Thermal Implications of Energy-Saving Schedulers

Sandeep M. D'souza[1] and Ragunathan (Raj) Rajkumar[2]

1 **Carnegie Mellon University, Pittsburgh, PA, USA**
sandeepd@andrew.cmu.edu

2 **Carnegie Mellon University, Pittsburgh, PA, USA**
rajkumar@andrew.cmu.edu

Abstract

In many real-time systems, continuous operation can raise processor temperature, potentially leading to system failure, bodily harm to users, or a reduction in the functional lifetime of a system. Static power dominates the total power consumption, and is also directly proportional to the operating temperature. This reduces the effectiveness of frequency scaling and necessitates the use of sleep states. In this work, we explore the relationship between energy savings and system temperature in the context of fixed-priority *energy-saving* schedulers, which utilize a processor's *deep-sleep* state to save energy. We derive insights from a well-known thermal model, and are able to identify *proactive* design choices which are *independent* of system constants and can be used to reduce processor temperature. Our observations indicate that, while energy savings are key to lower temperatures, not all energy-efficient solutions yield low temperatures. Based on these insights, we propose the *SysSleep* and *ThermoSleep* algorithms, which enable a thermally-effective sleep schedule. We also derive a lower bound on the optimal temperature achievable by energy-saving schedulers. Additionally, we discuss partitioning and task phasing techniques for multi-core processors, which require all cores to synchronously transition into deep sleep, as well as those which support independent deep-sleep transitions. We observe that, while energy optimization is straightforward in some cases, the dependence of temperature on partitioning and task phasing makes temperature minimization non-trivial. Evaluations show that compared to the existing purely energy-efficient design methodology, our proposed techniques yield lower temperatures along with significant energy savings.

1998 ACM Subject Classification C.3 Real-Time and Embedded Systems

Keywords and phrases thermal analysis, real-time scheduling

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2017.21

1 Introduction

Computationally-intensive real-time applications are becoming ubiquitous. Autonomous vehicles are a prime example where, computational requirements are driven by the need to process streams of data from multiple sensors. Advancements in semiconductor technology have enabled such applications by increasing the number of transistors available to system designers. However, the side effects of rising transistor density include increased power and heat dissipation [23]. Hence, continuous operation may cause the temperature of a processor to exceed its operating limits, forcing it to reduce its frequency or shut down. This in turn can lead to missed deadlines, and possibly catastrophic failure. Similarly, violating thermal constraints in implantable medical devices can cause bodily harm [8]. Moreover, it is critical that the components used in such systems perform reliably over their lifetime. System temperature is one of the key factors which influence reliability. High temperatures degrade

29th Euromicro Conference on Real-Time Systems (ECRTS 2017).
Editor: Marko Bertogna; Article No. 21; pp. 21:1–21:23
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the system reliability over a period of time [31][32], and a 10-15°C difference in operating temperature can result in a 2x difference in the lifespan of a device [32].

Energy savings and system temperature are intricately tied together. Modern processors are equipped with energy-management features such as Dynamic Voltage and Frequency Scaling (DVFS) [35], and the use of low-power sleep states [28]. DVFS enables the processor to change its operating frequency and voltage, thereby reducing *dynamic switching power*, while low-power sleep states use power gating and/or clock gating [3] to reduce *static leakage power* dissipation when the processor is idle. As transistor geometries get smaller, the dominance of static power as a contributor to total power consumption is only expected to increase [22]. Since static power is also directly dependent on operating temperature, scheduling techniques will increasingly need to take advantage of processor sleep states.

1.1 Contributions of the Paper

In this work, we analyze the thermal properties of Energy-Saving (ES) Schedulers [12], which utilize the processor's deep-sleep state. Our contributions are as follows:

1. We analyze the thermal performance of ES Schedulers using the well-known thermal model based on Fourier's Law, and derive design choices to pro-actively (i.e. *a priori*) minimize the maximum temperature for both uni-core and multi-core processors.
2. We present the *SysSleep* algorithm to maximize the time the processor can be in deep sleep, and the *ThermoSleep* heuristic that yields a thermally-effective sleep schedule.
3. We derive a lower bound on the optimal maximum temperature achievable by ES Schedulers.
4. We propose task-partitioning heuristics that significantly reduce the maximum temperature for multi-core processors using ES Schedulers.
5. We analyze the impact of phasing each core's forced-sleep task on temperature, in the context of multi-core processors where cores can independently transition into deep sleep.

1.2 Background

We now introduce the background material and notation relevant to our work. Consider a task set Γ consisting of n independent[1] periodic real-time tasks $\tau_1, \tau_2, ..., \tau_n$. Each task $\tau_i \in \Gamma$ is 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. We assume that for each task $D_i = T_i$, i.e., deadlines are implicit. The utilization of a task τ_i is given by $U_i = C_i/T_i$ and task priorities are assigned using the rate-monotonic scheduling policy [27]. The task set is listed in non-increasing order of task priorities such that $T_1 \le T_2 \le .. \le T_n$. Each task has an initial arrival time (or phase) 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 following Energy-Saving (ES) Schedulers have been defined in [28] and [12]: Energy-Saving Rate-Harmonized Scheduling+ [28][12] (ES-RHS+), Energy-Saving Rate-Monotonic Scheduling [12] (ES-RMS) and Energy-Saving Deadline-Monotonic Scheduling [12] (ES-DMS). These techniques are characterized by a high-priority periodic Energy-Saver task (also referred to as an ES-task or forced-sleep task) τ_{sleep}, which puts the processor into an uninterrupted deep sleep for a duration $C_{sleep} \ge C_{SleepMin}$ every period $T_{sleep} \le T_1$. This ensures that the ES-task executes at the highest priority in accordance with the Rate-Monotonic (RM) [27]

[1] Task release jitter and task dependence can be incorporated using the frameworks proposed in [6] and [30], and are beyond the scope of this work.

priority assignment. If any idle durations precede and are contiguous with the ES-task, they can be used to put the processor into deep sleep [12]. $C_{SleepMin}$ is a system constraint that represents the minimum round-trip time required for the processor to go into the deep-sleep state and return back to the active state. We assume that $C_{SleepMin}$ captures the overhead involved in transitioning to deep sleep. While using ES Schedulers, the processor can be in one of the following states:

- *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*.

For ES Schedulers, the generalized worst-case response time test for a task τ_i is given by the following recurrence relation:

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

where, 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, where, D'_i is the *generalized deadline* of a task τ_i and depends on the type of ES Scheduler used. Based on this notation, we briefly describe each of the ES Schedulers:

(1) ***ES-RMS***: Tasks execute as per rate-monotonic priorities and deadlines are assumed to be implicit ($D_i = T_i$). Here, the generalized deadline, $D'_i = T_i$.

(2) ***ES-DMS***: Tasks execute as per deadline-monotonic priorities. This implies that the generalized deadline, $D'_i = D_i$.

(3) ***ES-RHS+***: Tasks execute as per rate-monotonic priorities, and deadlines are implicit. However, tasks become eligible to execute based on the principle of *harmonization*: A task is eligible to execute only when the processor is *busy* or a *Harmonizing Period* boundary has been reached [12]. The use of harmonization enables every *idle* duration in the ES-RHS+ schedule to precede and be contiguous with the ES-task. Hence, all the processor's idle durations can be utilized to put it into deep sleep, thereby providing maximal energy savings [12]. Due to harmonization, each task can be delayed by at most $T_{sleep} - C_{sleep}$ [12]. This implies that the generalized deadline, $D'_i = T_i - (T_{sleep} - C_{sleep})$, and provides a tight schedulability test compared to the slightly looser one proposed in [12]. An example schedule for ES-RHS+ and ES-RMS using a taskset with 3 tasks is illustrated in Figure 1.

Multi-core processors also support a number of low-power states called C-states. In some processors, individual cores can transition to intermediary *idle* states. However, in many processors, cores cannot individually transition into deep sleep. Based on the ability to transition into deep sleep, two types of problems were defined in [12] for ES Schedulers:

1. *Synchronized-Sleep* or *SyncSleep Scheduling* where, all cores transition *synchronously* into deep sleep. Example processors include Intel Core^2 Duo [13] and AMD Opteron [15].
2. *Independent-Sleep* or *IndSleep Scheduling* where, each core can independently transition into deep sleep. Example processors with this flexibility include Samsung Exynos 5800 [2] and the 4th generation Intel Core processors [1].

In the SyncSleep context, only for the idle durations that overlap across all cores and exceed $C_{SleepMin}$ can the processor be put into deep sleep. Given the same T_{sleep}, ES-RMS can guarantee higher forced-sleep utilization U_{sleep} than ES-RHS+ [12]. This makes ES-RMS a better choice for SyncSleep [12]. For IndSleep, it was proved that using ES-RHS+ can yield an energy-optimal schedule *for all* feasible partitions [12].

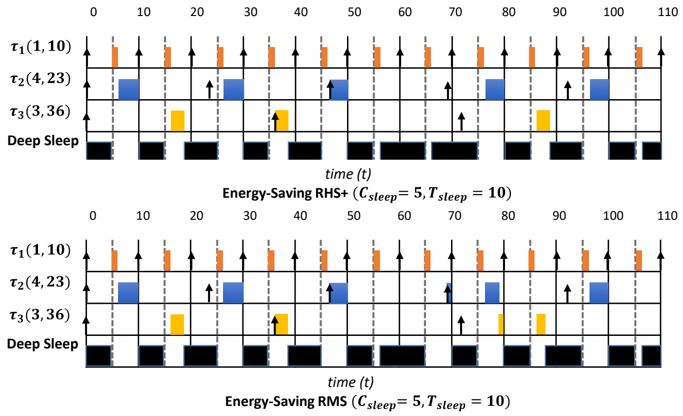

Figure 1 Energy-Saving Schedulers: ES-RHS+ & ES-RMS ($C_{sleep} = 5, C_{SleepMin} = 5, T_{sleep} = 10$).

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 introduces the thermal model used in the paper. Section 4 introduces the *SysSleep* algorithm, and discusses utilizing ES Schedulers for reducing temperature in uni-core processors. Section 5 discusses utilizing ES Schedulers for reducing temperature in multi-core processors. Section 6 presents comparative evaluations, and Section 7 provides concluding remarks.

2 Related Work

Thermal Management can be done reactively at runtime [8, 17, 11, 36] or proactively at design time [20, 5, 16, 33, 10, 9, 4]. In the scope of reactive techniques, Fu. et al. [17] proposed a control-theoretic algorithm to meet the desired temperature requirement on a multi-core processor, subject to timing constraints. Yun et al. [36] used a machine-learning technique (SVM) to predict the temperature profile of a multi-processor system. Based on the predicted value, a dynamic temperature management scheme is used. In [8], Chandarli et al. proposed an optimal reactive scheduler for fixed-priority uniprocessor sleep scheduling along with an associated response-time based analysis framework. However, reactive schedulers require temperature sensors, which may not always be present in real platforms.

In the scope of proactive techniques, [10] describes a real-time scheduling algorithm for uniprocessors, based on a thermal model approximated by Fourier's Law. The algorithm derives a speed schedule by minimizing temperature under both timing and thermal constraints. In [9], an assignment and scheduling technique for an MPSoC was proposed, which utilizes a mixed-integer linear program solver to optimize the peak temperature. In [16], an optimal speed schedule is derived for a multi-core platform, based on a thermal model given at design time. In [4], Masud et al. proposed the use of a thermal-aware periodic resource to minimize peak temperature, in the context of uniprocessor Earliest Deadline First (EDF) scheduling. The processor slack is utilized to put the processor into a sleep state.

Most of the pieces of work stated [17, 11, 36, 16, 20, 5] have focused on the use of DVFS to optimize the processor temperature. However, the dominance of static power makes it necessary to investigate techniques which utilize sleep states. Additionally, many low-powered devices often lack DVFS, but support sleep states [28]. The work in [8] and [4] propose thermal-aware techniques which utilize processor sleep states. However, [4] assumes dynamic-priority EDF scheduling. On the other hand, [8] presents a reactive framework for uniprocessor fixed-priority scheduling. To the best of our knowledge, no thermal analysis framework for proactive fixed-priority sleep scheduling exists in the literature.

Fixed-priority energy-saving schedulers, which periodically utilize the processor's deep-sleep state, were proposed in [28][12]. For these schedulers, the work in [12] proposed various techniques to design energy-efficient schedules in both the uni-core and multi-core processor contexts. In this paper, we analyze the thermal implications of ES Schedulers in light of their energy-saving properties. Based on a well-known thermal model, we derive practical insights and algorithms. Our proposed techniques focus on minimizing the maximum temperature, rather than optimizing to meet a set of thermal constraints.

3 Thermal Modeling of ES Schedulers

In this section, we introduce the thermal model used in the paper, and derive insights in the context of ES Schedulers. The temperature of a processor is dependent on the power consumption, and the variation in power consumption over time. Therefore, we can broadly define three factors responsible for a processor's thermal profile: (i) Heat generation by a core (due to power consumption). (ii) Heat dissipation to the environment (using heat sinks). (iii) Heat dissipation between adjacent cores (due to difference in power consumption patterns).

3.1 Power and Thermal Model

The power consumption of a CMOS circuit is modeled as a combination of two components:

1. *Dynamic Switching Power* is dependent on the processor operating frequency, and is consumed when the processor is *busy*. The dynamic power consumption, P_D, can be modeled as a convex function of the operating frequency s as [8]: $P_D = \kappa_0 s^\alpha$ where, α and κ are system constants which depend on the semiconductor technology used.
2. *Static Leakage Power* is due to leakage current, which depends on the semiconductor technology and the operating temperature. Static power is consumed even when the processor is *idle*, but can be nearly eliminated by putting the processor into *deep sleep*. Static power, P_S, can be conservatively modeled as a linear function of temperature [8]: $P_S = \kappa_1 \Theta + \kappa_2$ where, κ_1 and κ_2 are technology-dependent system constants, and Θ is the operating temperature.

Hence, the total power consumption P, as a function of time t, can be modeled as: $P(t) = P_D(t) + P_S(t)$. This model can be used to derive the thermal model for a uniprocessor. As OS schedulers control task execution at the granularity of a processor core, each core can be treated as a single unit producing heat and can be modeled as an *RC* circuit [8] [37]. When a core is *busy*, it generates heat. Using the RC thermal model, Fourier's Law [8] can be used to state the differential equation of the temperature, Θ^* with respect to time:

$$d\Theta^*(t)/dt = [P(t)/C] - [(\Theta^*(t) - \Theta_A)/RC] \tag{2}$$

where, Θ_A is the ambient temperature of the environment. By substituting P_D and P_S in Equation 2, we can rewrite Equation 2 as a classical linear differential equation [8]:

$$d\Theta(t)/dt = a - b\Theta(t) \tag{3}$$

where, $a = \kappa_0 s^\alpha / C$, $b = (1 - \kappa_1 R)/RC$ and the temperature has been offset from $\Theta^*(t) - [(\kappa_2 R + \Theta_A)/(1 - \kappa_1 R)]$ to $\Theta(t)$. Solving Equation 3 gives the temperature at time t as:

$$\Theta(t) = a/b + (\Theta(t_0) - a/b)e^{-b(t-t_0)}\,. \tag{4}$$

When the processor is in deep sleep, the power consumption can be assumed to be negligible. This is a valid assumption as the difference in power consumption between the busy and deep-sleep states is different by several orders of magnitude [28]. Hence, the processor can be deemed to be cooling when in the deep-sleep state. Using this assumption, one can set $a = 0$ in Equation 4 to obtain the model for cooling:

$$\Theta(t) = \Theta(t_0)e^{-b(t-t_0)}. \tag{5}$$

3.2 Thermal-Aware ES Scheduler Design

Consider a uni-core processor. For ES Schedulers, the processor is *guaranteed* to be in deep sleep atleast for a duration C_{sleep} every T_{sleep}. Hence, in the worst case, a core is *busy* for a duration of $T_{sleep} - C_{sleep}$ every T_{sleep}. Therefore, in the worst case, a processor core heats up from kT_{sleep} to $kT_{sleep} + C_{sleep}$ and cools down from $kT_{sleep} + C_{sleep}$ to $(k+1)T_{sleep}$, where k is a non-negative integer. As the heating function is monotonic in the period T_{sleep}, the temperature would be maximum at the end of the heating duration. We call this temperature Θ_{max}. Similarly, as the cooling function is monotonic in the period T_{sleep}, the temperature would be minimum at the end of the cooling duration. We call this temperature Θ_{min}. Applying the heating and cooling models from Equations 4 and 5 in the duration $[kT_{sleep}, (k+1)T_{sleep})$, we can write Θ_{max} and Θ_{min} as recurrent equations:

$$\Theta^k_{max} = a/b + (\Theta^{k-1}_{min} - a/b)e^{-b(Tsleep-Csleep)},\ \Theta^k_{min} = \Theta^k_{max}e^{-bC_{sleep}}. \tag{6}$$

At steady state, as $k \to \infty$, then $\Theta^k_{min} = \Theta^{k-1}_{min}$ and $\Theta^k_{max} = \Theta^{k-1}_{max}$. Hence, the steady state worst-case values of Θ_{max} and Θ_{min} are given by:

$$\Theta_{min} = (a/b) * [(e^{bT_{sleep}(1-U_{sleep})} - 1)/(e^{bT_{sleep}} - 1)], \Theta_{max} = \Theta_{min}e^{bU_{sleep}T_{sleep}} \tag{7}$$

where, $U_{sleep} = C_{sleep}/T_{sleep}$ denotes the *guaranteed* utilization of the ES-task. Based on the steady state temperatures, we can draw the following conclusions:

- Increasing U_{sleep}, keeping T_{sleep} constant, decreases the maximum temperature Θ_{max}.
- Decreasing T_{sleep}, keeping U_{sleep} constant, decreases the maximum temperature Θ_{max}.

Hence, minimizing T_{sleep}, while maximizing U_{sleep}, leads to a low maximum temperature. Thus, while it is advantageous to increase the total fraction of time the processor cools, i.e. $U_{sleep} \uparrow$ (also increases guaranteed energy savings), the cooling durations should be smaller but more frequent, i.e. $T_{sleep} \downarrow$. Note that these statements hold regardless of the system's thermal constants. Hence, using these principles, we can design techniques which can be used to minimize the temperature across a range of different systems.

In prior work [28][12], it was assumed that the period of the ES-task is a sub-harmonic of the highest-priority task. In the following section, we relax this constraint and provide techniques to design a thermally-effective ES schedule. Additionally, we show how choosing a proper T_{sleep} can maximize energy savings and improve schedulability.

4 SysSleep Algorithm

Consider a uni-core processor. To lower the worst-case maximum temperature for a taskset, we need to find an ES-task with a small period T_{sleep}, which also maximizes U_{sleep}. Maximizing U_{sleep} corresponds to finding the maximum *highest-priority* workload that can be added to a taskset without making it unschedulable. In [29], Saewong et al. proposed the SysClock algorithm which calculates the lowest processor frequency at which all tasks (with RM/DM priority assignment) meet their deadlines. SysClock calculates the slack at all scheduling points in the critical zone [24] to determine the optimal operating frequency. We extend that algorithm in the context of ES Schedulers, and use it to compute the set of T_{sleep} values which maximize U_{sleep}. Our algorithm is called *SysSleep*, and its pseudo-code is presented in Algorithm 1. We illustrate the working of *SysSleep* by proving its optimality.

▶ **Theorem 1.** *For a taskset* Γ *using ES-RMS, SysSleep yields the maximum possible forced-sleep utilization* U_{sleep}^{max}.

Proof. Consider the critical zone theorem [24] where, in the worst case, the requests of all tasks arrive simultaneously. In order to be schedulable, a task τ_i must complete before its deadline D_i, i.e., its worst-cast response time $R_i \leq D_i$. If an ES-task is added to the system, all tasks will now complete at a later time, which should still be less than D_i for the task to remain schedulable. Since the workload changes at every scheduling point, *SysSleep* determines the maximum workload α_i^t, that can be added to the system, such that a task τ_i completes exactly at the end of each *idle* period t between R_i and D_i. This maximum workload corresponds to the slack utilization in the schedule up to time t. While calculating α_i^t, we consider a task's execution as well as all other higher-priority tasks. For a task, the maximum workload that can be added is chosen to be the *maximum* of these candidate values. We refer to this as the *maximum additional workload*, $\rho_i^{max} = max_t(\alpha_i^t)$ for a task τ_i.

For a taskset Γ, the maximum highest-priority workload that can be added also corresponds to the maximum possible forced sleep U_{sleep}^{max}, which is the *minimum* of the *maximum additional workload* of all the tasks, i.e., $U_{sleep}^{max} = min_{\tau_i \epsilon \Gamma}(\rho_i^{max})$. Hence, U_{sleep}^{max} corresponds to the task, τ_c with the lowest *maximum additional workload*, i.e. $min_{\tau_i \epsilon \Gamma}(\rho_i^{max})$. If the added workload exceeds U_{sleep}^{max}, then τ_c will miss its deadline and the taskset will become unschedulable. ◀

▶ **Example 2.** Consider a taskset Γ consisting of two tasks $\tau_1 = (1, 5)$ and $\tau_2 = (1, 7)$. For τ_1, the only end-of-idle period to consider is 5.

$$\alpha_1^5 = (t - C_1)/t = 0.8, \rho_1^{max} = max(\alpha_1^5) = 0.8$$

For τ_2, the end-of-idle periods to consider are 5 and 7.

$$\alpha_2^5 = [t - (C_1 + C_2)]/t = 0.6, \alpha_2^7 = [t - (2C_1 + C_2)]/t = 0.57, \rho_2^{max} = max(\alpha_2^5, \alpha_2^7) = 0.6$$

Hence, the maximum workload U_{sleep}^{max} that can be added is: $U_{sleep}^{max} = min(\rho_1^{max}, \rho_2^{max}) = 0.6$

We now need to find the set of T_{sleep} values which yield the maximum forced-sleep utilization U_{sleep}^{max}. For each task τ_i, let the end-of-idle period to which ρ_i^{max} corresponds be its *critical deadline*, $t_i^{critical}$. Using this notation, we can state the following lemma:

▶ **Lemma 3.** *If* T_{sleep} *is a sub-harmonic of* $t_i^{critical}$, *then the ES-task* τ_{sleep} *can utilize all the slack* ρ_i^{max} *till* $t_i^{critical}$, *such that* τ_i *completes at* $t_i^{critical}$.

Proof. If T_{sleep} is a sub-harmonic of $t_i^{critical}$, the effective utilization [34] of τ_{sleep} in the duration $[0, t_i^{critical}]$ is equal to its utilization U_{sleep}. The effective utilization of a task in a duration $[0, t]$ is the fraction of processor time used by a task in that duration. The actual utilization of a task cannot exceed its effective utilization in *any* duration. Hence, τ_{sleep} can optimally utilize all the slack ρ_i^{max} in the duration $[0, t_i^{critical}]$, such that its effective and actual utilizations are equal in the duration, i.e. $U_{sleep} = \rho_i^{max}$. ◀

The calculated U_{sleep}^{max} corresponds to the task with the minimum ρ_i^{max}. Let us call this the *critical task* τ_c, and let the end-of-idle period to which ρ_c^{max} corresponds be its *critical deadline*, $t_c^{critical}$. Applying Lemma 2 in the context of τ_c, we can state the following corollary:

Algorithm 1 SysSleep Algorithm

procedure SysSleep(Γ)
  for $\tau_i \in \Gamma$ **do**
    $(\rho_i^{max}, t_i^{critical}) = \text{CalculateMaxSlack}(\tau_i, \Gamma)$
  $U_{sleep}^{max} = min(\rho_i^{max}, \tau_i \in \Gamma)$ ▷ Max Sleep Utilization
  $t^{critical} = t_{argmin(\rho_i^{max})}^{critical}$ ▷ Critical Deadline
  return $U_{sleep}^{max}, t^{critical}$
procedure CalculateMaxSlack(τ_i, Γ)
  /* S = slack, I = idle duration, BusyFlag is set if core *busy*, β = workload */
  $S = I = \beta = \Delta = 0, \mu = 1$, BusyFlag=TRUE
  $\omega = C_i, \omega' = 0$
  while $\omega < D_i$ **do**
    if BusyFlag == TRUE **then** ▷ Start of a busy period
      $\Delta = D_i - \omega$
      while $\omega < D_i$ AND $\Delta > 0$ **do**
        $\omega' = \sum_{j=0}^{i}[C_j * (\lfloor \omega/T_j \rfloor + 1)] + S$ ▷ Workload Calculation
        $\Delta = \omega' - \omega, \omega = \omega'$
      BusyFlag = FALSE
    else ▷ Start of an idle period
      $I = min_{\forall j<i}[(T_j * \lceil \omega/T_j \rceil - \omega), D_i - \omega]$ ▷ Slack Computation
      $S = S + I, \omega = \omega + I, t = \omega, \beta = \omega - S$
      if $\beta/t < \mu$ **then**
        $\mu = \beta/t, t^{critical} = t, \rho = 1 - \mu$ ▷ Update the maximum additional workload
      BusyFlag = TRUE
  return $\rho, t_{critical}$

▶ **Corollary 4.** *If T_{sleep} is a sub-harmonic of $t_c^{critical}$, then the ES-task, τ_{sleep}, optimally utilizes all the slack, such that the critical task τ_c completes at $t_c^{critical}$.*

Unfortunately, choosing any sub-harmonic of $t_c^{critical}$ may not guarantee schedulability for other tasks in Γ. If the effective utilization of τ_{sleep} exceeds ρ_k^{max} in the duration $[0, t_k^{critical}]$, for another task $\tau_k \in \Gamma$, then τ_k will become unschedulable. Hence, we need to choose T_{sleep} such that the effective utilization of τ_{sleep} is always less than $\rho_i^{max} \forall \tau_i \in \Gamma$.

▶ **Theorem 5.** *Choosing T_{sleep} as a common divisor of all $t_i^{critical} \forall \tau_i \in \Gamma$ such that $T_{sleep} \le T_1$, always yields a schedule with the optimal forced-sleep utilization U_{sleep}^{max}.*

Proof. From Lemma 2, choosing T_{sleep} as a *common divisor* of all $t_i^{critical}$ ensures that the effective utilization U_{sleep}^{eff} of the energy-saver task τ_{sleep} is equal to its maximum utilization U_{sleep}^{max} in all the critical durations $[0, t_i^{critical}] \forall \tau_i \in \Gamma$. The optimal forced-sleep utilization is given by, $U_{sleep}^{max} = min_{\tau_i \in \Gamma}(\rho_i^{max})$. Hence, $U_{sleep}^{eff} = U_{sleep}^{max} \le \rho_i^{max} \forall \tau_i \in \Gamma$. ◀

It is very important to note that, in practice, the choice of T_{sleep} is constrained by the system constraint $C_{SleepMin}$ on the lower side and the period of the highest-priority task T_1 (τ_{sleep} must execute at the highest priority) on the higher side. Given this system constraint, we can state the following theorem:

▶ **Theorem 6.** *Consider a taskset Γ, schedulable by an ES scheduler, running on a system with the minimum deep sleep round-trip duration $C_{SleepMin}$. Then for Γ, the lower bound on*

Algorithm 2 ThermoSleep Heuristic

procedure ThermoSleep($\Gamma, C_{SleepMin}, num\_core$)
  while True **do**
    $U_{sleep}^{max}, t_j^{critical} = \text{SysSleep}(\Gamma)$ ▷ Invoke SysSleep
    if $t_j^{critical} \le D_j'$ **then break** ▷ If critical deadline is within generalized deadline
  if $C_{SleepMin}/U_{sleep}^{max} < T_1$ **then** ▷ Check if feasible solution exists
    $\mu = \lfloor U_{sleep}^{max} * t^{critical}/C_{SleepMin} \rfloor$, $\nu = \lceil t^{critical}/T_1 \rceil$ ▷ Range of divisors
    if $\mu < \nu$ **then**
      $\mu = \nu$
    $\Theta_{best} = \infty$
    for $k = \mu$ to ν **do**
      $T_{sleep}^k = t^{critical}/k$
      $\Theta_k^{best} = \text{CalcTemperature}(U_{sleep}^{max}, T_{sleep}^k)$ ▷ Lowest temperature for T_{sleep}^k
      if $\Theta_{best} < \Theta_k^{best}$ **then**
        break
      else
        $U_{sleep}^{best} = \text{FindSleepUtil}(\Gamma, T_{sleep}^k, num\_core)$ ▷ Find best U_{sleep} for T_{sleep}^k
        $\Theta_{max} = \text{CalcTemperature}(U_{sleep}^{best}, T_{sleep}^k)$
        if $\Theta_{max} < \Theta_{best}$ **then**
          $T_{sleep} = T_{sleep}^k, U_{sleep} = U_{sleep}^{best}, \Theta_{best} = \Theta_{max}$ ▷ Best Solution found
  else
    return NotSchedulable ▷ No feasible solution exists
  return T_{sleep}, U_{sleep}
procedure FindSleepUtil(Γ, T_{sleep}, m)
  /* m = num_cores, Γ_i = tasks allocated to core i */
  for $i = 1$ to m **do**
    $U_{sleep}^i = \text{FindBestSleep}(\Gamma_i, T_{sleep})$ ▷ Invoke FindBestSleep
  return $min_i(U_{sleep}^i)$

the optimal worst-case maximum temperature Θ_{max}^{best} achievable by ES schedulers is:

$$\Theta_{max}^{best} = (a/b)[(e^{bT_{sleep}^{min}(1-U_{sleep}^{max})} - 1)/(e^{bT_{sleep}^{min}} - 1)] * e^{bU_{sleep}^{max}T_{sleep}^{min}}. \tag{8}$$

Proof. For a taskset Γ, *SysSleep* returns the maximum possible forced-sleep utilization U_{sleep}^{max}. Hence, given the system constraint $C_{SleepMin}$, the smallest feasible ES-task period is $T_{sleep}^{min} = C_{SleepMin}/U_{sleep}^{max}$. From Equation 7, the worst-case maximum temperature Θ_{max} is minimized by simultaneously minimizing T_{sleep} and maximizing U_{sleep}. Hence, substituting the smallest feasible ES-task period, T_{sleep}^{min}, and the largest schedulable forced-sleep utilization U_{sleep}^{max} in Equation 7 yields the lower bound on the *optimal* worst-case maximum temperature Θ_{max}^{best} achievable by ES Schedulers, corresponding to the taskset Γ. ◀

From a thermal perspective, for a fixed U_{sleep}, a smaller T_{sleep} yields a lower worst-case maximum temperature. Hence, a possible thermally-effective solution with optimal forced-sleep utilization can be the smallest *common divisor* of all $t_i^{critical} \forall \tau_i \in \Gamma$ that lies in the range $[C_{SleepMin}/U_{Sleep}^{max}, T_1]$. If $C_{SleepMin}/U_{sleep}^{max} > T_1$, then no feasible solution exists. Note that, choosing T_{sleep} as *any* common divisor of $t_i^{critical} \forall \tau_i \in \Gamma$ that lies in the range

$[C_{SleepMin}/U^{max}_{Sleep}, T_1]$ would yield solutions with equivalent energy consumption. However, the dependence of temperature on T_{sleep} would yield different thermal profiles.

Unfortunately, in many cases, no common divisor of the critical deadlines may lie in $[C_{SleepMin}/U^{max}_{Sleep}, T_1]$. Hence, we present the *ThermoSleep* heuristic. *ThermoSleep* invokes *SysSleep* to compute U^{max}_{sleep}, along with the critical deadline $t^{critical}_c$ corresponding to U^{max}_{sleep}. *ThermoSleep* uses these values to return the smallest possible sub-harmonic of the critical deadline $t^{critical}_c$ corresponding to the critical task τ_c, that yields a thermal and energy-efficient schedule. The pseudo-code for *ThermoSleep* is presented in Algorithm 2.

Given an ES-task period $T_{sleep} \leq T_1$, *ThermoSleep* uses the *FindBestSleep* (FBS) algorithm to compute the optimal C_{sleep} for a core, which allows a taskset Γ to be schedulable. The pseudo-code for FBS is provided in Algorithm 3. We now prove the optimality of FBS.

▶ **Theorem 7.** *For a taskset Γ schedulable by ES-RMS, with an ES-task τ_{sleep} having a period T_{sleep}, FindBestSleep returns the optimal forced-sleep utilization U'_{sleep}.*

Proof. Consider the critical zone theorem [24] where, in the worst case, the requests of all tasks arrive simultaneously. In order to be schedulable, a task τ_i must complete before its deadline D_i. Given that a new job of τ_{sleep} is dispatched every T_{sleep}, for each task $\tau_i \in \Gamma$, FBS determines the maximum workload that can be added to the taskset, such that τ_i completes by t where, t is an integer multiple of T_{sleep}, i.e., $(k * T_{sleep} \leq D_i)$ or D_i. This gives the effective slack, α^t_i, that τ_{sleep} can utilize, if τ_i and all higher-priority tasks complete by t. For a task τ_i, the maximum highest-priority workload with period T_{sleep} that can be added is the *maximum* of these calculated values $\rho^{max}_i = max_t(\alpha^t_i)$. For a taskset Γ with an ES-task period T_{sleep}, this workload corresponds to the maximum possible forced sleep, U'_{sleep}, which is the minimum of the ρ^{max}_i of all the tasks. Hence, $U'_{sleep} = min_{\tau_i \in \Gamma}(\rho^{max}_i)$, which corresponds to the task, $\tau_c \mid c = argmin_{\tau_i \in \Gamma}(\rho^{max}_i)$. If the added workload exceeds U'_{sleep}, then τ_c will miss its deadline and Γ will become unschedulable. ◀

For ES-RHS+, the total deep-sleep utilization $U_{SleepTotal}$ is given by $1 - \sum_{\tau_i \in \Gamma}(C_i/T_i)$. Hence, for a schedulable taskset, ES-RHS+ guarantees a sleep schedule with *optimal* energy savings. However, this deep-sleep utilization is not uniformly distributed over each period. To reduce the worst-case maximum temperature, the ES-task utilization U_{sleep} must be increased, and its period T_{sleep} must be decreased. In Section 1.2 the schedulability test for ES-RHS+ was discussed, and for each task the *generalized deadline* $D'_i = T_i - (T_{sleep} - C_{sleep})$, is a function of both C_{sleep} and T_{sleep}. Hence, *ThermoSleep* invokes *SysSleep* multiple times to compute U^{max}_{sleep} until the *critical deadline* of the *critical task* lies within its *generalized deadline*. To calculate the generalized deadline, we choose T_{sleep} to be the smallest sub-harmonic of the critical deadline in the feasible range $[C_{SleepMin}/U^{max}_{sleep}, T_1]$, and $C_{sleep} = U^{max}_{sleep} * T_{sleep}$.

Given a forced-sleep period, T_{sleep}, ES-RMS can provide a higher forced-sleep utilization, U_{sleep}, than ES-RHS+ [12]. Hence, for a taskset Γ, in most cases, ES-RMS will yield a lower worst-case maximum temperature compared to ES-RHS+. In practice, ES-RHS+ can yield lower temperatures, as it utilizes *all* idle durations to put the processor into deep sleep.

5 Thermal-Aware Multi-Core ES Scheduling

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 an ES-task, $\tau_{sleep,k}$, which has a forced-sleep duration of $C_{sleep,k} \geq C_{SleepMin}$ every $T_{sleep,k}$. As mentioned in Section 1.2, two types of multi-core ES scheduling problems were

Algorithm 3 FindBestSleep Algorithm

procedure FINDBESTSLEEP($\Gamma, C_{SleepMin}, T_{sleep}$)
  for $\tau_i \in \Gamma$ **do**
    $(\rho_i^{max}, t_i^{critical})$ = CalculateSlack($\tau_i, \Gamma, T_{sleep}$)
  $U_{sleep} = min(\rho_i^{max}, \tau_i \in \Gamma)$ ▷ Max Sleep Utilization
  if $U_{sleep} * T_{sleep} \geq C_{SleepMin}$ **then** ▷ Check if feasible solution exists
    return $U_{sleep}^{max} * T_{sleep}$
  else
    return NotSchedulable
procedure CALCULATESLACK(τ_i, Γ, T_s)
  /* S = slack, I = idle duration, BusyFlag is set if core *busy*, β = workload */
  $S = I = \beta = \Delta = 0, \mu = 1$, BusyFlag=TRUE, $\omega = C_i, \omega' = 0$
  while $\omega < D_i$ **do**
    if BusyFlag == TRUE **then** ▷ Start of a busy period
      $\Delta = D_i - \omega$
      while $\omega < D_i$ AND $\Delta > 0$ **do**
        $\omega' = \sum_{j=0}^{i}[C_j * (\lfloor \omega / T_j \rfloor + 1)] + S$ ▷ Workload Calculation
        $\Delta = \omega' - \omega, \omega = \omega'$
      BusyFlag = FALSE
    else ▷ Start of an idle period
      $\Omega = \{j \in Z^+ \mid (j-1) * T_s \leq D_i < j * T_s\}$
      $I = min_{\forall j \in \Omega}[(j * T_s * \lceil \omega / j * T_s \rceil - \omega)]$ ▷ Slack computation
      $S = S + I, \omega = \omega + I, t = \omega, \beta = \omega - S$
      if $\beta / t < \mu$ **then**
        $\mu = \beta / t, \rho = 1 - \mu$ ▷ Update the maximum additional workload
      BusyFlag = TRUE
  return ρ

defined in [12]. In this section, we analyze the thermal implications of *SyncSleep* and *Indsleep* scheduling, and propose techniques to derive thermally-effective partitioned schedules.

In multi-core processors, heat also dissipates between adjcacent cores, and the rate of dissipation depends on the temperature differences between them. Hence, each core can be modeled using the RC model with the addition of thermal resistances between adjacent cores [16]. Let the instantaneous temperature on each core be Θ_j, for $j = 1, 2, ..., m$. Using Fourier's Law, the differential equation for each core's temperature can be given by:

$$\frac{d\Theta_j(t)}{dt} = \frac{P_j(t)}{C} - \frac{\Theta_j(t) - \Theta_A}{RC} - \sum_{k=1}^{m} \frac{\Theta_j(t) - \Theta_k(t)}{R_{jk}C} \tag{9}$$

where, P_j is the instantaneous power dissipated by the core, and R_{jk} is the thermal resistance between the cores j and k. For *non-adjacent* cores one can reasonably assume there is no heat dissipation between them and hence, $R_{jk} = \infty$ [16].

5.1 SyncSleep Scheduling

For *SyncSleep* scheduling, the forced-sleep task must be synchronized across all cores [12]. As the sleep transition is synchronous, for all cores $T_{sleep,k} = T_{sleep}$, and the initial ES-task phase can be taken as $\phi_{sleep,k} = 0$ [12]. Additionally, the minimum amount of time for which the

Algorithm 4 SyncSleep Partitioning Heuristic

1: **procedure** PARTITIONTASKSET($\Gamma, C_{SleepMin}, m$)
2: /* m = number of cores, Γ_i = tasks allocated to core i */
3: $T_s = 1$ ▷ Set forced-sleep period to 1
4: $\Gamma_i \forall i \in 1$ to m = MaxSyncSleep($\Gamma, C_{SleepMin}, T_s, m$) ▷ from [12]
5: U_s, T_s = ThermoSleep($\Gamma, C_{SleepMin}, m$) ▷ Invoke ThermoSleep
6: **return** U_s, T_s ▷ SyncSleep task parameters

system can be in deep sleep is dictated by the core which has the least forced-sleep duration [12]. Hence, if the system synchronous sleep $C_{SyncSleep} = \min_{k=1}^{m}(C_{sleep,k}) \geq C_{SleepMin}$, then the minimum guaranteed deep-sleep utilization is given by $\min_{k=1}^{m}(C_{sleep,k})/T_{sleep}$.

Based on the synchronous-sleep constraint, in the worst case, we can assume that all the cores are in deep sleep for the durations $[kT_{sleep}, kT_{sleep} + C_{SyncSleep})$, and busy from $[kT_{sleep} + C_{SyncSleep}, (k+1)T_{sleep})$. Hence, all cores will have the same worst-case execution profile (as illustrated in Figure 2(a)), and we can assume that in the *worst case*, at any time instant, all cores share the *same* temperature. Thus, the worst-case inter-core temperature difference is always zero, and the model reduces to the uniprocessor thermal model. Hence, from a worst-case perspective, we can consider the entire system as one thermal unit. Applying these assumptions in Equation 9, the worst-case SyncSleep temperature model is given by:

$$d\Theta_j(t)/dt = P_j(t)/C - (\Theta_j(t) - \Theta_A)/RC\,. \tag{10}$$

Figure 2(b) presents an example using SyncSleep ES-RMS for a quad-core system with cores $M_i, i = \{1,2,3,4\}$. The taskset $\Gamma = \{\tau_1(6,10), \tau_2(7,10), \tau_3(5,10), \tau_4(4,10)\}$ is used, such that, during partitioning, each core receives one task (τ_i is assigned to M_i). Due to the synchronous nature of forced sleep, all cores have similar temperature profiles, making the heat dissipation between cores negligible. Hence, like the uniprocessor case, the problem reduces to finding a forced-sleep task τ_{sleep} which minimizes T_{sleep} while maximizing U_{sleep}. However, given that we have multiple cores, partitioning the tasks among them also plays a major role in determining the thermally-effective τ_{sleep}. The temperature minimization problem can be stated as the following task-partitioning problem: "Find a partition that has a synchronized ES-task which minimizes the worst-case maximum temperature, such that the workload allocated to each core can be scheduled feasibly by an ES Scheduler."

The stated partitioning 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 [18][25]. Hence, the thermal-aware *SyncSleep* scheduling problem is also NP-hard. Consider the trivial case where all tasks have the same periods, with different computation times. In this case, choosing the optimal T_{sleep} is trivial (from Theorem 4, it is a sub-harmonic of the task period). Given T_{sleep}, the temperature across all cores will be minimized if all cores have the same load. Hence, the problem reduces to calculating the optimal balanced partition for independent tasks with known computation times, which is known to be equivalent to the Partition problem [21] which is NP-Complete [21].

We now present a two-stage heuristic for the partitioning problem:

Partitioning for Thermal Performance: In the first stage, we choose the best possible hypothetical $T_{sleep} = 1$ to find the best synchronous forced sleep that a partitioning heuristic can achieve. Theorem 4 states that, on a single core, the optimal U_{sleep} is achieved when T_{sleep} is a common divisor of the critical deadline. Since 1 is a divisor of all integers, choosing

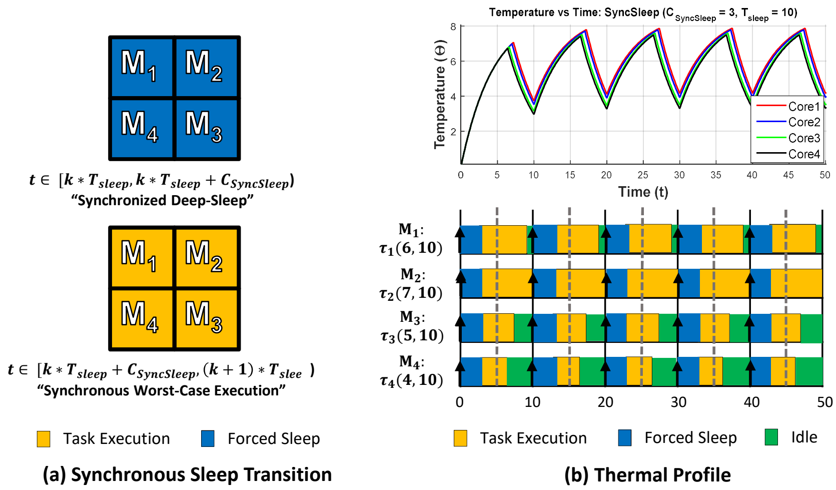

Figure 2 *SyncSleep* Scheduling for a quad-core system with cores M_i.

$T_{sleep} = 1$ enables a heuristic to achieve its best possible forced-sleep utilization. If a taskset cannot be scheduled when $T_{sleep} = 1$, we consider it unschedulable. Setting $T_{sleep} = 1$ and maximizing the forced-sleep utilization is similar to the energy minimization problem for SyncSleep Scheduling [12]. To realize energy savings and minimize temperature in multi-core systems, load balancing is often used [12]. Worst-Fit Decreasing (also referred to as WFD or List Scheduling when the number of cores is fixed *a priori*) is commonly used to obtain a load-balanced partition. WFD allocates tasks to the core with the least utilization, one by one in non-increasing order of their utilization. For ES Schedulers, the period ratios also play an important role in dictating the forced-sleep utilization, something that WFD does not take into account. In [12], the *MaxSyncSleep* (MSS) partitioning heuristic was proposed. Instead of using utilization to allocate tasks to cores, MSS measures the impact of a task's allocation on the synchronous forced-sleep duration.

Choosing the SyncSleep Period: In the second stage, we find a thermally-effective T_{sleep}. For an m-core system, let the best possible synchronous forced-sleep utilization (setting $T_{sleep} = 1$) obtained by a partitioning heuristic A be $U^{max}_{SyncSleep}$, which corresponds to the core k with the minimum forced-sleep utilization. The feasible range for T_{sleep} can now be given by $[C_{SleepMin}/U^{max}_{SyncSleep}, T_1]$. To find a good value for T_{sleep}, we run *ThermoSleep* on the partition. The proposed partitioning technique is described in Algorithm 4.

5.2 IndSleep Scheduling

Some processors allow each core to individually transition into deep sleep, enabling better energy savings. Hence, each core M_k has a forced-sleep task, $\tau_{sleep,k}$, which has a forced-sleep duration of $C_{sleep,k} \geq C_{SleepMin}$ every $T_{sleep,k}$, with a phasing $\phi_{sleep,k}$. Note that, compared to *SyncSleep* scheduling, each core's forced-sleep task can have a different $C_{sleep,k}$, as well as a different phasing $\phi_{sleep,k}$. Hence, we need to consider heat dissipation between cores. Thus, the *IndSleep* thermal model is given by Equation 9, and takes into account both heat dissipation to the environment, as well as between cores.

For *IndSleep* scheduling, the thermal-aware scheduling problem can be defined as follows: "Find a partition and forced-sleep task parameters (including phasing) on each core, that minimizes the maximum temperature of the system, under the constraint that the workload allocated to each core can be scheduled by an ES Scheduler."

In [12], it was proved that using ES-RHS+ can yield an energy-optimal schedule *for all* feasible partitions. A partition is feasible if the tasks allocated to each core are schedulable. However, unlike the energy-minimization problem, all the feasible partitions are *not* optimal from a thermal perspective. This is due to the dependence of temperature on the ES-task period, as well as the execution pattern between cores, i.e. relative ES-task phasing.

The heat flow between two objects is primarily dependent on their thermal properties as well as the temperature difference between them. At any instant, the temperature difference between two adjacent cores will always be less than the temperature difference between a core and the environment. This is based on the practical assumption that the environmental temperature is *always* lower than that of any core. Thus, we can safely assume that heat dissipation to the environment is the dominant factor for cooling. Hence, from an optimization standpoint, we first optimize the schedule on each core to reduce its own temperature, and then optimize the schedule between cores to ensure maximal heat dissipation between them. Based on this practical assumption, we propose a two-stage solution:

Partitioning for Thermal Performance: The objective of partitioning is to ensure that the worst-case maximum temperature of the system is minimized. If there were no heat dissipation between cores, then the worst-case maximum temperature Θ^k_{max} on a core k is a function of $T_{sleep,k}$ and $U_{sleep,k}$. A balanced partition helps ensure that all cores have similar Θ^k_{max}. In an unbalanced partition, a core with a significantly lower $U_{sleep,k}$ would yield a higher temperature, thus raising the maximum temperature of the system. This is similar to the SyncSleep problem, and hence is also NP-Hard. Hence, like SyncSleep, we initially set $T_{sleep,k} = 1$ on each core, and use *MaxSyncSleep* [12] (or WFD) to create a balanced partition. Applying *ThermoSleep* to all the cores together gives a single T_{sleep} that is suitable for all the cores. We refer to this as *uniform sleep*. However, since each core can independently transition into deep sleep, each core's ES-task can have a different period, that we refer to as *non-uniform sleep*. These non-uniform sleep periods $T_{sleep,k}$ can be calculated by applying *ThermoSleep* to each core *individually*. *FindBestSleep* is then used to obtain each $C_{sleep,k}$ using the corresponding $T_{sleep,k}$. While *uniform sleep* ensures that all cores have a similar temperature profile, *non-uniform sleep* can allow each core to attain a lower temperature.

Forced-Sleep Phasing: The phasing between ES-tasks plays an important role in the heat dissipation between cores. In the worst case, we can assume that each core $M_j, j = 1$ to m is in *deep sleep* for the durations $[\phi_{sleep,j} + kT_{sleep,j}, \phi_{sleep,j} + kT_{sleep,j} + C_{sleep,j})$, and *busy* from $[\phi_{sleep,j} + kT_{sleep,j} + C_{sleep,j}, \phi_{sleep,j} + (k+1)T_{sleep,j})$. To ensure maximal heat dissipation between adjacent cores, the temperature difference between them needs to be maximal. For two adjacent cores i and j, the largest temperature difference between them occurs when core i is at the start of its forced-sleep period and core j is at the end of its forced-sleep period. Hence, if $\tau_{sleep,i}$ starts exactly after $\tau_{sleep,j}$ ends, then the instantaneous temperature difference between the cores can be maximized. This leads to an execution pattern where core i is busy while core j is in deep sleep and vice versa.

Figure 3(b) presents an example using IndSleep ES-RMS with uniform periods for a quad-core system with cores $M_i, i = \{1, 2, 3, 4\}$. The taskset $\Gamma = \{\tau_1(6, 10), \tau_2(7, 10), \tau_3(5, 10), \tau_4(4, 10)\}$ is used, such that, during partitioning, each core receives one task (τ_i is assigned to M_i). Note that, each core has its own distinct thermal profile. Additionally, phasing the ES-task on each core, to minimize execution overlap can yield thermal benefits. For the

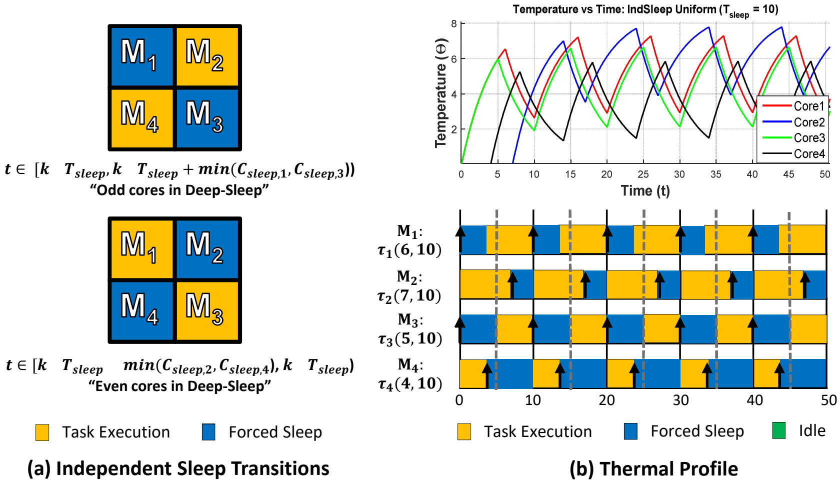

Figure 3 *IndSleep* Scheduling with uniform sleep periods for a quad-core system with cores M_i.

IndSleep example, the *odd-even* execution pattern illustrated in Figure 3(a) is noteworthy, where execution overlap is minimized by ensuring that odd-numbered cores are *busy* (i.e. execute tasks), while even-numbered cores are in deep sleep, and vice-versa. From the thermal profile, observe that this phasing causes the temperature difference between adjacent cores to be maximized, thus yielding better heat dissipation between adjacent cores.

As a simplification, we formulate the phasing problem as one of: "minimizing the execution (or forced-sleep) overlap between adjacent cores". By considering busy durations as *hot* and forced-sleep durations as *cool*, the execution overlap metric captures the durations where *hot* regions overlap, hence acting as a proxy for temperature difference. In most processor designs, cores are rectilinear, and adjacent cores are of the same size. Hence, to compute a thermally-effective phasing, the overlap between every pair of adjacent cores needs to be minimized. This execution overlap (also referred to as *overlap*) needs to be calculated over the *relative hyperperiod*, T_R, of all the cores. We define the *relative hyperperiod* as the least common multiple of all the cores' forced-sleep periods. In the simplest case, consider a dual-core system, with two adjacent cores. Let the cores be M_1 and M_2, and their forced-sleep tasks be $\tau_{sleep,i} = (C_{sleep,i}, T_{sleep,i})$ with phasing $\phi_{sleep,i}$ where, $i = 1, 2$. Assume that all the terms are integers, which is reasonable as we can convert timescales to arbitrarily small units (like nanoseconds). We have four possible cases:

1. $T_{sleep,1} = T_{sleep,2}$, i.e. uniform sleep. The phasing with the minimum overlap is computed over $T_R = T_{sleep,1} = T_{sleep,2}$. The minimum overlap possible is $T_R - C_{sleep,1} - C_{sleep,2}$. Then, $\phi_{sleep,1} = 0$, $\phi_{sleep,2} = C_{sleep,1}$, is one of the phasings which *guarantees* minimum overlap.
2. $T_{sleep,1}$ and $T_{sleep,2}$ are *relatively prime*, i.e. non-uniform sleep whose greatest common divisor is 1. The minimum overlap needs to be computed over $T_R = T_{sleep,1} * T_{sleep,2}$. In this case, any relative *integer phasing* of $\tau_{sleep,1}$ and $\tau_{sleep,2}$ guarantees the same overlap, which is the minimum overlap. This stems from the fact that all possible relative integer phasings between two periods are encountered, before the relative phasing is equal to that at the start.
3. $T_{sleep,1}$ and $T_{sleep,2}$ are *harmonic*, i.e. non-uniform sleep where one is a multiple of the other. Let $T_{sleep,2} = a * T_{sleep,1}, a \in Z^+$. Hence, $T_R = T_{sleep,2}$, and only one iteration of $\tau_{sleep,2}$ occurs in T_R. Then $\phi_{sleep,1} = 0$, $\phi_{sleep,2} = C_{sleep,1}$ can *guarantee* the minimum overlap.

4. $T_{sleep,1}$ and $T_{sleep,2}$ are not *relatively prime* and not *harmonic*, i.e. non-uniform sleep which share a common divisor, but one is non-divisible by the other. Here, $T_R < T_{sleep,1} * T_{sleep,2}$. In this case, no property can be stated on the relative phasing which guarantees minimum overlap.

Based on the above properties, we see that while simple approaches work for phasing uniform sleep, using non-uniform sleep requires more complex optimization techniques.

However, using *uniform sleep* does not always guarantee lower execution overlap than using *non-uniform sleep.* This can be seen from the following 3 cases:

Case 1: *Uniform Sleep* performs better than *Non-Uniform Sleep*. Consider a taskset with two tasks, $\tau_1 = (6, 9)$ and $\tau_2 = (10, 15)$. τ_1 is assigned to core M_1, and τ_2 to core M_2. In the *uniform sleep* case the best ES-task periods in terms of sleep utilization are $\tau_{sleep,1} = (3, 9)$ and $\tau_{sleep,2} = (2.5, 9)$. Using the best possible phasing, achieves a guaranteed minimum execution overlap of 3.5 every 9 (38.89%). In the non-uniform case, the best ES-task periods are $\tau_{sleep,1} = (3, 9)$ and $\tau_{sleep,2} = (5, 15)$. By searching the entire search space of unique relative integer phasings, the minimum execution overlap achievable is 20 every 45 (44.44%). Hence, in this case, using *uniform sleep* provides lower execution overlap.

Case 2: *Uniform Sleep* performs equal to *Non-Uniform Sleep*. Consider a taskset with two tasks, $\tau_1 = (6, 9)$ and $\tau_2 = (9, 12)$. τ_1 is assigned to core M_1, and τ_2 to core M_2. In the *uniform sleep* case, the best ES-task periods in terms of sleep utilization are $\tau_{sleep,1} = (3, 9)$ and $\tau_{sleep,2} = (1.5, 9)$. By using the best phasing, we can achieve a guaranteed minimum execution overlap of 4.5 every 9 (50%). In the non-uniform case, the best ES-task periods are $\tau_{sleep,1} = (3, 9)$ and $\tau_{sleep,2} = (3, 12)$. By searching the entire search space of unique relative integer phasings, the minimum execution overlap achievable is 18 every 36 (50%). Hence, both provide a solution with the same execution overlap.

Case 3: *Uniform Sleep* performs worse than *Non-Uniform Sleep*. Consider a taskset with two tasks, $\tau_1 = (6, 9)$ and $\tau_2 = (9, 11)$. τ_1 is assigned to core M_1, and τ_2 to core M_2. In the *uniform sleep* case, the best ES-task periods are $\tau_{sleep,1} = (3, 9)$ and $\tau_{sleep,2} = (1, 9)$. Using the best phasing, achieves a guaranteed minimum execution overlap of 5 every 9 (55.55%). In the non-uniform case, the best ES-task periods are $\tau_{sleep,1} = (3, 9)$ and $\tau_{sleep,2} = (2, 11)$. By searching the entire search space of unique relative phasings, the minimum execution overlap achievable is 54 every 99 (54.54%). Hence, in this case using *non-uniform sleep* provides lower execution overlap.

Since there is no exact solution for choosing ES-task periods for minimizing execution overlap, we examine the properties of using *uniform sleep* versus *non-uniform sleep*:

Best Phasing: While uniform sleep can be phased easily and optimally (using the odd-even execution pattern from Figure 3(a)) for a rectilinear multi-core processor, no such simple technique can be used for non-uniform sleep.

Temperature Profile: Uniform sleep will ensure that all cores have similar temperatures. However, using non-uniform sleep allows each individual core to choose the best T_{sleep}, to further reduce its temperature, based on the tasks allocated to it.

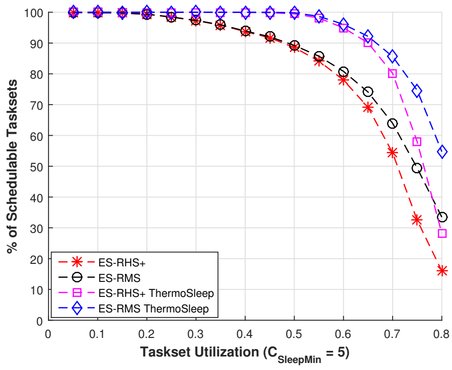

Figure 4 % of task sets schedulable w.r.t. taskset utilization, for uniprocessors

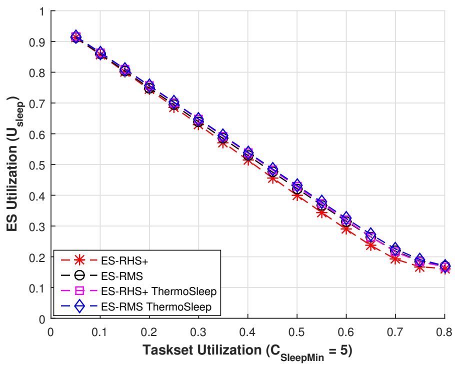

Figure 5 Utilization of the ES-task w.r.t. taskset utilization, for uniprocessors

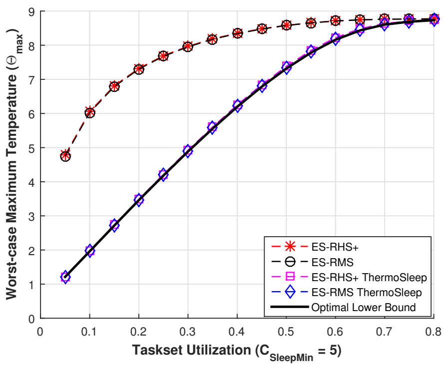

Figure 6 Worst-case maximum temperature w.r.t. taskset utilization, for uniprocessors

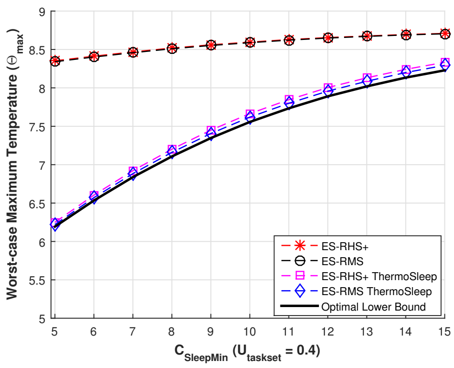

Figure 7 Worst-case maximum temperature w.r.t. $C_{SleepMin}$, for uniprocessors

6 Comparative Evaluation

We now evaluate our proposed techniques on the basis of schedulability and worst-case maximum temperature Θ_{max} with an offset. Results are obtained using both static worst-case analysis as well as dynamic simulations using Hotspot [37]. Static analysis experiments were performed on 100,000 tasksets generated randomly using UUniFast-Discard [14] for each data-point. In a taskset, each task is randomly assigned a period between 15 and 400 time units, and the number of tasks varies from 1 to 20. $C_{SleepMin}$ is set to 5 time units. The system thermal parameters were set to $a = 2$ and $b = 0.228$ [8]. To the best of our knowledge, no other *proactive* techniques exist for designing thermal-aware fixed-priority sleep schedules. Hence, we compare against the purely energy-efficient design methodology proposed in [12].

6.1 Static Worst-Case Analysis

Uniprocessor Comparisons: We compare ES-RMS and ES-RHS+ with and without using *ThermoSleep* on the basis of schedulability, and the worst-case maximum temperature, Θ_{max}. Figure 4 plots schedulability versus taskset utilization. In terms of schedulability: ES-RMS performs better than ES-RHS+. Observe that, using *ThermoSleep*, ES-RMS can schedule up to 62.5% more task sets than before. Figure 5 plots the ES-task utilization versus taskset

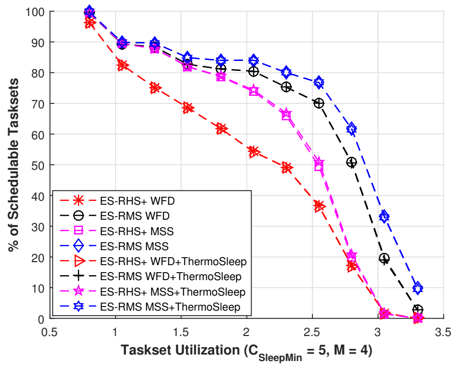

Figure 8 % of task sets schedulable w.r.t. taskset utilization, for multi-core *SyncSleep* scheduling ($m = 4$)

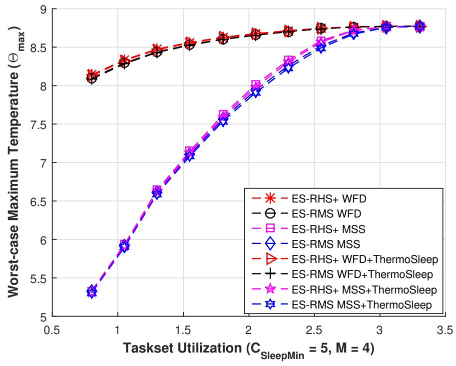

Figure 9 Worst-case maximum temperature w.r.t. taskset utilization, for multi-core *SyncSleep* scheduling ($m = 4$)

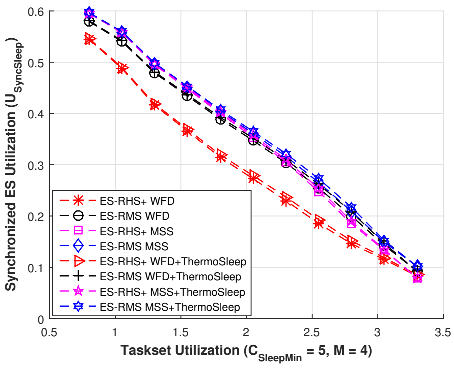

Figure 10 Synchronized ES-task utilization w.r.t. taskset utilization, for multi-core *SyncSleep* scheduling ($m = 4$)

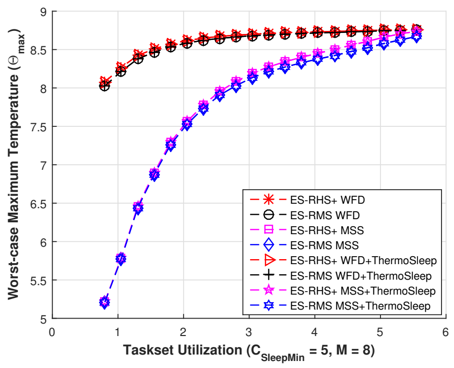

Figure 11 Worst-case maximum temperature w.r.t. taskset utilization, for multi-core *SyncSleep* scheduling ($m = 8$)

utilization for tasksets schedulable by all techniques. By using *SysSleep*, *ThermoSleep*-based techniques yield slightly greater ES-task utilization – up to 3.3% greater for ES-RMS. Figure 6 plots Θ_{max} versus taskset utilization for tasksets schedulable by all techniques. Despite the ES-task utilization being similar, by choosing a smaller ES-task period, *ThermoSleep* can achieve significantly lower temperatures – on average up to 4°K lower for ES-RMS, while simultaneously yielding better energy savings. Figure 6 also plots the average of the lower bound on Θ_{max} for the tasksets. On average, the worst-case deviation between the solution provided by ES-RMS and *ThermoSleep*, and the optimal lower bound was 0.028°K. Figure 7 plots Θ_{max} as a function of $C_{SleepMin}$, when taskset utilization $U_{taskset} = 0.4$. Observe that, despite varying $C_{SleepMin}$, our approach yields solutions with a worst-case temperature difference of 0.067°K compared to the optimal lower bound.

Multi-core SyncSleep Comparisons: We compare ES-RMS and ES-RHS+ on the basis of schedulability and the worst-case maximum temperature, Θ_{max}. We consider each technique using both WFD and *Max-SyncSleep* (MSS) for task partitioning, with and without using *ThermoSleep*. For a quad-core ($m = 4$) processor, Figure 8 plots schedulability versus taskset utilization, and Figure 10 plots the utilization of the synchronized ES-task $U_{SyncSleep}$. In

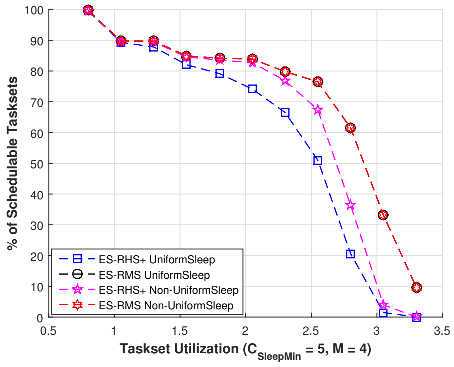

Figure 12 % of task sets schedulable w.r.t. taskset utilization, for multi-core *IndSleep* scheduling ($m = 4$).

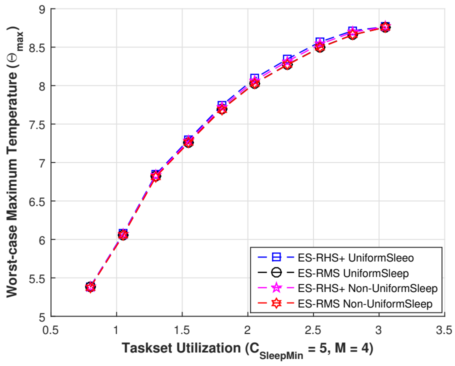

Figure 13 Worst-case maximum temperature w.r.t. taskset utilization, for multi-core *IndSleep* scheduling ($m = 4$).

terms of schedulability and $U_{SyncSleep}$ ES-RMS performs better than ES-RHS+ for all partitioning techniques. For partitioning techniques, MSS marginally dominates WFD in terms of schedulability and $U_{SyncSleep}$, both with and without using *ThermoSleep*. Using *ThermoSleep* provides marginally better $U_{SyncSleep}$ – up to 11.59% greater for ES-RMS with MSS. Figures 9 and 11 plot Θ_{max}, versus taskset utilization, for a quad-core ($m = 4$), and an octa-core ($m = 8$) processor respectively. Using *ThermoSleep* can give significantly lower Θ_{max} – on average up to 2.89°K lower for ES-RMS with MSS for $m = 4$.

Multi-core IndSleep Comparisons: We compare ES-RMS and ES-RHS+ using *Max-SyncSleep* to generate partitions. We consider using both uniform and non-uniform sleep for each core's ES-task. MSS along with *ThermoSleep* is used to determine the sleep periods. Figure 12 plots the percentage of schedulable tasksets. Note that using non-uniform sleep allows for greater schedulability – up to 1.2% greater for ES-RMS. Figure 13 plots Θ_{max} without considering inter-core heat dissipation. Note that, while ES-RHS+ provides maximal energy savings, in all cases ES-RMS yields lower temperatures than ES-RHS+. Additionally, due to better use of each core's *idle* durations, non-uniform sleep provides slightly lower temperatures than uniform sleep – up to 0.08°K.

6.2 Dynamic Simulations

To perform dynamic thermal simulation, we have designed a real-time multi-core scheduling simulation tool called *Inferno* (v1.0). Based on the processor floor-plan, prior temperature, power consumption in the interval and the interval length, *Inferno* uses Hotspot [37] to calculate each core's temperature, in each scheduler-simulation interval. *Inferno* supports fully-partitioned fixed-priority scheduling. Simulation parameters such as the number of cores, simulation cycles, simulation granularity, $C_{SleepMin}$, floorplan and thermal configuration can be specified by the user. The power consumption of each core for different operating frequencies in the *busy*, *idle* and *deep-sleep* states are specified in a look-up table. Based on the taskset and partition provided by the user, *Inferno* provides a trace of the power and temperature values at each simulation instant. The source code for *Inferno* can be found at `https://github.com/sandeepdsouza93/Inferno`.

In order to use realistic power values, we considered the automotive benchmark from the MiBench suite [19]. The benchmark was compiled and executed in the SniperSim [7] cycle-

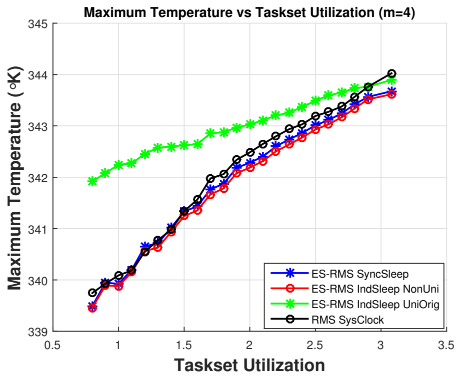

Figure 14 Dynamic simulation: maximum temperature w.r.t. taskset utilization ($m = 4$).

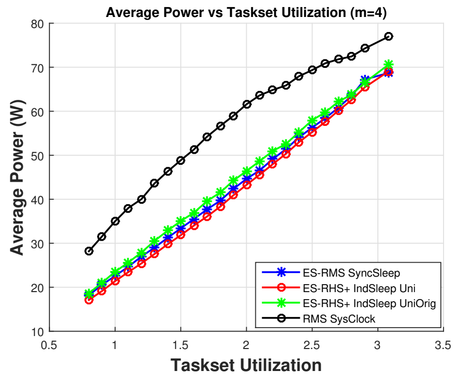

Figure 15 Dynamic simulation: average power w.r.t. taskset utilization ($m = 4$).

accurate x86 emulator (for a Nehalem-class x86 processor) for a range of frequency settings (1.22-2.66 GHz). The execution trace obtained from SniperSim is then fed to the McPAT [26] power simulator, which calculates the power consumption based on an x86 Nehalem power model (45 nm technology node). To model the dependency of static power on temperature, McPAT power calculations were done for the range of temperatures: 300-400°K, and the values were stored in a look-up table. *Inferno* uses these values to compute the core power consumption value, based on the previously calculated core temperature. The scheduling simulation granularity was set to 10μs, and Hotspot's default thermal configuration was used.

We have simulated a quad-core processor, with the floor-plan consisting of cores laid out in a square grid (as shown in Figures 2(a) and 3(a)). 10,000 randomly generated tasksets were considered, each containing 1 to 20 tasks. The taskset utilization varied from 0.8 to 3.2. Each taskset was simulated up to thermal steady state (Hotspot warm-up was considered).

Figure 14 plots maximum temperature versus taskset utilization. Observe that ES-RMS *IndSleep* with non-uniform sleep yields the lowest temperature. We also compared techniques from [12] following a purely energy-efficient design (UniOrig), and it returned the highest temperature – on average up to 3.91°K of difference between ES-RMS IndSleep without thermal considerations (UniOrig), and ES-RMS IndSleep with non-uniform sleep. We also compare our techniques with SysClock, which is the energy-optimal fixed-priority technique for static frequency scaling. We simulate SysClock with RMS where each core could have its own frequency. SysClock yields higher temperatures than IndSleep – up to 1.5°K higher.

Figure 15 plots the average power consumption versus taskset utilization. We find that by better utilizing the idle durations, ES-RHS+ IndSleep yields lower power consumption than ES-RMS SyncSleep – up to 5.04 W lower. ES-RHS+ IndSleep on average yields a power consumption that is 8.52 W lower than SysClock with a maximum difference of 21.74 W. This highlights the importance of energy-saving techniques based on sleep states. Our techniques also provide greater power savings compared to the purely energy-efficient design methodology presented in [12] – up to 8.36 W additional power-savings for ES-RHS+ IndSleep. Note that, although ES-RHS+ IndSleep provides greater energy savings, ES-RMS IndSleep yields lower temperatures. Additionally, even though SysClock consumes significantly more power than ES Schedulers, they both yield similar maximum temperatures. This highlights the fact that energy efficiency does not always imply lower temperatures.

7 Conclusions

In this paper, we analyze the thermal implications of fixed-priority energy-saving schedulers, which utilize the processor's deep-sleep state to save energy. We infer design choices from a well-known thermal model, and present two techniques for designing thermally-effective ES Schedulers: the *SysSleep* algorithm to provide optimal sleep utilization and the *ThermoSleep* heuristic to design a thermally-effective ES-task. Specifically, we derive a lower bound on the optimal maximum temperature, thus quantifying the best thermal performance achievable by ES Schedulers. In the multi-core context, we extend our analysis to two classes of scheduling problems [12]: *SyncSleep*, where cores need to synchronously transition into deep sleep, and *IndSleep*, where cores can independently transition into deep sleep. We consider the impact of both task partitioning and ES-task phasing on temperature. In the SyncSleep context, we observe that the synchronous deep-sleep constraint reduces the temperature-minimization problem to the energy-minimization problem, with the exception of the synchronous ES-task period calculation. On the other hand, while energy minimization is straightforward in the IndSleep context (all feasible partitions are optimal using ES-RHS+[12]), the same cannot be said for temperature minimization. The dependence of temperature on the ES-task periods and relative phasing makes the IndSleep problem non-trivial.

Since we focus on fully-partitioned scheduling, our proposed framework can be extended to heterogeneous multi-core processors. Additionally, our techniques do not require significant knowledge of a system's thermal parameters, and hence are applicable to a range of multi-core platforms. Static analysis and dynamic simulation validate our approach, yielding lower temperatures and better energy savings than both the purely energy-efficient ES Scheduler design [12], and frequency scaling based techniques [29]. Our results show that, while energy savings is key to lower temperatures, not all energy-efficient solutions yield low temperatures.

References

1 Intel Core Processor Family (4th Generation). [online]: `http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/4th-gen-core-family-desktop-vol-1-datasheet.pdf`.

2 Samsung Exynos 5800. [online]: `http://www.samsung.com/semiconductor/minisite/Exynos/w/solution/mobile_ap/5420/`.

3 K. Agarwal, K. Nowka, H. Deogun, and D. Sylvester. Power gating with multiple sleep modes. In *International Symposium on Quality Electronic Design*, pages 633–637, 2006. `doi:10.1109/ISQED.2006.102`.

4 M. Ahmed, N. Fisher, S. Wang, and P. Hettiarachchi. Minimizing peak temperature in embedded real-time systems via thermal-aware periodic resources. *Sustainable Computing: Informatics and Systems*, 1(3):226–240, 2011. `doi:doi.org/10.1016/j.suscom.2011.05.006`.

5 R. Ahmed, P. Ramanathan, and K. K. Saluja. On thermal utilization of periodic task sets in uni-processor systems. In *International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 267–276, 2013. `doi:10.1109/RTCSA.2013.6732227`.

6 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*, 8:284–292, 1993. `doi:10.1.1.132.2794`.

7 T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout. An evaluation of high-level mechanistic core models. *ACM Trans. Archit. Code Optim.*, 11(3):28:1–28:25, 2014. `doi:10.1145/2629677`.

8 Y. Chandarli, N. Fisher, and D. Masson. Response time analysis for thermal-aware real-time systems under fixed-priority scheduling. In *IEEE International Symposium on Real-Time Distributed Computing*, pages 84–93, 2015. doi:10.1109/ISORC.2015.34.

9 T. Chantem, R. P. Dick, and X. S. Hu. Temperature-Aware Scheduling and Assignment for Hard Real-Time Applications on MPSoCs. In *Design, Automation and Test in Europe*, pages 288–293, 2008. doi:10.1109/DATE.2008.4484694.

10 J. J. Chen, S. Wang, and L. Thiele. Proactive speed scheduling for real-time tasks under thermal constraints. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 141–150, 2009. doi:10.1109/RTAS.2009.30.

11 A. K. Coskun, T. S. Rosing, and K. Whisnant. Temperature aware task scheduling in mpsocs. In *Design, Automation Test in Europe Conference Exhibition*, pages 1–6, 2007. doi:10.1109/DATE.2007.364540.

12 S. D'souza, A. Bhat, and R. Rajkumar. Sleep scheduling for energy-savings in multi-core processors. In *Euromicro Conference on Real-Time Systems*, pages 226–236, 2016. doi:10.1109/ECRTS.2016.16.

13 Intel Core2 Duo. [online]: http://download.intel.com/design/processor/datashts/320390.pdf.

14 P. Emberson, R. Stafford, and R. Davis. Techniques for the synthesis of multiprocessor tasksets. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems*, pages 6–11, 2010.

15 AMD Opteron Family. [online]: http://support.amd.com/TechDocs/40036.pdf.

16 N. Fisher, J. J. Chen, S. Wang, and L. Thiele. Thermal-aware global real-time scheduling on multicore systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 131–140, 2009. doi:10.1109/RTAS.2009.34.

17 Y. Fu, N. Kottenstette, C. Lu, and X. D. Koutsoukos. Feedback thermal control of real-time systems on multicore processors. In *ACM International Conference on Embedded Software*, pages 113–122, 2012. doi:10.1145/2380356.2380379.

18 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co, 1990. doi:10.1137/1024022.

19 M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *IEEE International Workshop on Workload Characterization*, pages 3–14, 2001. doi:10.1109/WWC.2001.15.

20 P. M. Hettiarachchi, N. Fisher, M. Ahmed, L. Y. Wang, S. Wang, and W. Shi. The design and analysis of thermal-resilient hard-real-time systems. In *IEEE Real Time and Embedded Technology and Applications Symposium*, pages 67–76, 2012. doi:10.1109/RTAS.2012.17.

21 R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations: Symposium on the Complexity of Computer Computations*, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.

22 N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, 2003. doi:10.1109/MC.2003.1250885.

23 T. Kuroda. Cmos design challenges to power wall. In *International Microprocesses and Nanotechnology Conference*, pages 6–7, 2001. doi:10.1109/IMNC.2001.984030.

24 J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium*, pages 166–171, 1989. doi:10.1109/REAL.1989.63567.

25 J. Y. T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2(4):237–250, 1982. doi:10.1016/0166-5316(82)90024-4.

26 S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *IEEE/ACM International Symposium on Microarchitecture*, pages 469–480, 2009. `doi:10.1145/1669112.1669172`.

27 C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973. `doi:10.1145/321738.321743`.

28 A. Rowe, K. Lakshmanan, H. Zhu, and R. Rajkumar. Rate-harmonized scheduling for saving energy. In *IEEE Real-Time Systems Symposium*, pages 113–122, 2008. `doi:10.1109/RTSS.2008.50`.

29 S. Saewong and R. Rajkumar. Practical Voltage-Scaling for Fixed-Priority RT-Systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 106–115, 2003. `doi:10.1.1.123.1440`.

30 L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990. `doi:10.1109/12.57058`.

31 J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The impact of technology scaling on lifetime reliability. In *International Conference on Dependable Systems and Networks*, pages 177–186, 2004. `doi:10.1109/DSN.2004.1311888`.

32 R. Viswanath, V. Wakharkar, A. Watwe, and V. Lebonheur. Thermal performance challenges from silicon to systems. *Intel Corp. Manufacturing Group*, 2000. `doi:10.1.1.14.8322`.

33 S. Wang, J. J. Chen, Z. Shi, and L. Thiele. Energy-efficient speed scheduling for real-time tasks under thermal constraints. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 201–209, 2009. `doi:10.1109/RTCSA.2009.29`.

34 H. Wei, K. Lin, W. Lu, and W. Shih. Generalized rate monotonic schedulability bounds using relative period ratios. *Information Processing Letters*, 107(5):142–148, 2008. `doi:10.1016/j.ipl.2008.02.006`.

35 F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *IEEE Annual Foundations of Computer Science*, pages 374–382, 1995. `doi:10.1109/SFCS.1995.492493`.

36 B. Yun, K. G. Shin, and S. Wang. Predicting thermal behavior for temperature management in time-critical multicore systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 185–194, 2013. `doi:10.1109/RTAS.2013.6531091`.

37 R. Zhang, M. R. Stan, and K. Skadron. Hotspot 6.0: Validation, acceleration and extension. *University of Virginia*, CS-2015-04.