Sleep Scheduling for Energy-Savings in Multi-Core Processors

Sandeep D'souza, Anand Bhat, Ragunathan (Raj) Rajkumar
Electrical and Computer Engineering
Carnegie Mellon University
Email: sandeepd@andrew.cmu.edu, anandbha@andrew.cmu.edu, raj@ece.cmu.edu

***Abstract*—As transistor geometries get smaller, static leakage power dominates the power consumption in modern processors. This phenomenon diminishes the ability of frequency scaling-based techniques to save energy. Modern processors also provide sleep states which minimize leakage power by gating portions of the processor and/or the system clock. This paper presents partitioned fixed-priority scheduling solutions for utilizing these sleep states to efficiently schedule periodic real-time tasks, and maximize energy savings on multi-core processors. The techniques presented rely on an Enhanced Version of *Energy-Saving Rate-Harmonized Scheduling* (ES-RHS), and our newly proposed *Energy-Saving Rate-Monotonic Scheduling* (ES-RMS) policy to maximize the time the processor spends in the lowest-power *deep sleep* state. In some modern multi-core processors, all cores need to transition synchronously into deep sleep. For this class of processors, we present a partitioning technique called *Max-SyncSleep* which utilizes *a priori* task information, to maximize the synchronous deep sleep duration across all processing cores. The performance of *Max-SyncSleep* is compared to the classical *Worst-Fit Decreasing* load balancing heuristic. We also illustrate the benefits of using ES-RMS over ES-RHS for this class of processors. For processors which allow cores to individually transition into deep sleep, we prove that, while utilizing ES-RHS on each core, any feasible partition can optimally utilize all of the processor's idle durations to put it into deep sleep. Experimental evaluations indicate that the proposed techniques can provide significant energy savings.**

I. Introduction

CMOS power dissipation has significantly increased [1], as a result of technology scaling. Thus, increases in performance can no longer be obtained by increasing the operating processor frequency. Multi-core processors that consist of multiple processing units integrated on a single chip have instead become the norm for achieving higher performance. Apart from being ubiquitous in the data center and personal computing domains, it is not uncommon to find processors with up to eight cores on battery-powered platforms such as smartphones [2][3], and in low-cost embedded platforms such as the Raspberry Pi 2 [4]. The use of multi-core systems in a wireless sensor network platform [5] is also sometimes useful.

The emergence of "Fog Computing" and the Internet of Everything (IoE) [6] has begun to place an increasing demand for computation on highly-connected and mobile devices. Platforms such as Google Glass [7] and smartwatches are examples of IoE systems using multi-core processors. The need for high performance in an energy-constrained environment makes it necessary to investigate the use of various energy-management techniques for multi-core systems. To increase battery life, modern processors are equipped with a number of energy management features. Primary among them are Dynamic Voltage and Frequency Scaling (DVFS) [8], and the use of low-power sleep states. The use of DVFS enables the processor to change its operating frequency and voltage, thereby reducing *dynamic switching power*. On the other hand, low-power sleep states use power gating and/or clock gating [9] to reduce *static leakage power* dissipation when the processor is idle. However, there is a minimum *round-trip* time associated with each of these low-power sleep states [10]. This round-trip time is longer for moving to and from lower-power states due to the overhead required for the main oscillator to startup and stabilize [10].

From a real-time systems scheduling perspective, it is critical that all tasks meet their deadlines to ensure reliable system operation. The presence of such systems in resource-constrained and mobile environments, also makes it essential that the system save energy. As technology scales, the dominance of static power makes it necessary that scheduling techniques take advantage of built-in processor sleep states. Based on this motivation, Rowe. *et al.* [10] proposed *Energy-Saving Rate-Harmonized Scheduling* (ES-RHS) which utilizes the processor deep sleep state to maximize energy savings.

A. Contributions of the Paper

The focus of this paper is on energy-efficient multi-core partitioned fixed-priority real-time scheduling techniques, which utilize the processor's deep sleep state to reduce static leakage power, and hence save energy. The primary contributions of the paper are as follows:

- We propose an enhanced version of ES-RHS, named ES-RHS+, which has better schedulability properties than ES-RHS.
- We provide an energy-saving version of traditional rate-monotonic (deadline-monotonic) scheduling, named *Energy-Saving Rate-Monotonic (Deadline-Monotonic) Scheduling* (ES-RMS and ES-DMS), which has better energy-saving guarantees than ES-RHS and ES-RHS+ for multi-core processors where all cores can only transition into deep sleep state together.
- We present a new task partitioning heuristic that increases synchronized sleep times, where all cores can only transition into deep sleep state together.

- We prove that ES-RHS (ES-RHS+) is optimal for energy savings on multi-core processors where, each core can independently go into deep sleep state, for any partition feasible under multi-core ES-RHS (ES-RHS+).

B. Organization of the Paper

The rest of the paper is organized as follows. Section II discusses related work. Section III proposes an enhanced version of ES-RHS, named ES-RHS+. Section IV introduces ES-RMS. Section V introduces the problem of utilizing the processor's deep sleep state for energy savings in multi-core processors, and illustrates the usefulness of ES-RMS for multi-core processors where all cores can only transition into deep sleep state together. Section VI discusses both partitioning and scheduling techniques for multi-core processors where all cores can only transition into deep sleep state together. Section VII proves the optimality of ES-RHS+ in the context of multi-core processors where each core can independently go into deep sleep. Section VIII presents an experimental evaluation of all the proposed techniques, and Section IX provides concluding remarks.

II. Related Work

In the domain of real-time systems, the use of frequency scaling-based energy saving scheduling techniques has been well-studied [11-14]. In the context of Rate-Monotonic Scheduling (RMS), Saewong *et al.* [11] proposed the *Sys-Clock* algorithm to analytically determine the energy-optimal frequency at which the processor must run, so that a task set meets all its deadlines. In the same work, dynamic frequency scaling-based scheduling techniques named *PM-Clock* and *DynamicPM-Clock* [11] were also proposed. In [14], Arvind *et al.* proposed the *Static Frequency Assignment Algorithm* (SFAA) to partition tasks on multi-core processors which have a single frequency domain. SFAA extends the *Sys-Clock* framework to the multi-core context and aims to minimize the operating frequency across all cores, so as to minimize the energy consumption. In [12][15][16], DVFS-based scheduling techniques for multi-core processors can be found, where each core has its own voltage and frequency domain.

At technology nodes smaller than 65nm [17], static leakage power already dominates the total power consumption of CMOS-based VLSI circuits. For general-purpose computing workloads, Le Sueur *et al.* [18] compared the energy efficiency of DVFS and sleep state-based power management techniques. Their work experimentally analyzes the trade-offs between *slow down* (DVFS) and *race-to-halt* (sleep) for a range of computing platforms such as the desktop-class Intel i7 870 and the low-power Intel Atom Z550. The result of the analysis concluded that using C-state based techniques offer improved energy efficiency with a small impact on performance [18].

In the context of real-time systems, *a priori* task execution information can be utilized to schedule tasks efficiently so as to maximize the time spent in low-power states. Scheduling techniques that cluster task executions to save energy have been proposed in [19-23]. In [20,22,23], dynamic-priority EDF scheduling is used. [21] uses procrastination of tasks to determine the instances at which the processor can be shut-down. [19] uses a fixed-priority scheduling based approach, which relies on online simulation to estimate the duration for which a task can procrastinate, thus significantly increasing the scheduler run-time complexity. For tasks with given release times and deadlines, [24] and [25] present dynamic priority-based scheduling techniques to maximize the common idle time on multi-core processors. In [10], Rowe *et al.* proposed and analyzed the benefits of using ES-RHS in uniprocessor systems. ES-RHS is a simple, easy-to-implement approach based on a notion of *harmonization*, that aggregates all the processor idle durations together. This allows the processor to be put into deep sleep for all idle durations, thus enabling optimal energy savings for processors which lack frequency-scaling capabilities [10]. In conjunction with reservation-based real-time operating systems (such as Linux/RK [26]), ES-RHS presents an effective approach for energy management in uniprocessor real-time systems, and is particularly useful for processors which have only one sleep state [2]. The work by Rowe *et al.* [10] also offers brief guidelines for the use of ES-RHS in the multi-core context.

III. Energy-Saving Rate-Harmonized Scheduling

This section introduces the notation used in the context of uniprocessor ES-RHS. We then describe a version of ES-RHS with enhanced schedulability conditions, named ES-RHS+.

A. Notation and Background

Consider a task set Γ consisting of n independent[1] periodic real-time tasks $\tau_1, \tau_2, ..., \tau_n$. Each task $\tau_i \in \Gamma$ can be characterized by $\{C_i, T_i, D_i\}$, where C_i is the worst-case execution time, T_i is the period, and D_i is the relative deadline from its arrival time. In this paper, we assume that $D_i = T_i$, i.e., for each task, deadlines are implicit. The utilization of a task τ_i is given by $U_i = C_i/T_i$. Consider fixed-priority preemptive scheduling, with task priorities assigned using the rate-monotonic policy [27]. The task set is listed in non-increasing order of task priorities such that $T_1 \leq T_2 \leq .. \leq T_n$. Each task has an initial arrival time of ϕ_i, such that its arrival times are $\phi_i, \phi_i + T_i, \phi_i + 2T_i, ...$ Without loss of generality, we assume that the initial arrival time of task $\tau_1, \phi_1 = 0$.

The family of *Rate-Harmonized Schedulers* [10] utilizes a periodic value T_H, referred to as the *Harmonizing Period*. As described in [10], the Harmonizing Period has the same initial phasing as the highest priority task τ_1, i.e, $\phi_1 = 0$. No such phasing constraints are imposed on the other tasks. In the basic *Rate-Harmonized Scheduler* (RHS), tasks that arrive before or after integral multiples of T_H are not eligible to execute until the next closest boundary of T_H, when they are serviced based on their priority [10]. For a given task set Γ, T_H is chosen so as to improve schedulability [10]. As stated in [10], let us suppose $\Psi = \{\tau_j | T_j < 2T_1, j \neq 1\}$. If $\Psi = \varnothing$, $T_H = T_1$, otherwise $T_H = T_1/2$.

[1]Task release jitter and task dependence can be incorporated using the frameworks proposed in [28] and [29], and is beyond the scope of this work.

In ES-RHS, by using a periodic *Energy Saver* task τ_{sleep} in conjunction with RHS, optimal energy savings can be achieved. The *Energy Saver* task τ_{sleep}, is scheduled at the highest priority with its period $T_{sleep} = T_H$, initial arrival time $\phi_{sleep} = \phi_1 = 0$ and execution time $C_{sleep} \geq C_{SleepMin}$, where $C_{SleepMin}$ is a system constraint that represents the minimum round-trip time required for the processor to go into the deep sleep state, and revert back to the active state. While using ES-RHS, the state of the processor can be broadly classified as follows [10]:

- *Busy:* The processor is executing a task $\tau_i \in \Gamma$.
- *Forced Sleep:* The processor is forced into *deep sleep* by the *Energy Saver* task τ_{sleep}.
- *Idle:* The processor is neither *busy* nor in *forced sleep*.

ES-RHS exhibits an interesting property, where every *idle* duration precedes, and is contiguous with the *forced sleep* duration. Thus, idle durations can *always* be merged with the subsequent *forced sleep* duration [10]. Hence, by *harmonizing* the executions of non-harmonic tasks, ES-RHS can yield an optimal sleep schedule. We now re-define the notion of *harmonization* to enhance the schedulability of ES-RHS.

B. Energy-Saving Rate-Harmonized Scheduling+

We first re-define the notion of *harmonization* as follows:

A task is eligible to execute when the processor is *busy* or a *Harmonizing Period* boundary has been reached.

The above re-definition allows tasks to become eligible earlier than previously defined rate-harmonized schedulers including RHS and ES-RHS, without affecting their worst-case energy savings. Based on this new notion of harmonization, we propose ES-RHS+. To illustrate our new definition of harmonization, consider the ES-RHS schedule in Figure 1. The second instance of task τ_2 arrives at time $t = 23$ but only becomes eligible to execute at $t = 30$. Under our re-definition, the second instance of τ_2 becomes eligible to execute at time $t = 23$, because the processor is *busy*, i.e., the forced sleep task is "executing". Similarly, the second instance of task τ_3 which arrives at time $t = 36$ becomes eligible at $t = 40$ under ES-RHS. Under our new definition, since the processor is busy at time $t = 36$ (executing τ_1), it becomes eligible to execute immediately. Eligible tasks will continue to be scheduled based on their respective scheduling priorities.

Using the new definition of harmonization, a task which arrives when the processor is *busy* (including *forced sleep*) becomes eligible to execute immediately. In ES-RHS, such tasks can execute only after the *next* instance of τ_{sleep} finishes execution. In the worst case, a task $\tau_j, j \neq 1$, which arrives just after the harmonizing period boundary, has to wait until the next harmonizing period boundary to become eligible to execute. This induces a worst-case blocking duration of T_{sleep}. Under ES-RHS+, the worst-case blocking for a task $\tau_j, j \neq 1$, happens when it arrives just after the Energy Saver task has *finished* execution. It becomes eligible to execute *no later* than the next harmonizing period boundary, giving rise to a worst-case blocking term of $T_{sleep} - C_{sleep}$.

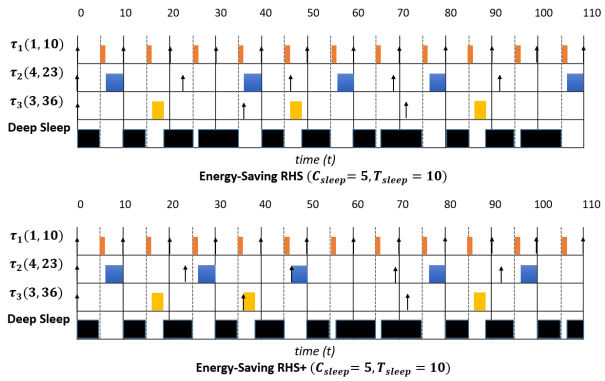

Fig. 1. The task set $\tau_1 = (1, 10)$, $\tau_2 = (4, 23)$, $\tau_3 = (3, 36)$ being scheduled with ES-RHS on the top, and ES-RHS+ at the bottom. For both cases, $C_{sleep} = 5, T_{sleep} = 10$.

We now prove that the energy savings obtained by using ES-RHS are still true for ES-RHS+.

Theorem 1: Every *idle* duration in the ES-RHS+ schedule will precede and be contiguous with a forced sleep duration.

Proof: The Energy Saver task, τ_{sleep}, executes at the highest priority in the system, with an initial phasing $\phi_{sleep} = \phi_1 = 0$. Hence, the processor will be in forced sleep in the intervals $[(k-1)T_{sleep}, (k-1)T_{sleep} + C_{sleep})$, where, $k = 1, 2, 3, ...$ Correspondingly, the processor is considered to be *busy* in the intervals $[(k-1)T_{sleep} + C_{sleep}, kT_{sleep})$ where, $k = 1, 2, 3, ...$ Let t be any time instant at which the processor becomes *idle*, i.e., t represents the beginning of an idle duration. Given the execution pattern followed by ES-RHS+, t must lie in the interval $[(k-1)T_{sleep} + C_{sleep}, kT_{sleep})$ where, $k = 1, 2, 3, ...$ It needs to be shown that the interval (t, kT_{sleep}), for any positive value of k, is an idle duration which in turn precedes the forced sleep execution of the Energy Saver task τ_{sleep}.

For ES-RHS+, $T_H = T_{sleep}$. Hence, any task $\tau_i \in \Gamma$ that arrives in the interval (t, kT_{sleep}), for any positive value of k, would become eligible to execute at the next harmonizing period boundary, i.e., kT_{sleep}. Any task $\tau_i \in \Gamma$, which arrives before $(k-1)T_{sleep}$, in the worst case, would have become eligible to execute at $(k-1)T_{sleep}$. If any task arrives in the interval $[(k-1)T_{sleep} + C_{sleep}, t)$, i.e., when the processor is *busy*, then using the re-defined notion of harmonization, it would have become eligible to execute immediately. If τ_i still has some execution time left over at t, then ES-RHS+ must schedule τ_i at time t. This contradicts the assumption that t represents the beginning of an idle duration. ■

Theorem 2: Every idle duration in the ES-RHS+ schedule can be utilized to put the processor into deep sleep without any additional penalty.

Proof: From Theorem 1, all idle durations precede and are contiguous with the forced sleep execution. Hence, all idle durations in the system can be combined with C_{sleep} to create a single chunked deep sleep execution, guaranteeing that whenever the system becomes idle, it can transition into a deep sleep duration greater than or equal to C_{sleep}. ■

Given that all idle durations in the ES-RHS+ schedule can be combined with the forced sleep execution, the processor

utilization spent in deep sleep is given by:

$$U_{sleep} = 1 - \sum_{i=1}^{n} \frac{C_i}{T_i} = 1 - U_{taskset}$$

where, $U_{taskset}$ is the total utilization of Γ. Thus, for a task set Γ schedulable by ES-RHS+, the processor utilization spent in deep sleep is maximal. The following theorem yields the worst-case feasibility conditions, based on utilization bounds, for a task set to be schedulable by ES-RHS+.

Theorem 3: A task set Γ is feasible under ES-RHS+ if

$$\frac{C_{sleep}}{T_{sleep}} + \frac{C_1}{T_1} \leq 1 \quad \wedge$$

$$\frac{C_{sleep}}{T_{sleep}} + \sum_{j=1}^{i} \frac{C_j}{T_j} + \frac{T_{sleep} - C_{sleep}}{T_i} \leq i(2^{1/i} - 1) \quad \forall i = 2 \; to \; n$$

Proof: Under ES-RHS+, it is always guaranteed that τ_1 executes immediately after the execution of τ_{sleep}. Hence, we can equivalently assume that τ_1 and τ_{sleep} together form a high-priority task set scheduled using RMS with harmonic periods. Thus, given the harmonicity of τ_1 and τ_{sleep}, using RM-theory [27], if $(C_{sleep}/T_{sleep}) + (C_1/T_1) \leq 1$, then τ_1 is schedulable by ES-RHS+.

Consider other tasks in the task set Γ, $\tau_i \; \forall i = 2 \; to \; n$. Compared to RMS, an instance of τ_i incurs a maximum additional delay (or blocking) of $T_{sleep} - C_{sleep}$. Hence, apart from the preemption term contributed by the execution of τ_{sleep}, the term $T_{sleep} - C_{sleep}$ can be added as a blocking term to the computational time of τ_i (C_i), and the RMS utilization bounds [27] can be used to test feasibility. ■

The tests based on RMS utilization bounds are pessimistic in nature. A more practical schedulability test utilizes the estimation of the worst-case response time of task $\tau_i \in \Gamma$. For ES-RHS+, the worst-case response time test for a task τ_i is given by the following recurrence relations:

$$W_0 = C_i + T_{sleep} - C_{sleep} \tag{1}$$

$$W_{k+1} = W_0 + \left\lceil \frac{W_k}{T_{sleep}} \right\rceil C_{sleep} + \sum_{j=1}^{i-1} \left\lceil \frac{W_k}{T_j} \right\rceil C_j \tag{2}$$

until $W_{k+1} = W_k$, in which case, W_{k+1} is the worst-case response time of the task τ_i. If $W_{k+1} \leq D_i$, then τ_i will be schedulable, otherwise τ_i will miss its deadline.

Compared to the schedulability conditions of ES-RHS [10], ES-RHS+ enhances the schedulability and feasibility conditions due to the reduction in the blocking faced by tasks. Another important property of both ES-RHS and ES-RHS+ is their inherent ability to perform *slack stealing*. In situations where tasks do not execute up to their worst-case execution time, all the additional slack can be used to put the processor into deep sleep without any additional penalty (Theorem 2).

IV. Energy-Saving Rate-Monotonic Scheduling

ES-RMS is a practical extension to RMS designed with the objective of maximizing energy savings in some existing operating systems. As presented in subsequent sections, ES-RMS can also help maximize energy savings for some multi-core processors. Motivated by ES-RHS, the basic Rate-Monotonic Scheduler can be extended to use a periodic *Energy Saver* task, τ_{sleep}, that executes at the highest priority with its execution time $C_{sleep} \geq C_{SleepMin}$, period $T_{sleep} = T_1$ or $T_1/2$ and phasing $\phi_{sleep} = \phi_1 = 0$. The following theorem provides the worst-case feasibility conditions for a task set to be schedulable by ES-RMS, based on utilization bounds.

Theorem 4: A task set on a uniprocessor is feasible under ES-RMS if

$$\frac{C_{sleep}}{T_{sleep}} + \frac{C_1}{T_1} \leq 1 \quad \wedge$$

$$\frac{C_{sleep}}{T_{sleep}} + \sum_{j=1}^{i} \frac{C_j}{T_j} \leq i(2^{1/i} - 1) \quad \forall i = 2 \; to \; n$$

Proof: Under ES-RMS, the forced sleep execution τ_{sleep}, has an initial phasing of $\phi_{sleep} = \phi_1 = 0$ and a period $T_{sleep} = T_1 \; or \; T_1/2$. Hence, τ_1 always executes immediately after the execution of τ_{sleep}, and together form a high-priority task set, scheduled using RMS with harmonic periods. Thus, given the harmonicity of τ_1 and τ_{sleep}, using RM-theory [27] we can say that if $(C_{sleep}/T_{sleep}) + (C_1/T_1) \leq 1$, then τ_1 is schedulable by ES-RMS.

Consider other tasks in the task set Γ, $\tau_i \; \forall i = 2 \; to \; n$. The preemption term contributed by the execution of τ_{sleep} has to be included, and the RMS utilization bounds [27] are used to test feasibility. ■

For ES-RMS, the worst-case response time test for a task τ_i is given by the recurrence relations:

$$W_0 = C_i \tag{3}$$

$$W_{k+1} = C_i + \left\lceil \frac{W_k}{T_{sleep}} \right\rceil C_{sleep} + \sum_{j=1}^{i-1} \left\lceil \frac{W_k}{T_j} \right\rceil C_j \tag{4}$$

until $W_{k+1} = W_k$, in which case, W_{k+1} is the worst-case response time of the task τ_i. If $W_{k+1} \leq D_i$, then τ_i will be schedulable, otherwise τ_i will miss its deadline.

ES-RMS can also be readily extended to the case where task deadlines are not implicit, by using a version of *Deadline-Monotonic Scheduling* (DMS), ES-DMS.

V. Sleep-State based Energy-Saving on Multi-Core Processors

In this section, we are concerned with utilizing the processor's deep sleep state for energy-efficient real-time multi-core scheduling. We focus on *fully partitioned* fixed-priority scheduling. Most modern multi-core processors support a number of low-power states called C-states. In processors which have more than one C-state, individual cores can transition to idle states. However, not all processors give each core the ability to individually transition into deep sleep. Based on the ability to transition into deep sleep, we can define two types of energy-saving scheduling problems:

1) Multi-core processors where all cores can only transition *synchronously* into deep sleep. For this class of processors, we refer to the scheduling problem as "Synchronized-Sleep Multi-Core Energy-Saving Scheduling", hereafter referred to

as *SyncSleep Scheduling*. Examples of such processors are Intel Core2 Duo [30] and AMD Opteron [31].

2) Multi-core processors where each core can independently transition into deep sleep. For this class of processors, we refer to the scheduling problem as "Independent-Sleep Multi-Core Energy-Saving Scheduling", hereafter referred to as *IndSleep Scheduling*. Examples of such processors are Samsung Exynos 5800 [2] and the 4th generation Intel Core processors [32].

Consider *SyncSleep* scheduling, where the minimum amount of time for which the system can be in deep sleep is dictated by the core which has the least forced sleep duration. To maximize energy savings, the scheduler needs to maximize the minimum forced sleep duration across all the cores. This requires task partitioning heuristics which efficiently distribute the load across the cores so as to achieve more energy savings. Given a balanced partition, the amount of forced sleep duration possible on a core also depends on the scheduling algorithm used on each core.

Consider the use of ES-RHS+ on each core for *SyncSleep* scheduling. There is no guarantee that the idle durations across all cores are of the same length. Hence, all the idle time cannot be used to put the processor into deep sleep. We need to maximize and synchronize the per-core forced sleep duration, so as to maximize the guaranteed overlap between the forced sleep executions on each core. Thus, scheduling techniques which can guarantee a longer *synchronous forced sleep* execution will provide greater energy savings.

Lemma 1: Consider two uniprocessor fixed-priority preemptive scheduling policies X and Y, where X has *better* schedulability conditions than Y (hence X can schedule all task sets schedulable by Y but not vice versa). Then, given a task set Γ and the period of the sleep task T_{sleep}, the maximum amount of time for which the processor can be in *forced sleep* for policy X is greater than or equal to policy Y.

Proof: Assume that a task set exists, such that Y provides a longer forced sleep duration than X. This implies that the combined utilization of the task set and the forced sleep for Y, is greater than that for X. This contradicts the assumption that X has better schedulability conditions than Y. ■

Using Equations (1-2) and (3-4), we can conclude that all possible task sets schedulable by ES-RHS+ will be schedulable by ES-RMS. However, the opposite is not true. Hence, using Lemma 1, we can say that given a task set, ES-RMS can provide a forced sleep duration that is greater compared to ES-RHS+. This property of ES-RMS is beneficial for *SyncSleep* scheduling.

We extend ES-RMS and ES-RHS+ to the multi-core scheduling context by adding an Energy Saver task on each core. In the following sections, solutions for both *SyncSleep* and *IndSleep* scheduling are presented.

VI. SyncSleep Scheduling

In this section, we formally state the energy minimization (or deep sleep maximization) problem for *SyncSleep Scheduling*, which is as follows:

Consider a task set Γ, consisting of *n* periodic real-time tasks $\tau_1, \tau_2, ..., \tau_n$ that need to be scheduled on a homogeneous multi-core processor with *m* cores, $M_1, M_2, ..., M_m$. Each core M_k has a Energy Saver task, $\tau_{sleep,k}$ where $k = 1, 2, ..., m$. $\tau_{sleep,k}$ has a forced sleep duration of $C_{sleep,k}$ every $T_{sleep,k}$. The system has the constraint that all cores must synchronously transition into deep sleep. Our objective is to find a partition, and compute the global synchronized forced sleep duration for all the cores such that:

1) The workload allocated to each core can be scheduled by ES-RMS (or ES-RHS+) in a feasible manner.
2) The synchronized forced sleep duration is maximized, thus ensuring that the partition maximizes the minimum guaranteed energy savings among all partitions.

The above problem is a more constrained form of the feasibility problem in multi-core processor scheduling, which is known to be NP-hard in the strong sense [14]. Hence, the *SyncSleep* scheduling problem is also NP-hard. For the case, where all tasks have the same periods, with different computation time, it is similar to the Bin-Packing problem, which is known to be NP-Hard [14].

For *SyncSleep* scheduling, the forced sleep execution must be synchronized across all the cores. Hence, for every $\tau_{sleep,k}$ where $k = 1, 2, ..., m$, we let $T_{sleep,k} = T_{sleep}$, and the initial phase be $\phi_{sleep,k} = \phi_{sleep} = 0$. Therefore, if $\min_{k=1}^{m}(C_{sleep,k}) \geq C_{SleepMin}$, then the minimum guaranteed deep sleep utilization is given by $\min_{k=1}^{m}(C_{sleep,k})/T_{sleep}$. For a partition to be feasible, we must have $\min_{k=1}^{m}(C_{sleep,k}) \geq C_{SleepMin}$. If $\min_{k=1}^{m}(C_{sleep,k}) < C_{SleepMin}$, then the system cannot transition into deep sleep.

Given a feasible partition and assuming that T_{sleep} is fixed for a given task set Γ, to maximize energy savings, we need to maximize $C_{sleep,k}$ for each processing core without changing the partition, such that the tasks on each core do not become unschedulable. For each core in a feasible partition, $C_{SleepMin} \leq C_{sleep,k} \leq T_{sleep}$. By using the schedulability tests based on utilization bounds (presented in Section III and IV), we can obtain a conservative estimate of $C_{sleep,k}$ for each processing core. However, the schedulability tests based on utilization bounds are pessimistic in nature, and higher $C_{sleep,k}$ values can be achieved on each core. We now present a technique, which conducts a binary search over $[C_{SleepMin}, T_{sleep}]$ so as to efficiently compute the maximum $C_{sleep,k}$ on each core.

Consider a partitioning algorithm P, which partitions a task set Γ onto m cores, such that the partition is feasible in the context of *SyncSleep* scheduling (either using ES-RMS or ES-RHS+). Then, for the subset of tasks $\Gamma_k \in \Gamma$ assigned to core $k \in 1, 2, ..., m$, from Equations (2) and (4), the response time of the task $\tau_i \in \Gamma_k$ is a non-decreasing function of the forced sleep duration $C_{sleep,k}$. Since the schedulability of a task depends on its response time, for a subset of tasks $\Gamma_k \in \Gamma$ assigned to core $k \in 1, 2, ..., m$, if Γ_k is not schedulable for $C_{sleep,k} = C_x$, then Γ_k will not be schedulable for any value of $C_{sleep,k} \geq C_x$. Based on this property, we use binary search to maximize $C_{sleep,k}$ on a core k, given a partition.

We call this technique *Binary "maxSleep" Search* (or BMS) and present it in Algorithm 1. BMS utilizes the response time tests (Equations (1-2) and (3-4)) to test schedulability, and has a complexity of $O(log_2(T_{sleep}/\epsilon))$. Hereafter, we assume that all the partitioning schemes used in this paper use BMS to maximize the forced sleep execution on each core.

Algorithm 1 Binary "maxSleep" Search

```
1:  procedure BMS (Γ_k, C_SleepMin, T_sleep, ϵ)
2:      C^dn_sleep,k ← C_SleepMin                ▷ Set lower bound
3:      C^up_sleep,k ← T_sleep                   ▷ Set upper bound
4:      TestSchedulability(Γ_k, C_SleepMin, T_sleep)
5:      if not Schedulable then
6:          return Γ_k not Schedulable
7:      end if
8:      while (C^up_sleep,k − C^dn_sleep,k) ≥ ϵ do
9:          C_sleep,k ← (C^up_sleep,k + C^dn_sleep,k)/2
10:         TestSchedulability(Γ_k, C_sleep,k, T_sleep)
11:         if Schedulable then
12:             C^dn_sleep,k ← C_sleep,k
13:         else
14:             C^up_sleep,k ← C_sleep,k
15:         end if
16:     end while
17:     return C^dn_sleep,k
18: end procedure BMS
```

For a given partition, the guaranteed energy savings depends on the deep sleep utilization [10]. As ES-RMS has better schedulability conditions than ES-RHS+, the former can provide better guarantees on minimum deep sleep utilization for *SyncSleep* scheduling. Therefore, we consider multi-core ES-RMS, and then compare it experimentally with multi-core ES-RHS+.

A. Load Balancing to Maximize Energy Saving

Load balancing is often used to realize energy savings in multi-core systems [14]. For *SyncSleep* scheduling, it makes intuitive sense to balance the load across all cores, so as to maximize the synchronized forced sleep duration. Based on this intuition, we state the following lemma:

Lemma 2: For *SyncSleep* scheduling, an ideal energy-aware partitioning technique is one that partitions a task set Γ with utilization U_{total} onto m processors, such that the forced sleep utilization on each processor is $1 - (U_{total}/m)$.

Proof: Assume an algorithm Φ exists, which for a task set Γ, can achieve a partition with forced sleep utilization of $1-(U_{total}/m)+\epsilon$ on each processor, where ϵ is a finite positive quantity. Given this forced sleep execution, the amount of processor utilization available for tasks to run is $U_{total} - m\epsilon$. This is less than the utilization required for the task set Γ to execute, and hence tasks will start missing their deadlines. Hence, no such algorithm Φ can exist. ■

From Lemma 2, we can conclude that, if a balanced partition exists, then an ideal algorithm creates a completely balanced partition, such that the synchronous forced sleep utilization is maximized across all the cores in the system. Among the task partitioning heuristics studied in the literature, the *Worst-Fit Decreasing* (WFD) algorithm is known to typically produce a well-balanced partition [14]. WFD allocates tasks one by one in non-decreasing order of their utilization. Given a task to be allocated, WFD allocates it to the core with the least utilization. For this type of problem, where WFD can allocate tasks to use only *m* cores, it is equivalent to *List Scheduling*, where tasks with utilization $u_i \in [0, 1]$ are allocated to m cores, with the aim of reducing the maximum utilization on any core. Given that WFD and List Scheduling are equivalent in this context, we refer to both as WFD.

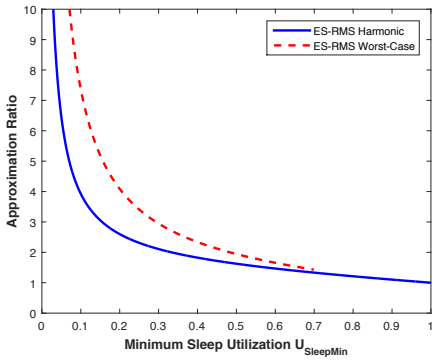

Fig. 2. Approximation ratio for ES-RMS and WFD as a function of $U_{SleepMin}$

In [33], it was proved that, in an m-core system, for a partition generated using WFD with *Earliest-Deadline First* (EDF) scheduling (or equivalently RMS with only harmonic tasks), the core with the maximum load has a utilization of no more than $(4/3) - (1/m)$ times that of the optimal. As $m \to \infty$, the ratio becomes $4/3$. In [14], Arvind *et al.* extended this result to the case where RMS is used along with WFD. The result states that the core with the maximum load has a utilization no more than $[(4/3) - (1/m)]/ln2$ times that of the optimal. As $m \to \infty$, the ratio becomes $4/(3ln2)$. In the general case, for a scheduling technique with a per-core utilization bound U_{bound}, it can be stated that the core with the maximum load has a utilization no more than $[(4/3) - (1/m)]/U_{bound}$ times that of the optimal [14].

The approximation ratio of an algorithm is the worst-case ratio between the result obtained by the algorithm, as compared to the optimal solution. For *SyncSleep* scheduling, the approximation bound can be defined as the ratio of the synchronous forced sleep utilization obtained using an algorithm, compared to the optimal. In the following theorems, we state and prove the approximation bound for WFD while using ES-RMS. Given the nature of the problem, the approximation bound is a function of the synchronous forced sleep utilization obtained by WFD. This worst-case synchronous forced sleep utilization is the minimum forced sleep utilization supported by the system and is given by $U_{SleepMin}$.

Theorem 5: For *SyncSleep* scheduling, the WFD heuristic for partitioning independent *harmonic* tasks onto a multiprocessor, under ES-RMS, has an approximation ratio of at

most:

$$\frac{4 - 3(1 - U_{SleepMin})^2}{4U_{SleepMin}}$$

Proof: Let WFD yield a partition with a synchronous forced sleep utilization of $U_{SleepMin}$. Therefore, given that the task set is harmonic, the combined utilization of the tasks on the core with the maximum load is $1 - U_{SleepMin}$. Let the synchronous forced sleep utilization for the optimal partition be $U_{SleepOpt}$. Hence, the combined utilization of the tasks on the core with the maximum load is $1 - U_{SleepOpt}$. Using the result in [33], the following must hold:

$$\frac{\frac{4}{3} - \frac{1}{m}}{U_{bound}}(1 - U_{SleepOpt}) = 1 - U_{SleepMin}$$

By substituting the ES-RMS utilization bound $U_{bound} = 1 - U_{SleepMin}$, we obtain:

$$U_{SleepOpt} = 1 - \frac{(1 - U_{SleepMin})^2}{(4/3) - (1/m)}$$

The above function is maximized as $m \to \infty$. Hence, for the case where the task set is *harmonic*, the approximation ratio for WFD under ES-RMS is:

$$\frac{U_{SleepOpt}}{U_{SleepMin}} = \frac{4 - 3(1 - U_{SleepMin})^2}{4U_{SleepMin}}$$

■

Theorem 6: For *SyncSleep* scheduling, the WFD heuristic for partitioning independent tasks onto a multi-processor, under ES-RMS, has an approximation ratio of at most:

$$\frac{4 - 3(ln2 - U_{SleepMin})^2}{4U_{SleepMin}}$$

Proof: Let WFD yield a partition with forced sleep utilization of $U_{SleepMin}$. For a set of n schedulable tasks the utilization bound for RMS is $n(2^{1/n} - 1)$. As $n \to \infty$, the utilization bound $U_{bound} \to ln2$. Therefore, the combined utilization of the tasks on the core with the maximum load is $ln2 - U_{SleepMin}$. Let the forced sleep utilization for the optimal partition be $U_{SleepOpt}$. Hence, in the best case, the maximum combined utilization of the tasks on the core with the maximum load is $1 - U_{SleepOpt}$. The theorem can now be proved using the technique used to prove Theorem 5. For the sake of brevity, the details are omitted. The plot for the approximation ratios derived are illustrated in Figure 2. ■

Given the non-linear nature of the schedulability tests for ES-RMS (and ES-RHS+), load balancing may not always be the best approach to obtain better energy savings for *SyncSleep* scheduling. This observation is illustrated by the following example. Consider a task set Γ which contains four tasks $\tau_1 = (40, 100)$, $\tau_2 = (40, 100)$, $\tau_3 = (105, 250)$ and $\tau_4 = (210, 500)$, which need to be scheduled on two cores M_1 and M_2. We assume that each core schedules tasks using ES-RMS. Hence, across both cores, $T_{sleep} = 50$ (here, $T_{sleep} = T_1/2$). Let us consider the following two cases:

Case 1: Tasks are assigned using WFD. τ_1 and τ_4 are assigned to M_1, and τ_2 and τ_3 are assigned to M_2. By using the ES-RMS schedulability bounds followed by *Binary "MaxSleep" Search*, we obtain $C_{sleep,1} = 9$ and $C_{sleep,2} = 5$. Hence, for the system, the deep sleep utilization would be $U_{sleep} = min(C_{sleep,1}, C_{sleep,2})/T_{sleep} = 5/50 = 0.1$.

Case 2: Tasks are assigned using an alternate scheme. τ_3 and τ_4 are assigned to M_1, and τ_1 and τ_2 are assigned to M_2. We now obtain $C_{sleep,1} = 8$ and $C_{sleep,2} = 10$. Hence, for the system, the deep sleep utilization would be $U_{sleep} = min(C_{sleep,1}, C_{sleep,2})/T_{sleep} = 8/50 = 0.16$.

This example illustrates that task utilization is not the only factor which affects the time for which the system can be put into deep sleep. Other factors such as the period of a task, and how it affects the schedulability of tasks already allocated, also plays an important role.

B. Max-SyncSleep Task Partitioning Heuristic

We now present a task partitioning heuristic for maximizing the synchronous forced sleep duration. *Max-SyncSleep* uses ES-RMS (or ES-RHS+) in conjunction with BMS, to improve the deep sleep utilization of a multi-core processor, so as to maximize energy savings. The pseudo-code for *Max-SyncSleep* can be found in Algorithm 2.

Let C_{sleep} be the duration of time for which the entire multi-core processor can transition into deep sleep state. At any instant of time, *Max-SyncSleep* first adds each unassigned task τ_i in the task set Γ to each of the m cores and computes a value δ_i for each task. When a task τ_i is added to a processor k, a value δ_{ik} is computed which reflects the change in the system C_{sleep}, after adding the task. For each task τ_i, the minimum of these $\delta_{ik} \forall k \in 1, 2...m$ values is assigned to δ_i. The task having the maximum weight δ_i is chosen to be the next task to be allocated. Compared to WFD, which considers only the utilization of the task for allocation, *Max-SyncSleep* considers the effect of both the period and the worst-case execution time of a task on C_{sleep}. The metric δ_i effectively captures this intuition, as it measures the impact of each unassigned task on the synchronous forced sleep duration. Once a task τ_i is selected to be assigned, we need to choose a core to allocate the task to. This allocation is done on the previously computed δ_{ik} values. The chosen τ_i is assigned to the core which provides the minimum $\delta_{ik} \forall k \in 1, 2...m$. The motivation behind this step lies in the fact that we allocate the task τ_i to a core, such that, after addition of the task, the decrease in C_{sleep} is minimized. During the execution of *Max-SyncSleep*, if a task is found which is not schedulable on any core, then the heuristic declares the task set as not schedulable. The complexity of the *Max-SyncSleep* heuristic is $O(n^2 m log_2(T_{sleep}/\epsilon))$.

VII. IndSleep Scheduling

For processors which enable cores to *independently* transition to deep sleep, *IndSleep* scheduling is possible, and enables greater energy savings over *SyncSleep* scheduling. On such processors, it is feasible to run ES-RHS+ on each core, so as to utilize all the idle durations to put the processor into deep sleep. We formally state the energy minimization (or deep sleep maximization) problem as follows:

Algorithm 2 Max-SyncSleep Task Partitioning Heuristic

procedure MAX-SYNCSLEEP ($\Gamma, C_{SleepMin}, m$)
  $\Gamma_{unassigned} \leftarrow \Gamma$ ▷ Unassigned tasks
  $C_{sleep} = T_{sleep}$ ▷ Deep Sleep time
  for $k = 1$ to m **do**
    $\Gamma_{assigned,k} \leftarrow \phi$ ▷ Tasks assigned to core k
  end for
  while $\Gamma_{unassigned}$ is non empty **do**
    $\delta_{max} \leftarrow -\infty$
    for $i = 1$ to $Cardinality(\Gamma_{unassigned})$ **do**
      $\tau \leftarrow \Gamma_{unassigned}[i]$; $\delta_{min} \leftarrow \infty$
      for $k = 1$ to m **do**
        Add τ to $\Gamma_{assigned,k}$
        $TestSchedulability(\Gamma_{assigned,k})$
        if Schedulable **then**
          $\delta = C_{sleep} - BMS(\Gamma_{assigned,k})$
          if $\delta \leq \delta_{min}$ **then**
            $\delta_{min} \leftarrow \delta$; $\alpha \leftarrow k$
          end if
        end if
        Remove τ from $\Gamma_{assigned,k}$
      end for
      if τ not schedulable on any core **then**
        return Γ not-schedulable
      end if
      if $\delta_{min} \geq \delta_{max}$ **then**
        $task \leftarrow \tau$; $core \leftarrow \alpha$; $\delta_{max} \leftarrow \delta_{min}$
      end if
    end for
    Add $task$ to $\Gamma_{assigned,core}$
    Remove $task$ from $\Gamma_{unassigned}$
    $C_{sleep} \leftarrow Min[BMS(\Gamma_{assigned,k}) \forall k \in (1, m)]$
  end while
  return $C_{sleep}, \Gamma_{assigned,k} \forall k \in (1, m)$
end procedure Max-SyncSleep

Consider a task set Γ, consisting of n periodic real-time tasks $\tau_1, \tau_2, ..., \tau_n$ that need to be scheduled on a homogeneous multi-core processor consisting of m cores, $M_1, M_2, ..., M_m$. Each core M_k has a forced sleep execution, $\tau_{sleep,k}$ where $k = 1, 2, ..., m$. $\tau_{sleep,k}$ has a forced sleep duration of $C_{sleep,k}$ every $T_{sleep,k}$. Our objective is to find a partition that maximizes the sum of the forced sleep durations on each of the cores such that the workload allocated to each core can be scheduled by ES-RHS+ in a feasible manner.

For every $\tau_{sleep,k}$, where $k = 1, 2, ..., m$, the period T_{sleep} can be chosen based on the tasks allocated to core k. While using ES-RHS+ (or ES-RHS) on each core, $C_{sleep,k} \geq C_{SleepMin}$ must hold, in order to utilize all the processor idle durations. If $C_{sleep,k} < C_{SleepMin}$, then that core cannot transition into deep sleep. Hence, for *IndSleep* scheduling using ES-RHS+, we can define a feasible partition as one in which each core has a forced sleep execution, $C_{sleep,k} \geq C_{SleepMin}$. We now prove that using ES-RHS+ optimally solves the *IndSleep* scheduling problem.

Theorem 7: If a task set Γ can be feasibly scheduled on m cores using ES-RHS+, with *IndSleep* scheduling, then in *any* such feasible partition, all idle durations can be used to put the processor into deep sleep.

Proof: Let the total utilization of the tasks in Γ be $U_{taskset}$. Consider a feasible partition, such that Γ is partitioned into m disjoint sets $\Gamma_k, k \in 1, 2, ..m$, where tasks $\tau_{i,k} \in \Gamma_k$ are scheduled on core k. Let the worst-case utilization of each task be $u_{i,k}$. Given that ES-RHS+ is used on each core, the deep sleep utilization on each core is:

$$U_{sleep,k} = 1 - \sum_{i=1}^{|\Gamma_k|} u_{i,k}$$

where, $|\Gamma_k|$ is the cardinality of the set Γ_k. Hence, the deep sleep utilization of the system can be given by:

$$U_{sleep} = \sum_{k=1}^{m} U_{sleep,k} = \sum_{k=1}^{m} (1 - \sum_{i=1}^{|\Gamma_k|} u_{i,k})$$
$$= m - \sum_{k=1}^{m} \sum_{i=1}^{|\Gamma_k|} u_{i,k} = m - U_{taskset}$$

According to the theorem, using ES-RHS+ for *IndSleep* scheduling provides a lot of flexibility in task allocation, since *any* feasible partition is optimal from an energy-saving perspective. ■

Hence, we can obtain a feasible and optimal partition by using partitioning heuristics like WFD or *Max-SyncSleep*, by enforcing that, for each core k, $C_{sleep,k} = C_{SleepMin}$.

VIII. COMPARATIVE EVALUATIONS

In this section, we assess the performance of ES-RHS+, ES-RMS and *Max-SyncSleep*. We compare different techniques on the basis of schedulability and energy savings, with respect to the total utilization of the task set. Experiments have been performed using randomly generated task sets. Each task is randomly assigned a period between 20 and 400 time units, and a random worst-case computation time such that the per-task utilization is always below 25%. The minimum supported forced sleep execution, $C_{SleepMin}$ is set to 10 time units. The number of tasks used in the experiment depends on the target utilization of the task set. Each data point we plot is an average of 1000 randomly generated task-sets with the same utilization. For each task set, the maximum possible *forced sleep* execution, C_{sleep} is calculated using the BMS technique.

Note: The plots in this section are best viewed in color.

A. Uniprocessor Comparisons

In the uniprocessor context, we compare ES-RHS, ES-RHS+ and ES-RMS on the basis of schedulability, and the utilization of the forced sleep execution $U_{sleep} = C_{sleep}/T_{sleep}$.

Figure 3 shows how the schedulability of the compared techniques changes as the task set utilization is varied. In terms of schedulability; ES-RMS > ES-RHS+ > ES-RHS. From the plot, we can observe that ES-RMS can schedule upto 33% more task sets than ES-RHS+.

Figure 4 shows how the utilization of the forced sleep execution changes with the utilization of schedulable task

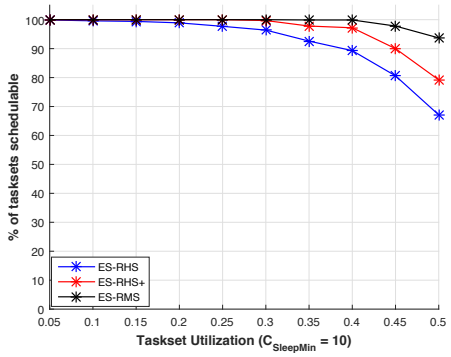

Fig. 3. Percentage of task sets schedulable with respect to task set utilization, in the uniprocessor context

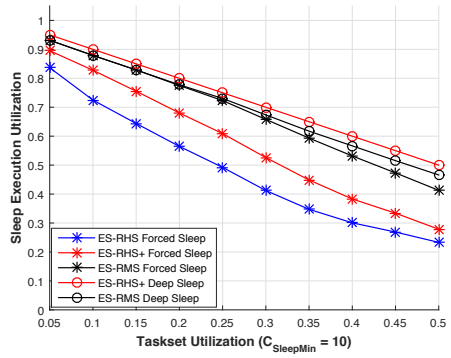

Fig. 4. Utilization of the forced sleep and total deep sleep with respect to task set utilization, in the uniprocessor context

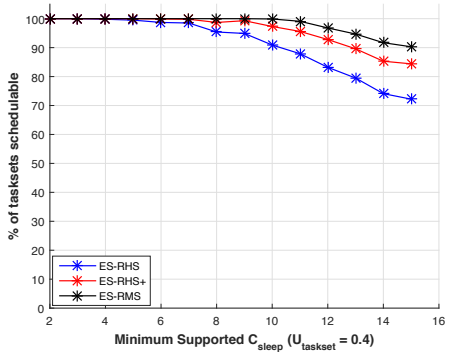

Fig. 5. Percentage of task sets schedulable with respect to the minimum supported sleep duration, $C_{SleepMin}$, in the uniprocessor context

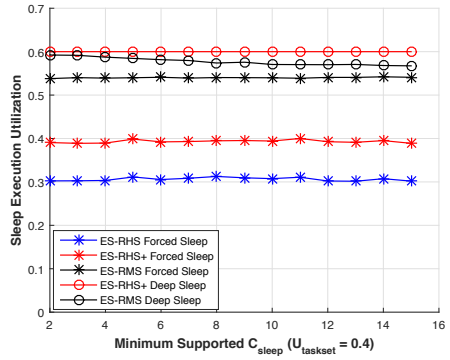

Fig. 6. Utilization of the forced sleep and total deep sleep with respect to the minimum supported sleep duration, $C_{SleepMin}$, in the uniprocessor context

sets. Again, observe that ES-RMS always outperforms both ES-RHS+ and ES-RHS. For schedulable task sets, ES-RMS can provide up to 18% more forced sleep execution than ES-RHS+. This validates the use of ES-RMS for *SyncSleep* scheduling. By performing simulations over the hyperperiod of the task sets, we also compare ES-RHS+ and ES-RMS on the basis of the total deep sleep utilization achieved. Observe that while ES-RHS+ provides optimal energy savings, ES-RMS comes very close to the optimal.

In Figures 5 and 6, the schedulability and sleep utilization are plotted as a function of $C_{SleepMin}$, when the utilization of the task sets is kept constant, $U_{taskset}$ = 0.4, and $C_{SleepMin}$ is varied from 2 to 15. Observe that the forced sleep utilization is not sensitive to $C_{SleepMin}$. However, the schedulability of the compared techniques degrades, as $C_{SleepMin}$ increases.

B. SyncSleep Scheduling

For multi-core *SyncSleep* scheduling, we compare ES-RHS, ES-RHS+ and ES-RMS on the basis of schedulability, and the utilization of the synchronous forced sleep execution (U_{sleep} = C_{sleep}/T_{sleep}). We consider each of these techniques using both WFD and *Max-SyncSleep* for task partitioning.

In Figures 7 and 9, we compare the schedulability of the techniques as a function of the task set utilization, for a quad-core (m = 4) and an octa-core (m = 8) processor respectively. In terms of schedulability; ES-RMS > ES-RHS+ > ES-RHS. In the average case, combining any of these techniques with the *Max-SyncSleep* partitioning technique yields significantly better schedulability than WFD. In both the quad-core and octa-core cases, observe that the schedulability of the WFD-based techniques significantly decreases as the number of cores/task set utilization increases.

Figures 8 and 10 show how the utilization of the synchronous forced sleep execution varies as a function of the utilization of schedulable task sets for a quad-core (*m=4*) and an octa-core (*m=8*) processor respectively. Observe that ES-RMS always outperforms both ES-RHS+ and ES-RHS in this respect. Using *Max-SyncSleep* with ES-RMS, guarantees up to 14% more synchronous forced sleep than *Max-SyncSleep* with ES-RHS+, in the octa-core case. Also, note that using *Max-SyncSleep* provides significantly larger synchronous forced sleep durations than WFD, thus leading to greater energy savings. The energy savings are greater, as the number of cores/utilization of the task set increase: up to 57% more for ES-RMS with *Max-SyncSleep* compared to ES-RMS with WFD for the octa-core case. Hence, using *Max-SyncSleep* provides both better schedulability and energy savings. Note that for some task set utilization values, using ES-RHS+ or ES-RHS with WFD does not yield any schedulable task sets. For these cases, the obtained synchronous forced sleep utilization is shown as zero.

C. IndSleep Scheduling

For *IndSleep* scheduling, we compare ES-RMS and ES-RHS+. For a feasible partition, ES-RHS+ is provably optimal in this context. Hence, we use *Max-SyncSleep* to generate

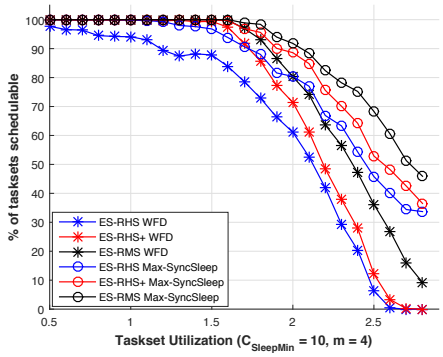

Fig. 7. Percentage of task sets schedulable with respect to task set utilization, in the multi-processor *SyncSleep* scheduling context (m = 4)

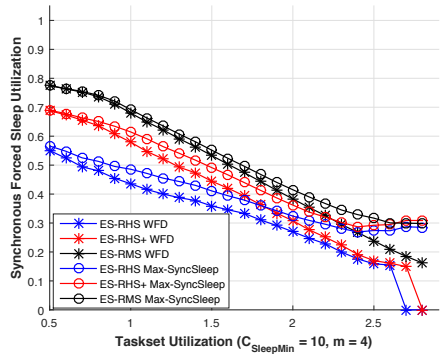

Fig. 8. Utilization of the synchronous forced sleep execution versus task set utilization, in the multi-processor *SyncSleep* scheduling context (m = 4)

a feasible partition for both ES-RHS+ and ES-RMS. Figure 11 shows the guaranteed deep sleep utilization as a function of task set utilization for a quad-core processor (m = 4). The straight line plot obtained for ES-RHS+ indicates that it optimally guarantees that all idle durations can be used to put the processor into *deep sleep*. The additional guaranteed sleep utilization obtained is up to 28% more than ES-RMS.

IX. Conclusions

In this paper, we have presented fixed-priority partitioned scheduling techniques, which utilize processor sleep states to save energy. In the uniprocessor context, we presented an enhanced version of ES-RHS. By re-defining the notion of harmonization [10], ES-RHS+ provides enhanced schedulability over ES-RHS, while still guaranteeing that all idle durations can be spent in deep sleep state. In the multi-core context, we identified two classes of scheduling problems: *SyncSleep Scheduling*, where cores need to synchronously transition to deep sleep, and *IndSleep Scheduling*, where cores can independently transition to deep sleep.

For *SyncSleep* scheduling, we proposed and used an energy saving version of RMS called ES-RMS. ES-RMS provides greater energy savings (i.e., longer synchronous forced sleep durations), as well as better schedulability than ES-RHS+. To maximize the synchronous forced sleep execution, it is necessary to balance the load across cores. We proved the approximation ratio of WFD using ES-RMS as a function of the minimum possible deep sleep utilization. We then showed

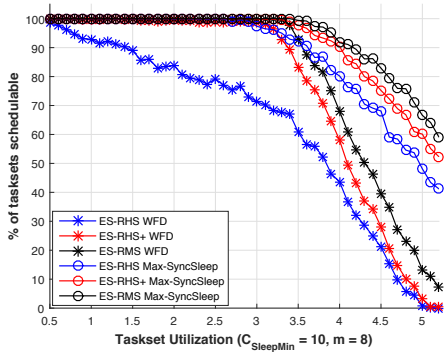

Fig. 9. Percentage of task sets schedulable with respect to task set utilization, in the multi-processor *SyncSleep* scheduling context (m = 8)

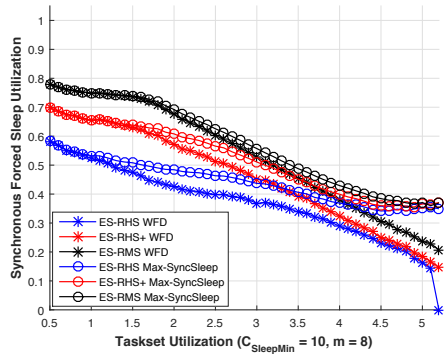

Fig. 10. Utilization of the synchronous forced sleep execution versus task set utilization, in the multi-processor *SyncSleep* scheduling context (m = 8)

that WFD does not always result in a partition with good energy savings, and proposed the *Max-SyncSleep* partitioning heuristic, which obtains significantly better schedulability and energy savings over WFD, by taking into account the impact of individual tasks on the synchronous forced sleep. In the *IndSleep* scheduling context, we proved that for any feasible partition, using ES-RHS+ on each core, optimally uses all the idle durations to put the processor into deep sleep.

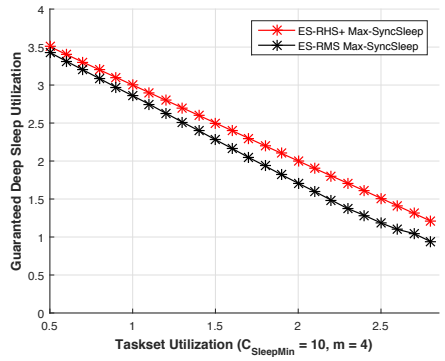

Fig. 11. Guaranteed deep sleep utilization versus task set utilization, in the multi-processor *IndSleep* scheduling context (m = 4)

In addition to power, thermal constraints have also become a major cause of concern in modern processors. As part of future work, we will look at leveraging a combination of frequency scaling and sleep states for power and temperature management of multi-core processors.

REFERENCES

[1] T. Kuroda, "CMOS design challenges to power wall", in *Proc. of 2001 International Microprocesses and Nanotechnology Conference*, November 2001

[2] "Samsung Exynos 5800," [Online]. Available:http://www.samsung.com /semiconductor/minisite/Exynos/w/solution.html#?v=octa_5410

[3] "Qualcomm Snapdragon 810," [Online]. Available:https://www. qualcomm.com /documents/snapdragon-810-processor-product-brief, 2014

[4] "Rasberry Pi 2 Model B," [Online]. Available:https://www.raspberrypi.org /products/ raspberry-pi-2-model-b/, 2015

[5] H. L. Shi, K. M. Hou, H. Zhou and X. Liu, "Energy Efficient and Fault Tolerant Multicore Wireless Sensor Network: EMWSN," in *Proc. 7th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM)*, September 2011

[6] M. Flannagan, "Fog Computing: The New Model for the IoE," [Online]. Available: http://blogs.cisco.com/datacenter/fog-computing-the-new-model-for-the-ioe, January 2015

[7] "Google Glass," [Online]. Available:https://support.google.com/glass/ answer/3064128?hl=en&ref_topic=3063354, Google Inc., April 2013

[8] F. Yao, A. Demers and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. 36th Annual Symposium on Foundations of Computer Science*, October 1995

[9] K. Agarwal, K. Nowka, H. Deogun and D. Sylvester, "Power Gating with Multiple Sleep Modes," in *Proceedings of the 7th International Symposium on Quality Electronic Design*, 2006

[10] A. Rowe, K. Lakshmanan, H. Zhu and R. Rajkumar, "Rate-Harmonized Scheduling and Its Applicability to Energy Management," in *IEEE Transactions on Industrial Informatics*, vol. 6, no. 3, August 2010

[11] S. Saewong and R. Rajkumar, "Practical voltage-scaling for fixed-priority RT-systems," in *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS03)*, 2003

[12] A. Hakan and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, 2003

[13] V. Devadas and H. Aydin, "Coordinated power management of periodic real-time tasks on chip multiprocessors," in *Proceedings of the International Conference on Green Computing*, 2010

[14] A. Kandhalu, J. Kim, K. Lakshmanan and R. Rajkumar, "Energy-Aware Partitioned Fixed-Priority Scheduling for Chip Multi-processors," in *Proceedings of IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA '11)*, August 2011

[15] T. AlEnawy and H. Aydin, "Energy-aware task allocation for rate monotonic scheduling," in *Proceedings of 11th Real Time and Embedded Technology and Applications Symposium (RTAS '05)*, March 2005

[16] J. J. Chen, C. Y. Yang, H. I. Lu and T. W. Kuo, "Approximation algorithms for multiprocessor energy-efficient scheduling of periodic real-time tasks with uncertain task execution time," in *Proceedings of Real-Time and Embedded Technology and Applications Symposium (RTAS 08)*, April 2008

[17] N.S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M.J. Irwin, M. Kandemir and V. Narayanan, "Leakage current: Moore's law meets static power," in *Computer*, vol.36, no.12, pp.68-75, December 2003

[18] E. Le Sueur and G. Heiser, "Slow down or sleep, that is the question," in *Proceedings of the 2011 USENIX annual technical conference (USENIX ATC'11)*, 2011

[19] J. J. Chen and T. W. Kuo, "Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor," in *Proc. ACM SIGPLAN/SIGBED LCTES*, pp. 153162, 2006

[20] S. Irani, S. Shukla, and R. Gupta, "Algorithms for power savings," in *Proc. 14th Annu. ACM-SIAM Symp. Discrete Algorithms*, pp. 3746, 2003

[21] R. Jejurikar and R. K. Gupta, "Procrastination scheduling in fixed priority real-time systems," in *Proc. 2004 CM SIGPLAN/SIGBED Conf. Languages, Compilers, and Tools for Embedded Systems*, pp. 5766, 2004

[22] L. Niu and G. Quan, "Reducing both dynamic and leakage energy consumption for hard real-time systems," in *Proc. 2004 Int. Conf. Compilers, Architecture and Synthesis for Embedded Systems*, pp. 140148, 2004

[23] J. Chen and T. Kuo, "Procrastination determination for periodic realtime tasks in leakage-aware dynamic voltage scaling systems," in *Proc. 2007 IEEE/ACM Int. Conf. Comput.-Aided Design*, pp. 289294, 2007

[24] J. Chang, H. N. Gabow, and S. Khuller. "A model for minimizing active processor time," *AlgorithmsESA*, 289-300, 2012

[25] C. Fu, Y. Zhao, M. Li, and C.J. Xue, "Maximizing common idle time on multi-core processors with shared memory," in *Proc, Design, Automation & Test in Europe Conference & Exhibition*, 2015.

[26] R. Rajkumar and S. Oikawa, "A portable resource kernel for guaranteed and enforced timing behavior," in *Proc. IEEE Real-Time Technol. Appl. Symp. (RTAS)*, 1999.

[27] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. ACM*, vol. 20, no. 1, pp. 4661, 1973.

[28] N. Audsley, A. Burns, M. Richardson, K. Tindell and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284-292, 1993.

[29] L. Sha, R. Rajkumar, J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers*, Vol. 39, No. 9, 1990

[30] "Intel Core 2 Duo Extreme," [Online]. Available: download.intel.com/design/processor/datashts/ 320390.pdf, January 2009

[31] "AMD Opteron Family 10h," [Online]. Available:http://support.amd.com/TechDocs/40036.pdf, June 2010

[32] "4th Generation Intel Core Processor Family," [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/4th-gen-core-family-desktop-vol-1-datasheet.pdf, March 2015

[33] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal on Applied Mathematics*, 17:416429, 1969.