Report Number: WUCSE-2011-45

2011

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

Abusayeed Saifullah, Kunal Agrawal, Chenyang Lu, and Christopher Gill

Multi-core processors over 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... Read complete abstract on page 2.

[Department of Computer Science & Engineering](#) - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

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

Abusayeed Saifullah, Kunal Agrawal, Chenyang Lu, and Christopher Gill

Complete Abstract:

Multi-core processors over 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 a directed acyclic graph (DAG) task model where each node in the DAG has a unit execution requirement. We show how these tasks can be converted into synchronous tasks such that the same decomposition can be applied and the same augmentation bounds hold. Simulations based on synthetic workload demonstrate that the derived resource augmentation bounds are safe and sufficient.

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

**Abusayeed Saifullah · Kunal Agrawal ·
Chenyang Lu · Christopher Gill**

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 a directed acyclic graph (DAG) task model where each node in the DAG has a unit execution requirement. We show how these tasks can be converted into synchronous tasks such that the same decomposition can be applied and the same augmentation bounds hold. Simulations based on synthetic workload demonstrate that the derived resource augmentation bounds are safe and sufficient.

Keywords parallel task · multi-core processor · real-time scheduling · resource augmentation bound

Abusayeed Saifullah, Kunal Agrawal, Chenyang Lu, Christopher Gill
Department of Computer Science and Engineering
Washington University in St. Louis
Campus Box. 1045
St Louis, MO 63130, USA
E-mail: {saifullaha, kunal, lu, cdgill}@cse.wustl.edu

1 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 increasing processor clock frequencies to increasing the number of cores per processor. For example, Intel has recently put 80 cores in a Teraflops Research Chip (Intel, 2007) with a view to making it generally available, and ClearSpeed has developed a 96-core processor (ClearSpeed, 2008). While hardware technology is moving at a rapid pace, software and programming models have failed to keep pace. For example, Intel (2007) 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 (Huang et al, 2010) 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 had previously mostly focused on sequential task models. The state-of-the-art work (Lakshmanan et al, 2010) on parallel scheduling for real-time tasks with intra-task parallelism 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 the execution requirements of all threads in a segment are equal. Most importantly, in that task model, the number of threads in every segment is *no greater* than the total number of processor cores.

While the work presented by Lakshmanan et al (2010) 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, it analyzes the resource augmentation bound under partitioned DM scheduling only, and does not consider other scheduling policies such as global EDF. In this work, we consider real-time scheduling on multi-core processors for a more general synchronous task model. Our tasks still contain segments where the threads of each segment synchro-

nize at its end. However, in contrast to the restrictive task model addressed in Lakshmanan et al (2010), 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. Furthermore, the execution requirements of the threads in any segment can vary. 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 periodic parallel tasks.

- 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 where each node in a DAG has a unit execution requirement. This is an even more general model for parallel tasks. Namely, we show that we can transform unit-node DAG tasks into synchronous tasks, and then use our proposed decomposition to get the same resource augmentation bounds for the former.

We evaluate the performance of the proposed decomposition through simulations based on synthetic workloads. The results indicate that the derived bounds are safe and sufficient. In particular, the resource augmentations required to schedule the decomposed tasks in our simulations are at most 2.4 and 3.4 for global EDF and partitioned DM scheduling, respectively, which are significantly smaller than the corresponding theoretical bounds.

In the rest of the paper, Section 2 describes the parallel synchronous task model. Section 3 presents the proposed task decomposition. Section 4 presents the analysis for global EDF scheduling. Section 5 presents the analysis for partitioned DM scheduling. Section 6 extends our results and analyses for unit-node DAG task models. Section 7 presents the simulation results. Section 8 reviews related work. Finally, we conclude in Section 9.

2 Parallel Synchronous Task Model

We primarily consider a synchronous parallel task model, where each task consists of a sequence of computation segments, each segment having an arbitrary number of parallel threads with arbitrary execution requirements that 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 (OpenMP, 2011) and CilkPlus (Intel, 2010).

We consider n periodic synchronous parallel tasks with implicit deadlines. Each task τ_i, $1 \leq i \leq n$, is a sequence of s_i segments, where the j-th segment, $1 \leq j \leq s_i$, consists of $m_{i,j}$ parallel threads. First we consider the case when, for any segment of τ_i, all parallel threads in the segment have equal execution requirements. For such τ_i, the j-th segment, $1 \leq j \leq s_i$, is represented by $\langle e_{i,j}, m_{i,j} \rangle$, with $e_{i,j}$ 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 the $(j-1)$-th segment have completed. Thus, a parallel task τ_i in which a segment consists of equal-length threads is shown in Figure 1, and is represented as $\tau_i : (\langle e_{i,1}, m_{i,1} \rangle, \langle e_{i,2}, m_{i,2} \rangle, \cdots, \langle e_{i,s_i}, m_{i,s_i} \rangle)$ where

- s_i is the total number of segments in task τ_i.
- In a segment $\langle e_{i,j}, m_{i,j} \rangle$, $1 \leq j \leq s_i$, $e_{i,j}$ 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.
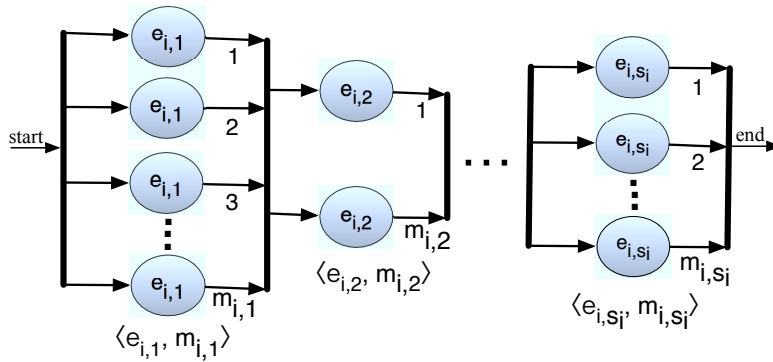

Fig. 1 A parallel synchronous task τ_i

Now, we consider the case when the execution requirements of parallel threads in a segment of τ_i may differ from each other. An example of such a task is shown in Figure 2(a), where each horizontal bar indicates the length of the execution requirement of a thread. As the figure shows, the parallel threads in the third segment have unequal execution requirements. By adding a new synchronization point at the end of each thread in a segment, any segment consisting of threads of unequal length can be converted to several segments each consisting of threads of equal length as shown in Figure 2(b).

Specifically, for the task with unequal-length threads in a segment shown in Figure 2(a), Figure 2(b) shows the corresponding task in which each segment consists of equal-length threads. Note that such a conversion does not change any task parameter such as period, deadline, or execution requirement. Thus, in any synchronous parallel task, any segment consisting of threads of different execution requirements can be converted to several segments each consisting of threads of an equal execution requirement. Hence, we concentrate only to the task model where each segment in a task consists of equal-length threads (such as the one shown in Figure 1).
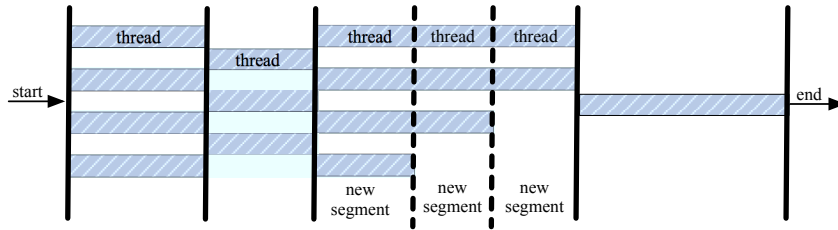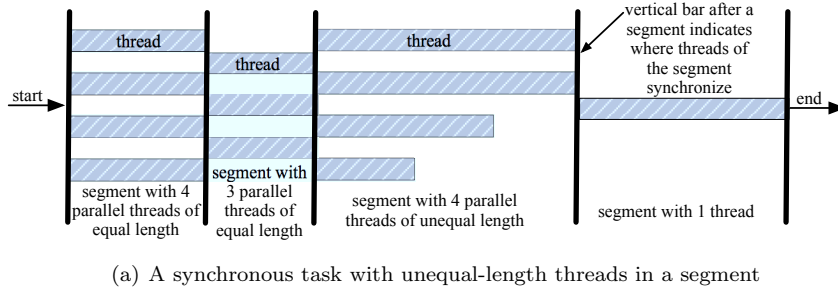

(a) A synchronous task with unequal-length threads in a segment

(b) The corresponding synchronous task with equal-length threads in each segment (each dotted vertical line indicates a newly added synchronization point at the end of a thread)

Fig. 2 Conversion of a segment with unequal-length threads to segments with equal-length threads in a synchronous parallel task

Therefore, considering a multi-core platform consisting of m processor cores, we focus on scheduling n parallel tasks denoted by $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$, where each τ_i is represented as $\tau_i : (\langle e_{i,1}, m_{i,1} \rangle, \langle e_{i,2}, m_{i,2} \rangle, \cdots, \langle e_{i,s_i}, m_{i,s_i} \rangle)$ (as the one shown in Figure 1). 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 task set is said to be *schedulable* when all tasks meet their deadlines.

3 Task Decomposition

In this section, we present a decomposition of the parallel tasks into a set of sequential tasks. In particular, we propose a strategy that decomposes each implicit deadline parallel task (like the one shown in Figure 1) 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 4 and 5). Here, we first present some useful terminology. We then present our decomposition and a density analysis for it.

3.1 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

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

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

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

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* $u_{\mathrm{sum}}(\tau)$ for the set of n tasks τ on a unit-speed multi-core platform are defined as

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

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* $\delta_{\mathrm{sum}}(\tau)$ of the set of n tasks τ on a unit-speed multi-core platform are defined as follows:

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

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

3.2 Decomposition

Following is the high-level idea of the decomposition of a parallel task τ_i.

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.

$$L_i = T_i - P_{i,2} = T_i - \frac{P_i}{2} \qquad (2)$$

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 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

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

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

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

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, and analyze them separately as follows.

3.2.1 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 $m_{i,j} \leq \frac{C_{i,2}}{T_i - P_{i,2}}$ of τ_i are assigned $f_{i,j} = 0$. The total available slack L_i is distributed among the heavy segments (segments with $m_{i,j} > \frac{C_{i,2}}{T_i - P_{i,2}}$) in such a way that each of these segments has the same density.

For simplicity of presentation, we first distinguish notations between the heavy and light segments. Let the heavy segments of τ_i be represented as $\{\langle e_{i,1}^h, m_{i,1}^h \rangle, \langle e_{i,2}^h, m_{i,2}^h \rangle, \cdots, \langle e_{i,s_i^h}^h, m_{i,s_i^h}^h \rangle\}$, where $s_i^h \leq s_i$ (superscript h standing for 'heavy'). Then, let

$$P_{i,2}^h = \frac{1}{2}\sum_{j=1}^{s_i^h} e_{i,j}^h; \qquad C_{i,2}^h = \frac{1}{2}\sum_{j=1}^{s_i^h} m_{i,j}^h . e_{i,j}^h \qquad (4)$$

The light segments are denoted as $\{\langle e_{i,1}^\ell, m_{i,1}^\ell \rangle, \langle e_{i,2}^\ell, m_{i,2}^\ell \rangle, \cdots, \langle e_{i,s_i^\ell}^\ell, m_{i,s_i^\ell}^\ell \rangle\}$, where $s_i^\ell = s_i - s_i^h$ (superscript ℓ standing for 'light'). Then, let

$$P_{i,2}^\ell = \frac{1}{2}\sum_{j=1}^{s_i^\ell} e_{i,j}^\ell; \qquad C_{i,2}^\ell = \frac{1}{2}\sum_{j=1}^{s_i^\ell} m_{i,j}^\ell . e_{i,j}^\ell \qquad (5)$$

Now, the following equalities must hold for task τ_i.

$$P_{i,2} = \frac{P_i}{2} = P_{i,2}^h + P_{i,2}^\ell; \qquad C_{i,2} = \frac{C_i}{2} = C_{i,2}^h + C_{i,2}^\ell \qquad (6)$$

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

$$\frac{m_{i,1}^h}{1 + f_{i,1}^h} = \frac{m_{i,2}^h}{1 + f_{i,2}^h} = \frac{m_{i,3}^h}{1 + f_{i,3}^h} = \cdots = \frac{m_{i,s_i^h}^h}{1 + f_{i,s_i^h}^h} \qquad (7)$$

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

$$f_{i,1}^h . e_{i,1}^h + f_{i,2}^h . e_{i,2}^h + f_{i,3}^h . e_{i,3}^h + \cdots + f_{i,s_i^h}^h . e_{i,s_i^h}^h = 2.L_i \qquad (8)$$

It follows that the value of each $f_{i,j}^h$, $1 \le j \le s_i^h$, can be determined by solving Equations 7 and 8 as shown below. From Equation 7, the value of $f_{i,j}^h$ for each j, $2 \le j \le s_i^h$, can be expressed in terms of $f_{i,1}^h$ as follows.

$$f_{i,j}^h = (1 + f_{i,1}^h)\frac{m_{i,j}^h}{m_{i,1}^h} - 1 \tag{9}$$

Putting the value of each $f_{i,j}^h$, $2 \le j \le s_i^h$, from Equation 9 into Equation 8:

$$
\begin{aligned}
2L_i &= f_{i,1}^h e_{i,1}^h + \sum_{j=2}^{s_i^h}\left(\left((1 + f_{i,1}^h)\frac{m_{i,j}^h}{m_{i,1}^h} - 1\right)e_{i,j}^h\right) \\
&= f_{i,1}^h e_{i,1}^h + \sum_{j=2}^{s_i^h}\left(\frac{m_{i,j}^h}{m_{i,1}^h}e_{i,j}^h + f_{i,1}^h\frac{m_{i,j}^h}{m_{i,1}^h}e_{i,j}^h - e_{i,j}^h\right) \\
&= f_{i,1}^h e_{i,1}^h + \frac{1}{m_{i,1}^h}\sum_{j=2}^{s_i^h}m_{i,j}^h e_{i,j}^h + \frac{f_{i,1}^h}{m_{i,1}^h}\sum_{j=2}^{s_i^h}m_{i,j}^h e_{i,j}^h - \sum_{j=2}^{s_i^h}e_{i,j}^h
\end{aligned}
$$

From the above equation, we can determine the value of $f_{i,1}^h$ as follows.

$$
\begin{aligned}
f_{i,1}^h &= \frac{2L_i + \sum_{j=2}^{s_i^h}e_{i,j}^h - \frac{1}{m_{i,1}^h}\sum_{j=2}^{s_i^h}m_{i,j}^h e_{i,j}^h}{e_{i,1}^h + \frac{1}{m_{i,1}^h}\sum_{j=2}^{s_i^h}m_{i,j}^h e_{i,j}^h} \\
&= \frac{2L_i + (\sum_{j=2}^{s_i^h}e_{i,j}^h + e_{i,1}^h) - (e_{i,1}^h + \frac{1}{m_{i,1}^h}\sum_{j=2}^{s_i^h}m_{i,j}^h e_{i,j}^h)}{e_{i,1}^h + \frac{1}{m_{i,1}^h}\sum_{j=2}^{s_i^h}m_{i,j}^h e_{i,j}^h} \\
&= \frac{2L_i + \sum_{j=1}^{s_i^h}e_{i,j}^h}{e_{i,1}^h + \frac{1}{m_{i,1}^h}\sum_{j=2}^{s_i^h}m_{i,j}^h e_{i,j}^h} - 1
\end{aligned}
$$

In the above equation, replacing $\sum_{j=1}^{s_i^h}e_{i,j}^h$ with $2P_{i,2}^h$ from Equation 4, we get

$$f_{i,1}^h = \frac{2L_i + 2P_{i,2}^h}{e_{i,1}^h + \frac{1}{m_{i,1}^h}\sum_{j=2}^{s_i^h}m_{i,j}^h e_{i,j}^h} - 1 = \frac{m_{i,1}^h(2L_i + 2P_{i,2}^h)}{m_{i,1}^h e_{i,1}^h + \sum_{j=2}^{s_i^h}m_{i,j}^h e_{i,j}^h} - 1$$

Similarly, in the above equation, replacing $(m_{i,1}^h e_{i,1}^h + \sum_{j=2}^{s_i^h} m_{i,j}^h e_{i,j}^h)$ with $2C_{i,2}^h$ from Equation 4, the value of $f_{i,1}^h$ can be written as follows.

$$
\begin{aligned}
f_{i,1}^h &= \frac{m_{i,1}^h(2L_i + 2P_{i,2}^h)}{2C_{i,2}^h} - 1 = \frac{m_{i,1}^h(L_i + P_{i,2}^h)}{C_{i,2} - C_{i,2}^\ell} - 1 && \text{(From 6)} \\
&= \frac{m_{i,1}^h((T_i - P_{i,2}) + P_{i,2}^h)}{C_{i,2} - C_{i,2}^\ell} - 1 && \text{(From 2)} \\
&= \frac{m_{i,1}^h(T_i - (P_{i,2}^h + P_{i,2}^\ell) + P_{i,2}^h)}{C_{i,2} - C_{i,2}^\ell} - 1 && \text{(From 6)} \\
&= \frac{m_{i,1}^h(T_i - P_{i,2}^\ell)}{C_{i,2} - C_{i,2}^\ell} - 1
\end{aligned}
$$

Now putting the above value of $f_{i,1}^h$ in Equation 7, for any heavy segment $\langle e_{i,j}^h, m_{i,j}^h \rangle$, we get

$$
f_{i,j}^h = \frac{m_{i,j}^h(T_i - P_{i,2}^\ell)}{C_{i,2} - C_{i,2}^\ell} - 1 \tag{10}
$$

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

$$
\frac{m_{i,j}^h}{1 + f_{i,j}^h} = \frac{m_{i,j}^h}{1 + \frac{m_{i,j}^h(T_i - P_{i,2}^\ell)}{C_{i,2} - C_{i,2}^\ell} - 1} = \frac{C_{i,2} - C_{i,2}^\ell}{T_i - P_{i,2}^\ell} \tag{11}
$$

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

3.2.2 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,

$$
f_{i,j} = \frac{L_i}{P_{i,2}} \tag{12}
$$

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

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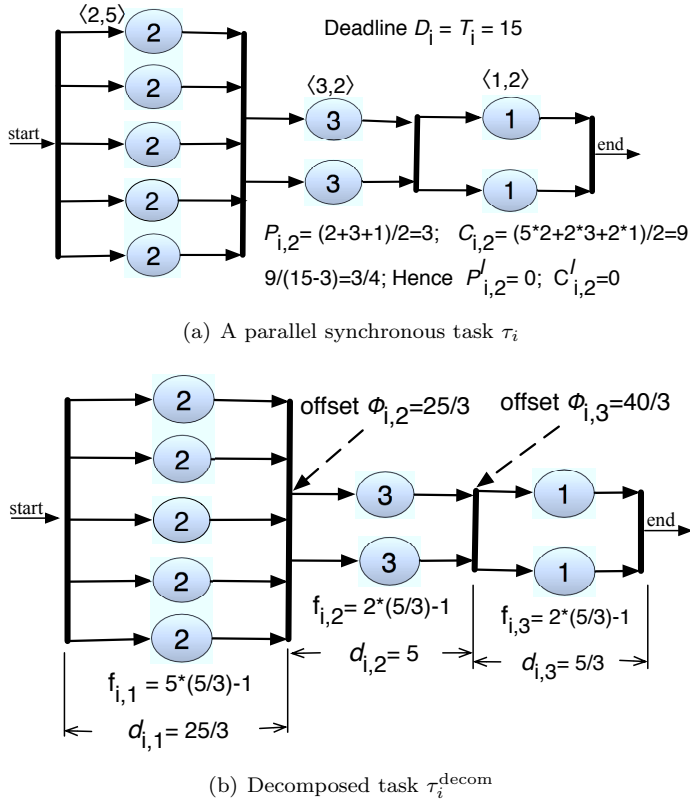$$

(a) A parallel synchronous task τ_i

(b) Decomposed task τ_i^{decom}

Fig. 3 An example of decomposition

3.3 Density Analysis

Once the above decomposition is done on task τ_i: $(\langle e_{i,1}, m_{i,1}\rangle, \cdots, \langle e_{i,s_i}, m_{i,s_i}\rangle)$, each thread of each segment $\langle e_{i,j}, m_{i,j}\rangle$, $1 \le j \le s_i$, 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 $f_{i,j} \ge 0$, $\forall 1 \le j \le s_i$, $\forall 1 \le i \le n$, the maximum density $\delta_{\mathrm{max},2}$ of any subtask (thread) among τ^{decom} on 2-speed processor core is

$$\delta_{\mathrm{max},2} = \max\{\frac{1}{1 + f_{i,j}}\} \le 1 \tag{14}$$

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

$$
\begin{aligned}
\frac{C_{i,2} - C_{i,2}^{\ell}}{T_i - P_{i,2}^{\ell}} &\leq \frac{C_{i,2}}{T_i - P_{i,2}^{\ell}} &&\text{(since } C_{i,2}^{\ell} \geq 0) \\
&\leq \frac{C_{i,2}}{T_i - P_{i,2}} &&\text{(since } P_{i,2} \geq P_{i,2}^{\ell}) \\
&= \frac{C_i/2}{T_i - P_i/2}
\end{aligned}
$$

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. $\square$

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, $\delta_{i,2} \leq \frac{C_i/2}{T_i - P_i/2}$. $\square$

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. □

4 Global EDF Scheduling

After our proposed decomposition, we consider the scheduling of synchronous parallel tasks. Lakshmanan et al (2010) 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, and conventional utilization bound approaches are not useful for schedulability analysis of parallel tasks. Hence, like Lakshmanan et al (2010), we use the *resource augmentation bound approach*, originally introduced by Funk et al (2001). 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 Earliest Deadline First (EDF) scheduling for them. The global 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 multi-core 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 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 processor cores proposed by Baruah (2007) as re-stated here in Theorem 4. This result is a generalization of the result for implicit deadline sporadic tasks (Goossens et al, 2003).

Theorem 4 *(Baruah, 2007) Any constrained deadline sporadic sequential task set π with total density $\delta_{\text{sum}}(\pi)$ and maximum density $\delta_{\text{max}}(\pi)$ is schedulable using global EDF strategy on m unit-speed processor cores if*

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

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

Theorem 5 *If there exists any way to schedule a synchronous parallel task set τ on m unit-speed processor cores, then the decomposed task set τ^{decom} is schedulable using global EDF on m processor cores each 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 (by 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}, suppose 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 $\delta_{\mathrm{sum},2\nu}$ and $\delta_{\mathrm{max},2\nu}$, respectively. From 14, we have

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

The value $\delta_{\mathrm{sum},2\nu}$ 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

$$\begin{aligned}
\delta_{i,2\nu} &\leq \frac{\frac{C_i}{2\nu}}{T_i - \frac{P_i}{2}} \leq \frac{\frac{C_i}{2\nu}}{T_i - \frac{T_i}{2}} && \text{(since } P_i \leq T_i\text{)} \\
&= \frac{\frac{C_i}{2\nu}}{\frac{T_i}{2}} = \frac{1}{\nu} \cdot \frac{C_i}{T_i}
\end{aligned}$$

Thus, from 15,

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

Note that, in the decomposed task set, every thread of the original task is a sequential task on a multiprocessor platform. Therefore, $\delta_{\mathrm{sum},2\nu}$ is the total density of all threads (i.e. subtasks), and $\delta_{\mathrm{max},2\nu}$ is the maximum density among all threads. Thus, by Theorem 4, the decomposed task set τ^{decom} is schedulable under global EDF on m processor cores each of speed 2ν if

$$\delta_{\mathrm{sum},2\nu} \leq m - (m-1)\delta_{\mathrm{max},2\nu} \tag{18}$$

Now using the values of $\delta_{\mathrm{sum},2\nu}$ (Equation 17) and $\delta_{\mathrm{max},2\nu}$ (Equation 16) into Condition (18), task set τ^{decom} is schedulable if

$$\begin{aligned}
&\frac{m}{\nu} \leq m - (m-1)\frac{1}{\nu} \\
\Leftrightarrow\ &\frac{1}{\nu} + \frac{1}{\nu} - \frac{1}{m\nu} \leq 1 \quad \Leftrightarrow \quad \frac{2}{\nu} - \frac{1}{m\nu} \leq 1
\end{aligned}$$

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

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

Hence follows the theorem.   □

5 Partitioned Deadline Monotonic Scheduling

Using the same decomposition described in Section 3, we now derive a re-
source augmentation bound required to schedule task sets under partitioned
Deadline Monotonic (DM) scheduling. Unlike global scheduling, in *partitioned
scheduling*, each task is assigned to a processor core. Tasks are executed only
on their assigned processor cores, and are not allowed to migrate among cores.
We consider the FBB-FFD (Fisher Baruah Baker - First-Fit Decreasing) par-
titioned DM scheduling proposed by Fisher et al (2006) which Lakshmanan
et al (2010) also uses as the scheduling strategy for parallel tasks in a more
restricted model. In fact, the FBB-FFD Algorithm was developed for peri-
odic tasks without release offsets while our decomposed subtasks have offsets.
Therefore, first we present how the FBB-FFD Algorithm should be adapted
to partition our subtasks with offsets, and then we analyze the resource aug-
mentation bound.

5.1 FBB-FFD based Partitioned DM Algorithm for Decomposed Tasks

The original FBB-FFD Algorithm by Fisher et al (2006) is a variant of the
first-fit decreasing bin-packing heuristic, and hinges on the notion of a request-
bound function for constrained deadline periodic sequential tasks. For a se-
quential task π_i with execution requirement e_i, utilization u_i, and deadline d_i,
its *request-bound function* $\mathrm{RBF}(\pi_i, t)$ for any time interval of length t is the
largest cumulative execution requirement of all jobs that can be generated by
π_i to have their arrival times within a contiguous interval of length t. In the
FBB-FFD Algorithm, $\mathrm{RBF}(\pi_i, t)$ is approximated as

$$\mathrm{RBF}^*(\pi_i, t) = e_i + u_i t$$

Let the processor cores be indexed as $1, 2, \cdots, m$, and Π_q be the set of
tasks already assigned to processor core q, where $1 \leq q \leq m$. Considering the
tasks in decreasing DM-priority order and starting from the highest priority
task, the FBB-FFD algorithm assigns a task π_i to the first processor core q,
$1 \leq q \leq m$, that satisfies the following condition

$$d_i - \sum_{\pi_j \in \Pi_q} \mathrm{RBF}^*(\pi_j, d_i) \geq e_i \tag{19}$$

If no processor core satisfies the above condition for some task, then the task
set is decided to be infeasible for partitioning.

When we adopt the FBB-FFD algorithm to partition our decomposed subtasks, we need to take the following two things into consideration while using Condition (19):

1. The subtasks of segment j should be both prioritized and partitioned only after the subtasks of segment $j-1$ are prioritized and partitioned (for the same original parallel task).
2. Ignore the interference from the subtasks in other segments in the same task.

5.2 Analysis for the FBB-FFD based Partitioned DM Algorithm

We use an analysis similar to the one used by Lakshmanan et al (2010) 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 by Baruah et al (1990), 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, a period of T_i, and a deadline of D_i, its DBF is given by

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

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

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

From Definition 5, for a constrained deadline task τ_i:

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

$$\leq \left\lfloor \frac{t}{D_i} \right\rfloor C_i \leq \frac{t}{D_i}.C_i = \delta_i.t$$

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

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

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

$$\lambda(\tau^{\mathrm{decom}}) = \max_{t>0} \left(\frac{\sum\limits_{i=1}^{n} \mathrm{DBF}(\tau_i^{\mathrm{decom}}, t)}{t} \right) \leq \sum_{i=1}^{n} \frac{C_i/2}{T_i - P_i/2} \qquad (20)$$

Theorem 6 *If there exists any (optimal) algorithm under which a synchronous parallel task set τ is schedulable on m unit-speed processor cores, then its decomposed task set τ^{decom} is schedulable using the FBB-FDD based partitioned DM Algorithm on m identical processor cores each of speed 5.*

Proof Fisher et al (2006) proves that any constrained deadline sporadic task set π with total utilization $u_{\mathrm{sum}}(\pi)$, maximum density $\delta_{\mathrm{max}}(\pi)$, and load $\lambda(\pi)$ is schedulable by the FBB-FFD Algorithm on m unit-speed processor cores if

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

Using the same method used by Fisher et al (2006) for proving the above sufficient schedulability condition, it can be shown that our decomposed (sub)tasks τ^{decom} are schedulable by the FBB-FFD based partitioned DM scheduling (presented in Subsection 5.1) on m unit-speed processor cores if

$$m \geq \frac{\lambda(\tau^{\mathrm{decom}}) + u_{\mathrm{sum}}(\tau^{\mathrm{decom}}) - \delta_{\mathrm{max}}(\tau^{\mathrm{decom}})}{1 - \delta_{\mathrm{max}}(\tau^{\mathrm{decom}})} \qquad (21)$$

where $\delta_{\mathrm{max}}(\tau^{\mathrm{decom}})$, $u_{\mathrm{sum}}(\tau^{\mathrm{decom}})$, and $\lambda(\tau^{\mathrm{decom}})$ denote the maximum density, total utilization, and load, respectively, of τ^{decom} on unit-speed processor cores.

We decompose tasks considering that each processor core has speed 2. To be able to schedule the decomposed tasks τ^{decom}, suppose 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 $\delta_{\mathrm{max},2\nu}$, $u_{\mathrm{sum},2\nu}$, and $\lambda_{2\nu}$ respectively, when each processor core has speed 2ν. Using these notations in Condition (21), task set τ^{decom} is schedulable by the FBB-FFD based partitioned DM Algorithm on m identical processor cores each of speed 2ν if

$$m \geq \frac{\lambda_{2\nu} + u_{\mathrm{sum},2\nu} - \delta_{\mathrm{max},2\nu}}{1 - \delta_{\mathrm{max},2\nu}} \qquad (22)$$

From Equation 1:

$$u_{\text{sum},2\nu} = \sum_{i=1}^{n} \frac{\frac{C_i}{2\nu}}{T_i} = \frac{1}{2\nu} \sum_{i=1}^{n} \frac{C_i}{T_i} = \frac{u_{\text{sum}}}{2\nu} \qquad (23)$$

From Equations 1 and 20:

$$\lambda_{2\nu} \leq \sum_{i=1}^{n} \frac{\frac{C_i}{2\nu}}{T_i - \frac{P_i}{2}} \leq \sum_{i=1}^{n} \frac{\frac{C_i}{2\nu}}{T_i - \frac{T_i}{2}} = \frac{1}{\nu} \sum_{i=1}^{n} \frac{C_i}{T_i} = \frac{u_{sum}}{\nu} \qquad (24)$$

Using Equations 24, 23, 16 in Condition (22), task set τ^{decom} is schedulable if

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

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

6 Generalizing to a Unit-node DAG Task Model

In the analysis presented so far, we have focused on 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 show 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 unit-node 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 unit-node 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

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

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
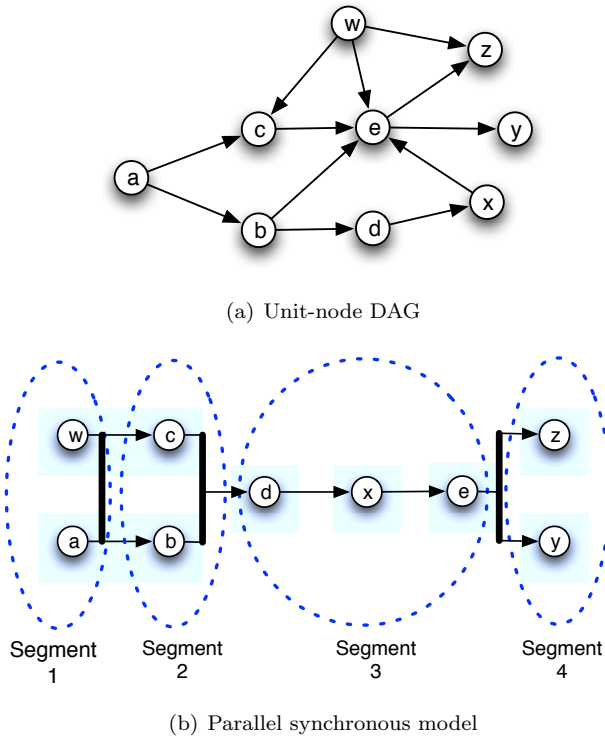
(a) Unit-node DAG

(b) Parallel synchronous model

Fig. 4 Unit-node DAG to parallel synchronous model

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 4 shows an example, where a DAG in Figure 4(a) is represented as a synchronous task in Figure 4(b). This transformation is valid since it preserves all dependences in the DAG, and in fact only adds extra dependences.

Upon representing a unit-node DAG task as a synchronous task, we perform the same decomposition proposed in Section 3. 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 $\sum C_i/T_i \leq m$ 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 unit-node DAG tasks can be scheduled with a resource augmentation bound of 4 under global EDF scheduling, and of 5 under partitioned DM scheduling.

7 Evaluation

In this section, we evaluate the proposed decomposition through simulations. We generate synchronous parallel tasks, decompose them, and simulate their schedules under global EDF and partitioned DM policies considering multi-core processors with different number of cores. We validate the derived resource augmentation bounds by considering different speeds of the processor cores.

In our simulation studies, parallel synchronous task sets are generated in the following way. The number of segments of each task is randomly selected from the range $[10, 30]$. The number of threads in each segment is randomly selected from the range $[1, 90]$. The execution requirements of the threads in a segment are selected randomly from the range $[5, 35]$. Each task is assigned a valid harmonic period (i.e. period is no less than its critical path length) of the form 2^k, where k is chosen from the range $[6, 13]$. We generate task sets considering $m = 20, 40$, and 80 (i.e. for 20-core, 40-core, and 80-core processors). In generating a task set for any particular value of m, we keep adding tasks to the set as long as their total utilization does not exceed m. For each value of m, we generate 1000 task sets. The tasks generated for a particular value of m are decomposed using our proposed decomposition technique. The decomposed tasks are then scheduled by varying the speed of the cores on the considered multi-core platform. We evaluate the performance in terms of a *failure ratio* defined as the proportion of the number of unschedulable task sets to the total number of task sets attempted. In Figures 5, 6, and 7, the failure ratios under global EDF (G-EDF) and partitioned DM (P-DM) based on the FBB-FFD Algorithm are labeled as G-EDF and P-DM, respectively.

In the first set of simulations, we evaluate the schedulability of 1000 task sets generated for a 20-core processor. The decomposed tasks are scheduled on a 20-core processor by varying the speed of all cores (all cores having the same speed). The failure ratios under different speeds are shown in Figure 5. In particular, we start by setting a speed of 1 (i.e. unit-speed) at every processor core. Then we keep increasing the speed of each core
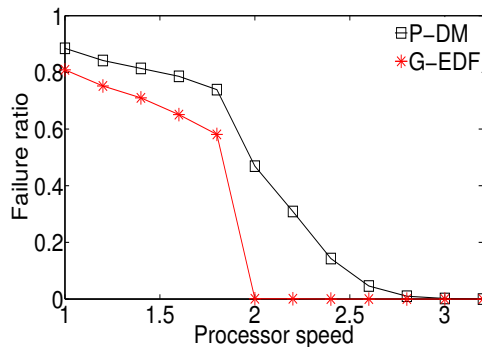

Fig. 5 Schedulability on a 20-core processor

by 0.2 in every step, and schedule the same set of tasks using the increased speed. When each core has speed 1, the failure ratio under G-EDF is 0.809. That is, out of 1000 test cases, 809 cases are not schedulable under G-EDF. As we gradually increase the speed of each core, the failure ratios decrease sharply. For example, when each core has speed of 1.2, 1.4, 1.6, and 1.8, the failure ratios are 0.752, 0.710, 0.651, and 0.581, respectively. When speed is increased

to 2 from 1.8, we observe a very sharp decrease in failure ratios. Specifically, at speeds 2 and 2.2, only one task set is unschedulable. This sharp decrease happens due to the following reason. According to our decomposition, when the speed of each core is less than 2, some (sub)task may have density greater than 1 meaning that it cannot meet its deadline at speed lower than 2. Since the decomposition guarantees that the maximum density among all (sub)tasks is at most 1 at speed 2, many task sets that were unschedulable at speed 1.8 become schedulable at speed 2. When each core has speed 2.4 or more, all task sets are schedulable. Thus, the resource augmentation required for the tasks we have evaluated under G-EDF is only 2.4 for this simulation setting.

Besides G-EDF, Figure 5 also plots the failure ratios under P-DM (based on the FBB-FFD Algorithm) on a 20-core processor. When each core has speed 1, the failure ratio under P-DM is 0.885 meaning that 885 test cases out of 1000 cases are unschedulable. With the increase in speed, the failure ratios decrease sharply, and at speed 2.4, it becomes only 0.143. After that, when the speed is increased to 2.6, 2.8, and 3, the failure ratios reduce to 0.046, 0.01, and 0.002, respectively. Although it decreases quite quickly with the increase in speed, it does not decrease as sharply as under G-EDF. In fact, at every speed up to 3, it is higher than that under G-EDF which is quite reasonable to expect since G-EDF is a highly efficient scheduling policy compared to P-DM. When the speed of each core is increased to 3.2 or more, all 1000 task sets are schedulable under P-DM, demonstrating a resource augmentation of 3.2 for this specific simulation setting which is smaller than our theoretical bound of 5 for P-DM.

In the second set of simulations, we evaluate the schedulability of 1000 task sets generated for a 40-core processor. Note that these task sets are different from those generated for the previous set of simulations, and have higher total utilization (as these are generated for a higher number of cores). We schedule the decomposed tasks on a 40-core processor, and show the failure ratios under different speeds in Figure 6. Here also, we start by
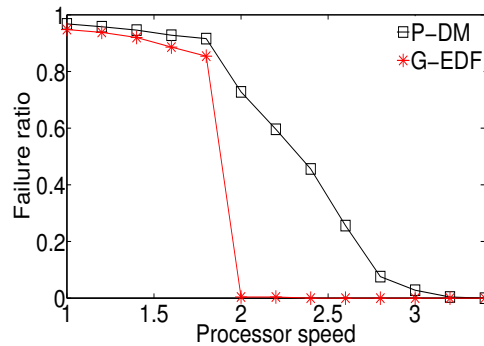

Fig. 6 Schedulability on a 40-core processor

setting a speed of 1 of every processor core. As the figure shows, when each core has speed 1, the failure ratio under G-EDF is 0.948. With the increase in speed, it keeps decreasing, and becomes 0.854 at a speed of 1.8. Similar to the previous experiment, when speed is increased to 2 from 1.8, the failure ratio sharply decreases (to 0.004). At a speed of 2.4 or more, all 1000 task sets are schedulable under G-EDF on a 40-core processor. Similar to our first set of simulations, these results also show that the required resource augmentation

under G-EDF is 2.4 for this simulation setting. Under P-DM, the failure ratio is 0.968 at speed 1, and decreases as the speed increases, and becomes 0.004 at speed 3.2. When the speed is increased to 3.4 or more, all task sets become schedulable under P-DM, demonstrating a resource augmentation of 3.4 for this simulation setting.

Finally, we evaluate using 1000 task sets generated for a 80-core processor, and show the failure rates under different speeds in Figure 7. No task set is schedulable under G-EDF at a speed of 1.6 or less on a 80-core processor. At a speed of 1.8, the failure ratio under G-EDF is 0.996, while all tasks become schedulable at a speed of 2.2. The results show that the required resource augmen-tation under G-EDF is only 2.2 for this simulation setting. For
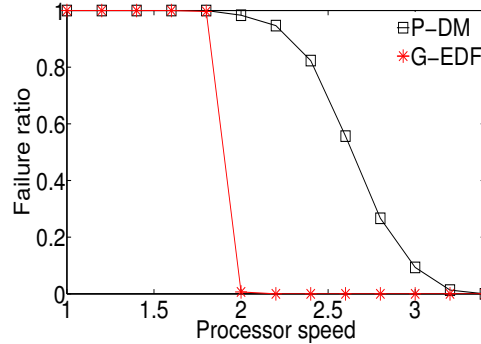

Fig. 7 Schedulability on a 80-core processor

scheduling under P-DM, no task set is schedulable at a speed of 1.8 or less. The failure ratios under P-DM keeps decreasing as we increase the speed, and becomes 0.013 at speed 3.2. When the speed of each core is increased to 3.4 or more, all task sets become schedulable. Thus, in this experiment, the required resource augmentation is at most 3.4 for P-DM.

The above results indicate that the decomposed tasks we evaluated re-quired a resource augmentation of at most 2.4 under global EDF. This bound is much smaller than the theoretical bound of 4. Similarly, the results show that the decomposed tasks we evaluated required a resource augmentation of at most 3.4 for partitioned DM scheduling, which is also much smaller than the theoretical bound of 5. The results thus demonstrate that the analytical augmentation bounds are safe and sufficient.

8 Related Work

There has been extensive work on traditional multiprocessor real-time schedul-ing (Davis and Burns, 2011). Most of this work focuses on scheduling sequential tasks on multiprocessor or multi-core systems. There has also been extensive work on scheduling of one or more parallel jobs on multiprocessors (Poly-chronopoulos and Kuck, 1987; Drozdowski, 1996; Deng et al, 1996; Arora et al, 1998; Bansal et al, 2004; Edmonds et al, 2003; Agrawal et al, 2006; Calandrino and Anderson, 2008; Calandrino et al, 2007a,b). However, the work in Poly-chronopoulos and Kuck (1987); Drozdowski (1996); Deng et al (1996); Arora et al (1998); Bansal et al (2004); Edmonds et al (2003); Agrawal et al (2006) does not consider task deadlines, and that in Calandrino and Anderson (2008);

Calandrino et al (2007a,b) considers soft real-time scheduling. In contrast to the goal of a hard real-time system (i.e. to meet all task deadlines), 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 and Calandrino (2006) proposes the concept of *megatask* as a way to reduce miss rates in shared caches on multi-core 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 and Lin (1989). Kwon and Chwa (1999) explores preemptive EDF scheduling of parallel task systems with linear-speedup parallelism. Wang and Cheng (1992) considers a heuristic for nonpreemptive scheduling. However, this work focuses on metrics like makespan (Wang and Cheng, 1992) or total work that meets deadline (Kwon and Chwa, 1999), 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 (2004), Lee and Lee (2006), and Collette et al (2008) 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 (1998) studies non-preemptive EDF scheduling for *moldable tasks*, where the actual number of used processors is determined before starting the system and remains unchanged. Kato and Ishikawa (2009) addresses 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 (2010) on real-time scheduling for a restrictive synchronous parallel task model. 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 analyze 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 the task period, and assign one processor core exclusively to it. The remaining threads are scheduled using the 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.

9 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. Then we have 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. Evaluation through simulation studies has validated that the derived resource augmentation bounds are safe and sufficient.

In the future, we plan to consider even more general DAG tasks where nodes have arbitrary execution requirements, and to provide analysis requiring no transformation to synchronous model. We also plan to address system issues such as cache effects, preemption penalties, and resource contention.

References

Agrawal K, He Y, Hsu WJ, Leiserson CE (2006) Adaptive task scheduling with parallelism feedback. In: PPoPP '06: Proceedings of the 11th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming, pp 100–109

Anderson JH, Calandrino JM (2006) Parallel real-time task scheduling on multicore platforms. In: RTSS '06: Proceedings of the 27th IEEE Real-Time Systems Symposium, pp 89–100

Arora NS, Blumofe RD, Plaxton CG (1998) Thread scheduling for multiprogrammed multiprocessors. In: SPAA '98: Proceedings of the 10th annual ACM Symposium on Parallel Algorithms and Architectures, pp 119–129

Bansal N, Dhamdhere K, Konemann J, Sinha A (2004) Non-clairvoyant scheduling for minimizing mean slowdown. Algorithmica 40(4):305–318

Baruah S (2007) Techniques for multiprocessor global schedulability analysis. In: RTSS '07: Proceedings of the 28th IEEE Real-Time Systems Symposium, pp 119–128

Baruah S, Mok A, Rosier L (1990) Preemptively scheduling hard-real-time sporadic tasks on one processor. In: RTSS '90: Proceedings of the 11th IEEE Real-Time Systems Symposium, pp 182–190

Calandrino JM, Anderson JH (2008) Cache-aware real-time scheduling on multicore platforms: Heuristics and a case study. In: ECRTS '08: Proceedings of the 20th Euromicro Conference on Real-Time Systems, pp 299–308

Calandrino JM, Anderson JH, Baumberger DP (2007a) A hybrid real-time scheduling approach for large-scale multicore platforms. In: ECRTS '07: Proceedings of the 19th Euromicro Conference on Real-Time Systems, pp 247–258

Calandrino JM, Baumberger D, Li T, Hahn S, Anderson JH (2007b) Soft real-time scheduling on performance asymmetric multicore platforms. In: RTAS '07: Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium, pp 101–112

ClearSpeed (2008) CoSy compiler for 96-core multi-threaded array processor. `http://www.clearspeed.com/newsevents/news/ClearSpeed_Ace_011708.php`

Collette S, Cucu L, Goossens J (2008) Integrating job parallelism in real-time scheduling theory. Information Processing Letter 106(5):180–187

Davis RI, Burns A (2011) A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems. ACM Computing Survey 43(4):35:1–44

Deng X, Gu N, Brecht T, Lu K (1996) Preemptive scheduling of parallel jobs on multiprocessors. In: SODA '96: Proceedings of the 7th annual ACM-SIAM Symposium on Discrete Algorithms, pp 159–167

Drozdowski M (1996) Real-time scheduling of linear speedup parallel tasks. Information Processing Letter 57(1):35–40

Edmonds J, Chinn DD, Brecht T, Deng X (2003) Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. Journal of Scheduling 6(3):231–250

Fisher N, Baruah S, Baker TP (2006) The partitioned scheduling of sporadic tasks according to static-priorities. In: ECRTS '06: Proceedings of the18th Euromicro Conference on Real-Time Systems, pp 118–127

Funk S, Goossens J, Baruah S (2001) On-line scheduling on uniform multiprocessors. In: RTSS '01: Proceedings of the 22nd IEEE Real-Time Systems Symposium, pp 183–192

Goossens J, Funk S, Baruah S (2003) Priority-driven scheduling of periodic task systems on multiprocessors. Real-Time Systems 25(2-3):187–205

Han CC, Lin KJ (1989) Scheduling parallelizable jobs on multiprocessors. In: RTSS '89: Proceedings of the 10th IEEE Real-Time Systems Symposium, pp 59–67

Huang HM, Tidwell T, Gill C, Lu C, Gao X, Dyke S (2010) Cyber-physical systems for real-time hybrid structural testing: a case study. In: ICCPS '10: Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems, pp 69–78

Intel (2007) Teraflops research chip. `http://techresearch.intel.com/ProjectDetails.aspx?Id=151`

Intel (2010) Cilk Plus. `http://software.intel.com/en-us/articles/intel-cilk-plus`

Jansen K (2004) Scheduling malleable parallel tasks: An asymptotic fully polynomial time approximation scheme. Algorithmica 39(1):59–81

Kato S, Ishikawa Y (2009) Gang EDF scheduling of parallel task systems. In: RTSS '09: Proceedings of the 30th IEEE Real-Time Systems Symposium, pp 459–468

Kwon OH, Chwa KY (1999) Scheduling parallel tasks with individual deadlines. Theoretical Computer Science 215(1-2):209–223

Lakshmanan K, Kato S, Rajkumar RR (2010) Scheduling parallel real-time tasks on multicore processors. In: RTSS '10: Proceedings of the 30th IEEE Real-Time Systems Symposium, pp 259–268

Lee WY, Lee H (2006) Optimal scheduling for real-time parallel tasks. IEICE Transactions on Information and Systems E89-D(6):1962–1966

Manimaran G, Murthy CSR, Ramamritham K (1998) A new approach for scheduling of parallelizable tasks inreal-time multiprocessor systems. Real-Time Systems 15(1):39–60

OpenMP (2011) OpenMP: Open multi-processing. `http://openmp.org`

Polychronopoulos CD, Kuck DJ (1987) Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. IEEE Transactions on Computers C-36(12):1425–1439

Wang Q, Cheng KH (1992) A heuristic of scheduling parallel tasks and its analysis. SIAM Journal of Computing 21(2):281–294