Abstract—Nowadays, the multi-core platforms have become the de-facto solution to cope with the rapid increase of system complexity and energy consumption. Additionally, the dynamic power management (DPM) and the dynamic voltage and frequency scaling (DVS) are two well-established techniques to adjust the trade-off between the system performance and power consumption during runtime. However, in the context of hard real-time systems the DPM and DVS have to be applied with great caution due to timing constraints. The problem of DPM/DVS based power-aware scheduling has been extensively addressed on single-core platforms. Therefore some recent studies have proposed to adapt the existing results to multi-core platforms by performing the task partition in advance. In this article, we show that this approach may not work correctly any more, if the cluster-based multi-core platforms with non-negligible DPM and DVS state switching overhead are considered. More specifically, additional delays are introduced into the task execution and thus the traditional schedulability analysis becomes insufficient. We propose a simple runtime mechanism for idle time prediction to deal with the DPM state switching overhead and two solutions to enhance the schedulability analysis by taking the DVS state switching overhead into consideration: the conservative protocol and the speed inheritance protocol. Finally, the solutions are evaluated by means of simulation.

Keywords—Dynamic Power Management; Dynamic Voltage and Frequency Scaling; Hard Real-Time Systems;

I. INTRODUCTION

In modern electronic systems, energy efficiency becomes one of the most important design concerns due to the growing requirements for low power consumption, long battery life and low heat dissipation. The Dynamic Power Management (DPM) and the Dynamic Voltage and Frequency Scaling (DVS) are two well-established system-level techniques to adjust the trade-off between the system performance and power consumption during runtime. In general, the main idea behind DPM is to switch the components to a sleep or low power state when they are idle. Hereby, due to the fact that the switching process from the active state to the sleep state usually consumes both time and energy, the mindless switching can not always ensure the energy saving or might even jeopardize the task deadline in a hard real-time system. Therefore, the concept of break even time introduced by [1] has been widely adopted to capture this issue. On the other side, the basic idea of DVS is to slow down the active components by lowering the operating speed and voltage. Obviously, hereby the switching from one speed/voltage state to another consumes time and energy as well. The voltage scaling is fairly expensive and is at least in the same order of magnitude as DPM state switching. For instance, the voltage switching on Intel XScale [2] processor typically takes 450µs and the shutdown to the deep-sleep mode takes 600µs. However, in the literatures the problem of DVS state switching is rarely studied in depth. Most work either ignore this overhead or simply account it to the task worst case execution time. In this article, we will study its impact on the cluster-based multi-core platforms.

Depending on the hardware support, the DPM/DVS capable chip multi-core processor platforms can be classified into three categories: full-chip platforms, per-core platforms and cluster-based platforms. They mainly differ in the levels, at which the DPM and DVS are applied. On the full-chip platforms the entire processor chip shares the same power supply net and therefore all cores can only operate at a common frequency at the same time. However, they can be shut down independently (e.g. Intel Core™2 Quad [3]), since switching off some cores does not affect other cores and only requires only a few transistors. With other words, the DVS is applied for the whole chip and the DPM can be applied on the individual cores. Obviously these platforms lack the flexibility for power management. With the introduction of Frequency/Voltage Island technique and on-chip voltage regulator, per-core platforms (e.g. AMD Phenom™ Quad-Core [4]) gain more and more interests. Hereby the difference to the full-chip platforms is that the DVS can be applied on the individual cores as well. Clearly, this type of platforms offers the most convenience but has to suffer the high implementation costs, because each core needs a dedicated on-chip voltage regulator. It might become impractical, if the number of cores dramatically increases as in the case of many-core processor platforms. In this article, our focus is on the cluster-based multi-core platforms, which is a generalized form of the full-chip and per-core platforms and provides the best compromise. The cores are divided into clusters. The cores from the same cluster behave like in the full-chip platforms and the cores from different clusters behave like in the per-core platforms. Additionally, we concentrate on the partitioned real-time scheduling, where all jobs instances of a periodic task can only be executed on its allocated core after the partitioning process.

The DPM/DVS based power-aware real-time scheduling on single-core platforms has been an extensively discussed research area in the last decade. According to some recent studies [5] [6], those single-core scheduling algorithms can be adapted to the multi-core platforms by additionally performing the task partitioning in advance, provided that the partitioned scheduling is applied. Hereby one of the major advantages is that the well-established single-core real-time scheduling algorithms like Earliest Deadline First (EDF) and Rate Monotonic (RM) can be adopted for each core. In this way, the traditional schedulability test via utilization can be performed for each core independently. However, if the cluster-based or full-chip multi-core platforms with non-negligible DPM and DVS state switching overhead are considered, this adaptation may not work appropriately any more, not only because of the reduced energy efficiency, but also due to the violation of timing constrains. As will be described in the Section IV, additional delays are introduced into the task execution, which makes the traditional schedulability analysis insufficient. In this article, our main contribution is not focused on proposing a specific power-aware real-time scheduling algorithm, but rather on studying the impact of non-negligible DPM and DVS state switching overhead.
on the schedulability analysis on cluster-based multi-core platforms. We propose a simple runtime mechanism for idle time prediction to handle DPM state switching overhead and two solutions to enhance the schedulability analysis by taking DVS state switching overhead into consideration: the conservative protocol and the speed inheritance protocol.

The remainder of this article is organized as follows. The Section II gives an overview of the related work. In the Section III we introduce the preliminaries, based on which the problem and the solution are presented. Afterwards, the Section IV describes the problems by means of several motivation examples and in the Section V we propose the solutions. Finally, before we conclude our work in the Section VII, the experiment results are presented in the Section VI.

II. RELATED WORK

In the context of single-core systems there exist extensive research work dealing with the energy aware real-time scheduling problem. [7] proposed a rate-harmonized scheduling algorithm for energy saving by means of delaying ready tasks to prolong the idle interval. Their approach was focused only on DPM techniques. On the contrary, in [8] the DVS has been applied to increase the energy efficiency with regard to non-preemptive real-time systems. Moreover, in order to further boost the energy saving by using DVS, [9] proposed a collaborative approach based on compiler and operating system support to consider application specific runtime information.

In the domain of full-chip platforms, most works are trying to balance the task partition. The authors in [10] proposed the dynamic repartitioning algorithm based on existing partition, which tries to balance the task load on different cores by considering the dynamic slack time. A dynamic core scaling scheme was proposed as well to reduce the number of unused cores. [11] proposed an approximation algorithm to schedule a set of frame-based tasks, which is a very restricted task model, because all tasks share a common deadline/period. In addition, the DPM and DVS state switching overhead are ignored in these work. Furthermore, [5] shows the \( \mathcal{NP} \)-hardness of the problem of task partitioning with energy minimization and proposed a load balancing framework based on per-core platforms. The authors of [12] described a per-core DVS algorithm taking different power characteristics of tasks into account. They considered the frame-based task model and ideal processor, which can operate at any speed within a given range. With regard to the leakage power, several approximation algorithms are provided in [6] on the basis of per-core platforms, but the analysis is based on the ideal processor model as well. [13] classified the tasks into three categories according to their utilization and proposed a heuristic scheduling scheme based on the classification. Their focus is on the lightly loaded multi-core platforms. Only recently the cluster-based multi-core platforms attracts more and more attention. [14] has focused on the problem of clustering the cores into DVS domains. The authors proposed to group similar cores into cluster based on their typical workload. [15] introduced a fundamentally alternate mean for cluster-based multi-core processor design. They believe that a core, which is designed for a dedicated frequency/voltage domain, is more energy efficient than a core designed with runtime DVS capabilities and configured to that frequency/voltage domain. These both work are not focused on hard real-time systems. More closely related work are presented in [16] and [17]. Both of them are based on the cluster-based multi-core processor and concentrate on hard real-time systems, however, they both assumed frame-based task model and ideal processor model.

To sum up, none of the above mentioned work has studied the non-negligible state switching overhead in depth. This article addresses the schedulability analysis problem on DPM/DVS enabled cluster-based multi-core platforms, especially by taking state switching overhead into consideration. Moreover, our work will consider a more general task model and non-ideal processor model, which are more common in the reality.

III. PRELIMINARIES

This section shows the basic terminology to specify the power model of a cluster-based multi-core platform and the task model of a hard real-time system.

A. Cluster-Based Multi-Core Platforms

We follow the Advanced Configuration & Power Interface (ACPI) specification to describe the power model of a cluster-based multi-core platform. More specifically, we adopt the concept of C-states and P-states to describe the DPM and DVS states of individual cores, respectively. The C-states describe the different power states including one active state and multiple low power (sleep) states with different sleep depth. The P-states reveal different performance states when the processor core is active and mainly differ in the operating speed/voltage and power consumption. The power model is defined for each core \( D_{x,y} \in \mathcal{D} \), where \( x \) is the cluster index and \( y \) is the core index. We assume that the cores in the same cluster share the same power model and the cores from different clusters could possess different power models. For each core we denote a set of power states (C-states) as \( C_0, C_1, ..., C_e \) where \( C_0 \) is the only active state and \( C_1, ..., C_e \) are low power states in non-increasing order of power consumption.

The active state \( C_0 \) contains a set of P-states denoted by \( S_1, S_2, ..., S_s \), where \( S_1 \) is the full performance state and the remaining states are arranged in non-increasing order of power consumption. \( \forall i: 1 \leq i \leq s \) we denote \( F(S_i) \) and \( P(S_i) \) as the frequency and power consumption of the state \( S_i \), respectively. Furthermore, the latency and the power consumption of the switching from one P-state to another P-state are denoted by \( L_p \) and \( P_p \), respectively. If the switching overhead among different P-states are different, we assume that the \( L_p \) and \( P_p \) denote the worst case.

B. Real-Time Task Model

Since we are interested in the hard real-time systems with independent periodic tasks, we adopt the traditional real-time task model. The task set is denoted by \( \Gamma = \{\tau_1, \tau_2, ..., \tau_n\} \) with \( \forall i: 1 \leq i \leq n, \tau_i = (W_i, T_i) \) where \( W_i \) denotes the Worst Case Execution Cycles and \( T_i \) is the relative deadline (equal to the period) of the task. Furthermore, we assume that the task execution time is linear to the operating speed.

IV. PROBLEM DESCRIPTION

The actual energy optimization problem of a real-time system on multi-core platforms is composed of three parts: partitioning the tasks, assigning the speeds to the tasks and developing a scheduling algorithm on each core. In order to find the optimal solution, these parts usually have to be considered coherently, because they have influence on each other. Since the partitioned scheduling is considered, most work in the literatures use existing single-core real-time schedules (e.g. EDF or RM) to solve the third part. Therefore, a low power solution usually contains two functions: assign : \( \Gamma \rightarrow D \) and assign : \( \Gamma \rightarrow S \), which denote the task partition and the speed assignment, respectively. The system feasibility is then checked by performing the schedulability analysis on each core independently. With other words, a solution can schedule all tasks without any deadline miss, if the following condition holds for each core \( D_{x,y} \):
platform is schedulable, if the following condition is satisfied:

**Lemma 1.**

\[ \sum_{alloc(\tau_i) = D_{x,y}} W_i \frac{F(\text{assign}(\tau_i))}{T_{i}} \leq U_{ub} \tag{1} \]

The \( U_{ub} \) is the utilization upper bound, e.g. by EDF \( U_{ub} = 1 \) and by RM \( U_{ub} = 0.69 \). In the remaining text, if there is no explicit specification, we always assume that the EDF schedule is applied. The main scope of this article is not focused on proposing any algorithm to compute the optimal low power solution, but rather on analysing the schedulability problem. More concretely, in this section we will demonstrate by several motivation examples that the traditional schedulability test (condition (1)) is not sufficient any more, if the cluster-based multi-core platforms with non-negligible C-state and P-state switching overhead are addressed.

### A. C-state Switching Problem

Since the DPM technique is applied on each core independently, the problem of dealing with the C-state switching overhead on multicore platforms can be considered for each core separately. On single-core platforms, this problem is well known. According to [1], if the overhead is not handled properly, the processor may consume more energy or even has to delay the ready tasks, which might lead to a deadline miss.

### B. P-state Switching Problem on Single-Core Platforms

In order to make the life more easier, we first address the problem on single-core platforms in this subsection and then the problem on multi-core platforms will be discussed in the next subsection. On single-core platforms the problem caused by the P-state switching is quite straightforward. Since each task runs at its assigned speed, at each task switch point the core operating speed may have to be changed. Clearly, the switching latency introduces additional delays into task execution. If this latency is not considered properly at each task switch point the core operating speed may have to be changed. Clearly, the switching latency introduces additional delays into task execution. If this latency is not handled properly, the processor may consume more energy or even has to delay the ready tasks, which might lead to a deadline miss.

### C. P-state Switching Problem on Cluster-Based Multi-Core Platforms

The Fig. 1 shows a simple example with two cores grouped into one cluster. Both cores support 50 MHz and 100 MHz with the switching latency of 5 ms. Moreover, there are two real-time tasks, which are partitioned to the first core and the second core, respectively. The task \( \tau_1 \) is assigned with 50 MHz and the task \( \tau_2 \) is assigned with 100 MHz. We observe that the tasks \( \tau_1 \) and \( \tau_2 \) have a speed conflict during the time interval between 0 ms and 5 ms, because they run in parallel but require different operating speeds. Our strategy is to select the highest speed as the cluster-wide operating speed, since in this way we can guarantee that no task will finish later than its worst case finishing time. On some platforms this strategy is even obliged due to the hardware support, such as Intel Core™2 Quad [3]. If we apply the schedulability test from the Lemma 1, the utilization for each core \( U_{D_{1,1}} = \frac{9}{10} \) and \( U_{D_{1,2}} = \frac{3}{4} \) can be computed respectively.

![Fig. 1. An example with two tasks running on a platform with two cores grouped into one cluster. Both cores support 50 MHz and 100 MHz with the switching latency of 5 ms.](image)

**Definition 1.** An Inter-Core Preemption of a task \( \tau_i \) is a preemption of its execution due to core speed change, which is caused by the arrival or the completion of another task \( \tau_j \) with the conditions \( alloc(\tau_i) \neq alloc(\tau_j) \).

In summary, as seen in the examples, the traditional schedulability analysis is not sufficient any more for cluster-based platforms with non-negligible C- and P-state switching overhead.

### V. ENHANCED SCHEDULABILITY ANALYSIS

In this section we propose the solution by first dealing with the problem of C-state switching overhead. Then, we will address the problem of P-state switching overhead.

### A. C-state Switching Overhead

Benini et al. [1] introduced the break even time to capture the idle time that is needed at least to compensate the wasted energy and time during the switching. Therefore, the main challenge here is to predict the length of the idle intervals, based on which the decision is made, whether the core can be switched off to a specific low power state. For this, we propose a simple runtime mechanism shown in the Algorithm 1.

The Algorithm 1 is invoked at each scheduling point and mainly maintains a sorted queue \( Q \) storing the arrival time of each task allocated on the current core. The \( Q \) is initialized with \( t_{i,\text{next}} = 0 \) for all tasks at the system start. In this way we can easily compute the length of next idle interval, which is equal to \( t - t_{\text{current}} \). Following this information a proper low power state can be selected easily as well. Since the \( Q \) is always kept in a sorted manner, each queue operation (i.e. insertion, search) can be completed in \( O(\log n) \) time.
using binary search, where the $n$ is the number of the tasks in the system. Hereby we assume that the number of the available low power states is relatively small and can be treated as a constant. Therefore, the Algorithm 1 has a complexity of $O(\log n)$. Moreover, according to the line 11 we can ensure that the processor core is always waked up before the task is ready. As a consequence, no task will be delayed due to the state switching latency and the traditional schedulability test can still be used.

B. P-state switching overhead

Now we present the solutions to enhance the schedulability analysis by taking the non-negligible P-state switching overhead into consideration.

1) Conservative Protocol: As mentioned earlier in the Section IV, the main cause of the deadline miss is due to the additional delays introduced by the ICPs. Therefore the most straightforward idea is to estimate the maximal number of such delays for each task and account them into the task worst case execution time. Formally, we propose the Theorem 1.

**Theorem 1.** Given a cluster-based multi-core processor platform and a set of independent periodic real-time tasks, a low power solution containing a task partition and a speed assignment can guarantee the system feasibility, if the following condition for each core $D_{x,y}$ holds:

$$\sum_{\text{alloc}(\tau_i)=D_{x,y}} \frac{W_i}{P_{\text{assign}(\tau_i)}} + 2 * L_p + \text{num}_i * L_p \leq U_{ub} \quad (3)$$

where $\text{num}_i$ is the maximal number of delays that can be introduced into the execution of the task $\tau_i$ due to ICPs. Formally it is computed by the Equation 4.

$$\text{num}_i = \sum_{\tau_j} \left[ \frac{T_i}{T_j} \right] * 2, \tau_j \text{ satisfies cond. in Definition 1} \quad (4)$$

**Proof:** We first show the correctness of the Equation 4, which estimates the maximal number of delays for each task $\tau_i$. Clearly, only the tasks satisfying the conditions in Definition 1 may generate the ICPs for $\tau_i$. Additionally, it is obvious that the number of the ICPs generated by a task $\tau_i$ is limited by the number of its occurrence during the period of the task $\tau_i$. Thus, for each task we compute the maximal number of its occurrence by the term $\frac{T_i}{T_j}$. Since each ICP may cause two delays at most (as shown in the Fig. 1), once at the beginning and once at the end, the maximal number of delays is equal to the number of occurrence multiplied with 2. Furthermore, in order to cover the problem that already exists on the single-core platforms, we apply the Lemma 1. As a result, the sufficient condition for the schedulability analysis is obtained in the Equation 3.

![Algorithm 1 C-state switching overhead handling](image)

**Algorithm 1** C-state switching overhead handling

1: if exists a $\tau_i$ that is finishing then
2: \hspace{1em} Get the arrival time $t_{i,next}$ of the next job instance of task $\tau_i$ by $t_{i,next} = t_{i,\text{next}} + T_i$
3: \hspace{1em} Reinsert $t_{i,next}$ into the sorted queue $Q$
4: end if
5: if there is no ready task then
6: \hspace{1em} Get the next task arrival time by finding the smallest time stamp $t$ in $Q$, which is larger than the current time stamp $t_{\text{current}}$. Select the lowest power state with the break even time larger than the idle interval.
7: end if

Even though we have derived a sufficient schedulability test, it is not hard to see that this test is too pessimistic. It is not necessary that each occurrence will generate an ICP. Many solutions may fail the test, even they are indeed schedulable. This is also the reason why this approach is called conservative protocol. However, one of the advantages by this protocol is that there is no need to modify the task execution. In the next step a more sophisticated approach is presented, where the test is much more tighter.

2) Speed Inheritance Protocol: The speed inheritance protocol is based on an important observation. Each time a task is interrupted by an ICP, its operating speed is usually increased as well. Therefore, the task will also run faster. We can utilize this property to compensate the delays caused by the ICP. This idea is similar to the DPM break event time. In this way the delays do not need to be accounted into the task worst case execution time any more. Before we present the speed inheritance protocol in detail, several definitions are introduced.

**Definition 2.** The $EFC_i(t)$ is defined as the Expected Finished Cycles of the task $\tau_i$ until the time point $t$. This describes the finished cycles of a task until $t$ in the expected case where no ICP happens.

**Definition 3.** The $AFC_i(t)$ is defined as the Actual Finished Cycles of the task $\tau_i$ until the time point $t$. This describes the finished cycles of a task until $t$ in the actual case of task execution.

The Fig. 2 shows the execution of a task $\tau_i$ under the expected case and the actual case, respectively. The expected case reflects the task execution that has been planned. If all tasks are executed as expected, then no task will miss its deadline. However, due to the ICP the task execution may vary in the actual case. For instance, at the time point $t_2$ in the Fig. 2 $EFC_i(t_2) = AFC_i(t_2)$ clearly holds, because no delay has been introduced before $t_2$. However, at $t_2$ the $AFC_i(t_2)$ is already behind the $EFC_i(t_2)$, as no work has been done during the interval between $t_1$ and $t_2$ in the actual case. The main idea here is to compute the time $t_{be}$ needed at least to compensate the delays. With other words, the task $\tau_i$ needs to run at the higher speed $S_1$ for at least $t_{be}$, so that the actual case execution can catch up the progress in the expected case. Formally, the Equation 5 is to be ensured.

$$AFC_i(t_3) = EFC_i(t_3) \quad (5)$$

Moreover, the Equations 6 and 7 obviously hold .

$$AFC_i(t_3) = AFC_i(t_2) + t_{be} * F(S_1) \quad (6)$$

$$EFC_i(t_3) = EFC_i(t_2) + (t_{be} + L_p) * F(S_2) \quad (7)$$

By applying the Equations 6 and 7 into the Equation 5, we obtain the formula to compute the break even time $t_{be}$.

$$t_{be} = \frac{EFC_i(t_2) - AFC_i(t_2) + L_p * F(S_2)}{F(S_1) - F(S_2)} \quad (8)$$

The speed inheritance protocol does not modify the actual scheduling algorithm, but rather has the impact on the selection of the task
execution speed. Now we describe the additional effort that has to be taken during runtime to enable the speed inheritance protocol. The AFC and EFC values for each task are initialized with 0 at the task start and updated at each scheduling point. Once a task is interrupted by an ICP, the \( t_{be} \) needs to be computed by means of the Equation 8. The system ensures that this task runs at the higher speed at least for \( t_{be} \). Afterwards the task can run at its originally assigned speed again. If an intra-core preemption occurs during this time, which is caused by the task with higher priority from the same processor core, then the system remembers the time that has been consumed and the remaining time will be consumed after the task is resumed. However, if another ICP happens and the task operating speed is increased again, then we need to re-calculate the \( t_{be} \) and ensure the task runs at the new speed for \( t_{be} \). Before we derive the schedulability test, the Lemma 2 shows one important feature of the \( EFC_i(t) \) and \( AFC_i(t) \) values for each task \( \tau_i \), i.e. the maximal cycles that the \( AFC_i \) value may lag behind the \( EFC_i \) value is limited.

**Lemma 2.** For any task \( \tau_i \) at any time point \( t \), the following condition holds:

\[
EFC_i(t) - AFC_i(t) \leq num'_i \ast L_p \ast F(assign(\tau_i)) \tag{9}
\]

where

\[
num'_i = |\{S_j|F(S_j) > F(assign(\tau_i)) \text{ and } F(S_j) \leq F(S_i)\}|
\]

\[
(10)
\]

**Proof:** At the task start, the \( EFC_i \) and \( AFC_i \) values are clearly equal, since they are initialized with 0. This equilibrium will not be broken, unless an ICP occurs. The curves in the Fig. 3 illustrate the value change of \( EFC_i \) and \( AFC_i \) based on the example shown in the Fig. 2. Obviously, at the time point \( t_2 \) the \( AFC_i \) lags behind the \( EFC_i \) the most. Afterwards the \( EFC_i \) starts to catch up the \( EFC_i \), because the task \( \tau_i \) runs now at a higher speed. In order to find the maximal value of \( EFC_i - AFC_i \), we need to consider the worst case, i.e. several ICPs happen consecutively. For instance in the Fig. 2, if at \( t_2 \) a third task interrupts the task \( \tau_i \) due to ICP and requires an even higher speed, then the distance between the \( EFC_i \) curve and the \( AFC_i \) curve will become larger. However, since each of these ICPs for \( \tau_i \) is always accompanied with the speed increase of the task \( \tau_i \), the number of the consecutive ICPs is limited by the number of available speeds, which is expressed in the Equation 10. Finally, if we transform the delays into the cycles, the Equation 9 is obtained.

Now we are ready to derive the schedulability test for speed inheritance protocol stated as the Theorem 2.

**Theorem 2.** Given a cluster-based multi-core processor platform and a set of independent periodic real-time tasks, a low power solution containing a task partition and a speed assignment can guarantee the system feasibility under speed inheritance protocol, if the following condition for each core \( D_{x,y} \) holds

\[
\sum_{alloc(\tau_i) = D_{x,y}} \frac{W_i}{T_i} + 2 \ast L_p + num'_i \ast L_p \leq U_{ub} \tag{11}
\]

**Proof:** By applying the Lemma 2 we obtain the maximal cycles that the \( AFC_i \) may lag behind the \( EFC_i \). In most cases this lag will be compensated by using the speed inheritance technique. However, it is under the assumption that the task has enough time to do it, i.e. the remaining cycles of the task are enough for \( t_{be} \). With other words, if there is not enough time, we have no chance to compensate these delays. In this case we use the old fashion where these non-compensable delays are accounted to the task worst case execution time. Therefore we obtain the sufficient condition (11) for the schedulability test by using the speed inheritance protocol.

Clearly, the speed inheritance protocol makes a more tighter bound than the conservative protocol, because the number of delays added into the task worst case execution time is now limited by the number of available speeds, which is usually relatively small. However, the speed inheritance protocol requires the runtime support by maintaining the \( EFC \) and \( AFC \) values as mentioned above. The impact of the runtime overhead is shown in the next section.

**VI. EVALUATION**

Before the results are presented, we first introduce the experiment setup.

**A. Experiment Setup**

Our evaluation is performed by using a SystemC RTOS simulation framework [19]. We extended their library by adding the additional support for multi-core simulation with DPM/DVS capabilities. Since we are interested in cluster-based multi-core processor platforms, different configurations are investigated, e.g. the platform with 4 cores grouped into 2 clusters or into 1 cluster. In this experiment setup we concentrate on symmetry cluster-based platforms, where all clusters contain the same number of cores. However, our algorithm is not constrained to symmetry configuration. Furthermore, for the sake of simplicity we assume that all cores possess the same power model. In the following test scenarios we refer to the platform with \( X \) cores grouped into \( Y \) clusters as \( \text{coreXclusterY} \).

For each core we adopt the power model of Intel PXA270 processor [2], which supports 5 P-states (from 208MHz to 624MHz with P-state switching latency 450µs, we assume that switching among different speeds takes the same latency) and 3 C-states (active, idle state with switching latency 1µs and deep-sleep state with switching latency 600µs). Furthermore, we investigated the platforms with 4, 8 and 16 cores. For each of them we generated 100 task sets (each task has a period within \([0.5ms, 20ms] \)) and for each task set we randomly generated 100 solutions. Therefore, for each platform we have performed 10000 tests. In addition, we applied the EDF as real-time schedule on each core.

**B. False Negative Errors**

In this subsection the false negative errors are investigated. These errors will occur, if the schedulability test fails but the system is still schedulable. This type of errors is a measurement to show how much a schedulability test overestimates. With other words, it indicates that the schedulability test is not a necessary condition. In our context these errors can be afforded, however, the number of them should
be kept as small as possible. The Fig. 4(a), 4(b) and 4(c) show the results of false negative error rate comparison on 4-, 8- and 16-core platforms, respectively. The figures confirm our expectation that the speed inheritance protocol (SIP) outperforms the conservative protocol (CP) on all configurations. By increasing the cluster size, the advantage of the speed inheritance protocol becomes more significant.

C. Runtime Overhead

For the DPM state switching overhead handling and speed inheritance protocol we have introduced some runtime mechanism at each scheduling point. In this subsection their actual overhead are discussed. Since the evaluation is performed through simulation, we compared the overhead introduced by our approach with the overhead caused by the traditional EDF scheduling algorithm implemented in the SystemC RTOS simulation framework [19]. More specifically, we define $\rho_1$ and $\rho_2$ as:

$$\rho_1 = \frac{\text{Algorithm 1 overhead at each scheduling point}}{\text{EDF overhead at each scheduling point}}$$  \hspace{1cm} (12)

$$\rho_2 = \frac{\text{SIP overhead at each scheduling point}}{\text{EDF overhead at each scheduling point}}$$  \hspace{1cm} (13)

The experiment results show $\rho_1 = 1.52$ and $\rho_2 = 0.85$ in average, which means that the actual overhead introduced by our approach is in the same order of magnitude as the EDF overhead. As a consequence, the energy consumed due to our overhead is similar to the energy consumed by the EDF schedule as well.

VII. CONCLUSION

With the continuous increasing of system complexity and the technology advance towards multi-core processor platforms, the problem of energy efficiency becomes more and more difficult. The DPM and DVS are two well accepted runtime techniques to trade off the system performance and power dissipation. In this article, we have shown that the existing single-core DPM/DVS related energy-aware scheduling can not be simply adopted for cluster-based multi-core platforms, especially by taking the non-negligible DPM and DVS state switching overhead into consideration. The main problem consists in the insufficiency of the traditional schedulability test. Therefore, we proposed a simple runtime prediction technique to deal with the DPM state switching overhead and two solutions to enhance the schedulability analysis for DVS state switching overhead handling. The conservative protocol suffers the problem of large overestimation, however, it requires no modification in the task execution. The second approach, the speed inheritance protocol, gives a tighter bound but needs some runtime support. Finally, we proved both schedulability analyses and the experiment results confirmed our expectation.

VIII. ACKNOWLEDGEMENT

This work was partly funded by the DFG Collaborative Research Centre 614 and by the German Ministry of Education and Research (BMBF) through the BMBF project SANITAS (01M3088) and the ITEA2 projects VERDE (01IS09012), and TIMMO-2-USE (01IS10034A).

REFERENCES