ of any j-th segment of τ_i^{decom} is at most $\frac{C_i/2}{T_i - P_i/2}$, the DBF of τ_i^{decom} over any interval of length t is:

$$DBF(\tau_i^{\text{decom}}, t) \leq \frac{C_i/2}{T_i - P_i/2}.t$$

The load of the decomposed task system τ^{decom} is:

$$\lambda(\tau^{\text{decom}}) = \max_{t>0} \left(\frac{\sum_{i=1}^{n} DBF(\tau_i^{\text{decom}}, t)}{t} \right) \leq \sum_{i=1}^{n} \frac{C_i/2}{T_i - P_i/2} \tag{18}$$

Theorem 6. *If a synchronous parallel task set τ is schedulable using an optimal algorithm on m unit-speed processor cores, then its decomposed task set τ^{decom} is schedulable using FBB-FDD on m identical processors of speed 5.*

Proof: According to [10], any constrained deadline sporadic task set π with total utilization $u_{\text{sum}}(\pi)$, maximum density $\delta_{\max}(\pi)$, and load $\lambda(\pi)$ is schedulable by FBB-FFD on m unit-capacity processors if

$$m \geq \frac{\lambda(\pi) + u_{\text{sum}}(\pi) - \delta_{\max}(\pi)}{1 - \delta_{\max}(\pi)} \tag{19}$$

We decompose tasks considering that each processor core has speed 2. To be able to schedule the decomposed tasks τ^{decom}, let we need to increase the speed of each processor core ν times further. That is, we need each processor core to be of speed 2ν. Let the maximum density, total utilization, and load of task set τ^{decom} be denoted by $\delta_{\max,2\nu}$, $u_{\text{sum},2\nu}$, and $\lambda_{2\nu}$ respectively, when each processor core has speed 2ν. From Equation 1:

$$u_{\text{sum},2\nu} = \sum_{i=1}^{n} \frac{\frac{C_i}{2\nu}}{T_i} = \frac{1}{2\nu} \sum_{i=1}^{n} \frac{C_i}{T_i} = \frac{u_{\text{sum}}}{2\nu} \tag{20}$$

From Equations 1 and 18:

$$\lambda_{2\nu} \leq \sum_{i=1}^{n} \frac{\frac{C_i}{2\nu}}{T_i - \frac{P_i}{2}} \leq \sum_{i=1}^{n} \frac{\frac{C_i}{2\nu}}{T_i - \frac{T_i}{2}} = \frac{1}{\nu} \sum_{i=1}^{n} \frac{C_i}{T_i} = \frac{u_{sum}}{\nu} \tag{21}$$

Since the task set τ^{decom} contains sequential tasks, due to Condition 19, the following is the sufficient condition for FBB-FFD schedulability of the decomposed task set on m identical processors each of speed 2ν:

$$m \geq \frac{\lambda_{2\nu} + u_{\text{sum},2\nu} - \delta_{\max,2\nu}}{1 - \delta_{\max,2\nu}} \tag{22}$$

Using, Equations 21, 20, 16 in Equation 22, task set τ^{decom} is schedulable if

$$m \geq \frac{\frac{u_{sum}}{\nu} + \frac{u_{sum}}{2\nu} - \frac{1}{\nu}}{1 - \frac{1}{\nu}}$$

If the original task set τ is schedulable by any algorithm on m unit-speed processor cores, then $u_{sum} \leq m$. Therefore, τ^{decom} is schedulable if

$$m \geq \frac{\frac{m}{\nu} + \frac{m}{2\nu} - \frac{1}{\nu}}{1 - \frac{1}{\nu}} \quad \Leftarrow \quad 2\nu - 2 \geq 3 \quad \Leftrightarrow \quad 2\nu \geq 5$$

Hence follows the theorem. ∎

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. Therefore, a DAG can be converted into a synchronous task by simply adding new dependence edges as explained below.

If there is an edge from node u to node v, we say that u is the *parent* of v. Then we calculate the depth, denoted by $h(v)$, of each node v. If v has no parents, then it is assigned depth 1. Otherwise, we calculate the depth of v as follows:

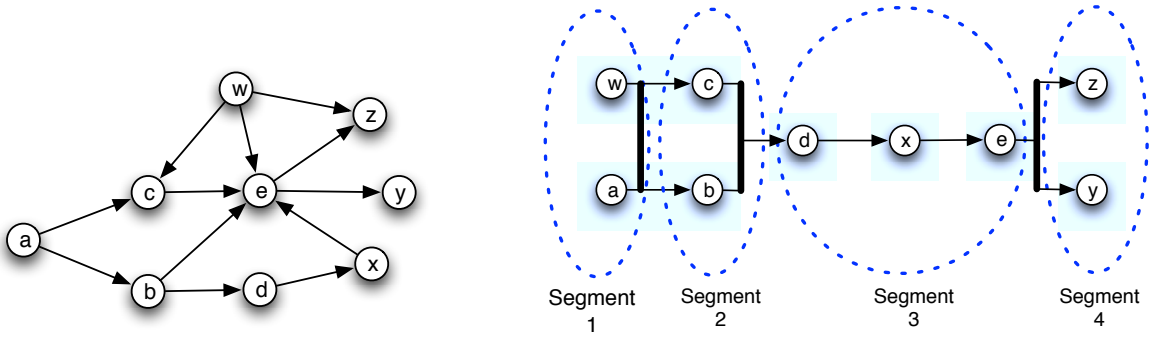
$$h(v) = \max_{u \text{ parent of } v} h(u) + 1$$

Each node with depth j is assigned to segment j. Then every node of the DAG is considered as a *thread* in the corresponding segment. The threads in the same segment can happen in parallel, and the segment is considered as a parallel segment of a synchronous task. If there are $k > 1$ consecutive segments each consisting 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 an 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 dependences in the DAG, and in fact only adds extra dependences.

Upon representing a DAG task as a synchronous task, we perform the same decomposition proposed in Section III. The decomposed task set can be scheduled using either global EDF or partitioned DM scheduling. Note that the transformation from a DAG task τ_i to a synchronous task preserves the work C_i of τ_i. Hence, the condition $\sum C_i/T_i < m$ used in our analysis still holds. Besides, the transformation preserves the critical path length P_i of τ_i and, hence, the rest of the analysis also holds. Therefore, a set of DAG tasks can be scheduled with a resource augmentation of 4 under global EDF scheduling, and of 5 under partitioned DM scheduling.

VII. Related Work

There has been extensive work on traditional multiprocessor real-time scheduling [12]. Most of this work focuses on scheduling sequential tasks on multiprocessor or multi-core



(a) DAG

(b) Parallel synchronous model

Figure 3. DAG to parallel synchronous model

systems. There has also been extensive work on scheduling of one or more parallel jobs on multiprocessors [13]–[22]. However, the work in [13]–[19] does not consider task deadlines, and that in [20]–[22] considers soft real-time scheduling. In contrast to the goal (i.e. to meet all task deadlines) of a hard real-time system, in a *soft real-time system* the goal is to meet a certain subset of deadlines based on some application specific criteria.

There has been little work on hard real-time scheduling of parallel tasks. Anderson et al. [23] propose the concept of a megatask as a way to reduce miss rates in shared caches on multicore platforms, and consider Pfair scheduling by inflating the weights of a megatask's component tasks. Preemptive fixed-priority scheduling of parallel tasks is shown to be NP-hard by Han et al. [24]. Kwon et al. [25] explore preemptive EDF scheduling of parallel task systems with linear-speedup parallelism. Wang et al. [26] consider a heuristic for nonpreemptive scheduling. However, this work focuses on metrics like makespan [26] or total work that meets deadline [25], and considers simple task models where a task is executed on up to a given number of processors.

Most of the other work on real time scheduling of parallel tasks also address simplistic task models. Jansen [27], Lee et al. [28], and Collette et al. [29] study the scheduling of *malleable tasks*, where each task is assumed to execute on a given number of cores or processors and this number may change during execution. Manimaran et al. [30] study non-preemptive EDF scheduling for *moldable tasks*, where the actual number of used processors is determined before starting the system and remains unchanged. Kato et al. [31] address Gang EDF scheduling of moldable parallel task systems. They require the users to select at 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. In contrast, the parallel task model addressed in this paper allows tasks to have different numbers of threads in different stages, which makes our solution applicable to a much broader range of applications.

Our work is most related to, and is inspired by, the recent work of Lakshmanan et al. on real-time scheduling for a restrictive synchronous parallel task model [4]. In their model, every task is an alternate sequence of parallel and sequential segments. All the parallel segments in a task have an *equal* number of threads, and that number cannot exceed the total number of processor cores. They also convert each parallel task into a set of sequential tasks, and then analyzes the resource augmentation bound for partitioned DM scheduling. However, their strategy of decomposition is different from ours. They use a *stretch transformation* that makes a master thread of execution requirement equal to task period, and assign one processor core exclusively to it. The remaining threads are scheduled using FBB-FDD algorithm. Unlike ours, their results do not hold if, in a task, the number of threads in different segments vary, or exceed the number of cores. In addition, tasks that can be expressed as a DAG of unit time nodes cannot be converted to their task model in a work and critical path length conserving manner. Therefore, unlike ours, their analysis does not directly apply to these more general task models.

VIII. Conclusion

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, in this paper we have presented new results on real-time scheduling for generalized parallel task models. First, we have considered a general synchronous parallel task model where each task consists of segments, each having an arbitrary number of parallel threads. We have then proposed a novel task decomposition algorithm that decomposes each parallel task into a set of sequential tasks. We have derived a resource augmentation bound of 4 under global EDF scheduling, which to our knowledge is the first resource augmentation bound for global EDF scheduling of parallel tasks. We have also derived a resource

augmentation bound of 5 for partitioned DM scheduling. Finally, we have shown how to convert a task represented as a Directed Acyclic Graph (DAG) with unit time nodes into a synchronous task, thereby holding our results for this more general task model.

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

ACKNOWLEDGEMENT

This research was supported by NSF under grants CNS-0448554 (CAREER) and CNS-1017701 (NeTS).

REFERENCES

[1] "Teraflops research chip," http://techresearch.intel.com/ProjectDetails.aspx?Id=151.

[2] "CoSy compiler for 96-core multi-threaded array processor," http://www.clearspeed.com/newsevents/news/ClearSpeed_Ace_011708.php.

[3] 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*.

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

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

[6] "Intel® Cilk™Plus," http://software.intel.com/en-us/articles/intel-cilk-plus.

[7] S. Funk, J. Goossens, and S. Baruah, "On-line scheduling on uniform multiprocessors," in *RTSS '01*.

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

[9] 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.

[10] N. Fisher, S. Baruah, and T. P. Baker, "The partitioned scheduling of sporadic tasks according to static-priorities," in *ECRTS '06*.

[11] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *RTSS '90*.

[12] R. Davis and A. Burns, "A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems," University of York, Department of Computer Science, Tech. Rep. YCS-2009-443, 2009.

[13] 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.

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

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

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

[17] 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.

[18] 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.

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

[20] J. M. Calandrino and J. H. Anderson, "Cache-aware real-time scheduling on multicore platforms: Heuristics and a case study," in *ECRTS '08*.

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

[22] J. M. Calandrino, D. Baumberger, T. Li, S. Hahn, and J. H. Anderson, "Soft real-time scheduling on performance asymmetric multicore platforms," in *RTAS '07*.

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

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

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

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