# Peak Power Management for Scheduling Real-time Tasks on Heterogeneous Many-Core Systems

Waqaas Munawar*, Heba Khdr*, Santiago Pagani*, Muhammad Shafique*, Jian-Jia Chen† and Jörg Henkel*
*Karlsruhe Institute of Technology (KIT), Germany, E-mail: firstname.secondname@kit.edu
†TU Dortmund University, Germany, E-mail: jian-jia.chen@cs.uni-dortmund.de

***Abstract*—The number and diversity of cores in on-chip systems is increasing rapidly. However, due to the Thermal Design Power (TDP) constraint, it is not possible to continuously operate all cores at the same time. Exceeding the TDP constraint may activate the Dynamic Thermal Management (DTM) to ensure thermal stability. Such hardware based closed-loop safeguards pose a big challenge in using many-core chips for real-time tasks. Managing the worst-case peak power usage of a chip can help toward resolving this issue. We present a scheme to minimize the peak power usage for frame-based and periodic real-time tasks on many-core processors by scheduling the sleep cycles for each active core and introduce the concept of a sufficient test for peak power consumption for task feasibility. We consider both inter-task and inter-core diversity in terms of power usage and present computationally efficient algorithms for peak power minimization for these cases, i.e., a special case of *"homogeneous tasks on homogeneous cores"* to the general case of *"heterogeneous tasks on heterogeneous cores"*. We evaluate our solution through extensive simulations using the 48-core SCC platform and gem5 architecture simulator. Our simulation results show the efficacy of our scheme.**

**Keywords:** Peak power managment, Real-time, many-core.

## I. Introduction

The trends of increased integration are resulting in many-core architectures. The increase of digital logic on the chips and the failure of Dennard's scaling [8] will result in increased power densities on next generation chips. Consequently, increased power densities have introduced the so-called Dark Silicon problem, where a significant percentage of the total available cores in a many-core system cannot be powered-on simultaneously due to the thermal constraints [9], [14], [28]. Commonly, the chip manufacturers provide a Thermal Design Power (TDP) value which is considered to be the highest *sustainable* power that a chip can consume without triggering any performance throttling mechanisms [16], e.g., Dynamic Thermal Management (DTM). The heat-sink and the chip's cooling solution are designed according to the TDP value.

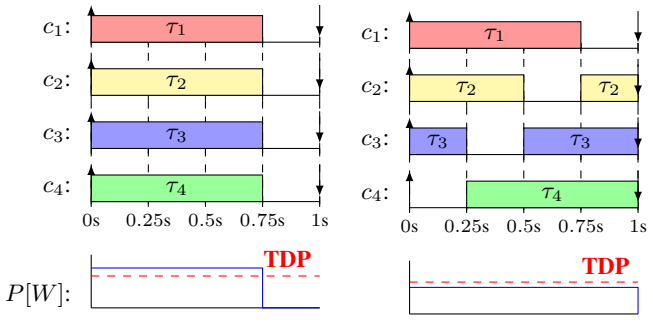
Activation of DTM can result in hardware-based *performance throttling* to keep the chip within safe operating conditions. Modeling the resultant performance loss precisely to guarantee real-time performance constraints is non-trivial due to the introduction of new variables, e.g., ambient temperature and the DTM policy of the chip. One way of guaranteeing sustainable performance is to have a DTM-free operation. If operation below TDP guarantees that DTM is not activated, then consequently, *managing the cumulative peak power consumption of the cores to be within the TDP limit guarantees DTM-free operation*.

Treating TDP as a hard limit negates the possibility of using the cores at high power for short spans of time, i.e., the so called *thermal sprinting*. But, on the other hand, the major benefit it provides is that it becomes a basis of a simple, online and pessimistic but *sufficient* test for guaranteeing the feasibility of real-time tasks scheduled on many-core chips whose TDP values are known. This *peak power based sufficient schedulability test* is similar to the widely used utilization based schedulability test for the design of schedulers in real-time systems. There have been extensive results for the latter in the literature. Moreover, for scheduling decisions, this test abstracts the need to consider the details such as initial temperature of the core, distance of the core from the periphery, distance from other active cores, etc.

In order to operate the chip within the TDP constraint, peak power minimization is helpful. It is important to note that peak power minimization is not the same as energy minimization. Much of the existing research has focused on *energy* minimization for real-time tasks on multi-core platforms [5], [6]. Energy minimization can be equivalent to *average* power minimization, whereas, in this paper our focus is on *peak* power minimization. Also, for cores whose voltage and frequency is individually scalable (per-core DVFS), the problem of minimizing the peak power can be equivalent to the problem of minimizing the energy. That is, as the cores can be individually slowed down appropriately to finish the workload just in time, the core's workload gets distributed over the whole frame. This optimally suppresses any peaks in the power consumption, and also results in optimal reduction of the energy consumption as the cores are operated at the minimum feasible frequency. In this direction, there exist results that can be directly applied [5], [6].

Nevertheless, due to monetary and chip-area cost, per-core DVFS is not feasible for many-core systems (i.e., systems with 100s or possibly 1000s of cores). Hence tiled architectures are getting popular [15]. Each tile consists of a group of cores and the operating frequency is selectable at the granularity of a tile [15]. For such architectures, after the frequency of operation for a tile has been selected, the cores on that tile can be individually be turned *on* or *off*, i.e., Dynamic Power Management (DPM) can be used to control the power consumption. For this, an appropriate scheduling of sleep time for each processing core belonging to a tile can result in reduced peaks in the power consumption for that tile, thereby reducing the peak power consumption for the whole chip. We address this problem in this paper.

**Motivational Example:** Consider a many-core system with




(a) Peak Power of 8 Watts (b) Peak Power of 6 Watts

Fig. 1: Motivational Example of two different schedules.

4 cores, i.e., $c_1$, $c_2$, $c_3$, and $c_4$, where each core executes tasks at 1 GHz. Assume that the power consumption for execution is 2 Watts per core, and that each core consumes 0 Watts when sleeping. Moreover, for simplicity in presentation, assume negligible overhead for sleeping. Under such hardware settings, consider that there are 4 real-time tasks arriving at time 0, i.e., $\tau_1$, $\tau_2$, $\tau_3$, and $\tau_4$. Each task needs to execute $7.5 \cdot 10^8$ computer cycles, and all tasks share a common deadline of 1 second, i.e., frame-based tasks. Although this results in a total utilization of 3, if task migration is not desired, then any task partitioning scheme will assign one task in each core.

Figure 1 shows two possible schedules where all tasks meet their deadlines. For the schedule in Figure 1a, all tasks start execution simultaneously at time 0, and all cores go to sleep after 0.75 seconds. This results in a peak power consumption of 8 Watts. On the other hand, by using Dynamic Power Management (DPM) to control the sleep cycles of the cores, Figure 1b shows another possible schedule. For this second case, the peak power consumption is 6 Watts, from only activating 3 cores at any given time. Assume that TDP for this chip is 7 Watts. In this case, the first schedule will result in activation of DTM whereas the second one can avoid it. In case of the first schedule, DTM might be activated, triggering hardware-based performance throttling to keep the chip within thermally safe operating conditions. This can result in tasks missing their deadlines. Contrarily, in the second schedule DTM activation can be avoided if execution below TDP guarantees DTM-free operation. Here, once a schedule is selected for the real-time task set, a sufficient feasibility test can be designed to check if the cumulative peak power of all cores exceeds the TDP for the chip. This example illustrates (i) the potential benefit of an appropriate coordinated schedule of sleep cycles of the cores in order to reduce the total peak power consumption, and, (ii) the need to add the additional criterion of peak power consumption for checking the feasibility of a task set with real-time requirements.

**Objective:** The goal of this paper is to find a sleeping schedule for active cores on a fixed hardware platform, considering the heterogeneity of cores and tasks, such that the peak power consumption is minimized to help toward satisfying the TDP constraint, while guaranteeing that all real-time tasks meet their deadlines.

**Our Contributions:** For hard real-time tasks:

- We present a peak power management scheme (PPM). For the sake of completeness, we first deal with the task partitioning onto the available cores and decide the individual schedule for all tasks. After this, we schedule the sleep cycle which is equivalent to the unconsumed utilization of each individual core. This is done such that the peak power consumption is minimized, without violating the hard real-time requirements of the tasks. An analysis of our scheme details how our solution decides the sleep cycle for individual cores, starting from *"homogeneous tasks on homogeneous cores"* to the most general case of *"heterogeneous tasks on heterogeneous cores"*.
- In addition to the existing utilization based schedulability tests, we introduce the concept of a sufficient test for schedulability considering the peak power consumption of a task set with real-time requirements.

We simulate our PPM scheme using power traces collected from the 48-core SCC [15] platform and gem5 architecture simulator in combination with McPAT [20].

The paper is organized as follows: In Section IV, we present an overview of our scheme. In Sections V, VI and VII we deal with the different cases arising due to heterogeneity of tasks and processing cores while focusing only on frame-based real-time tasks. In Section VIII we generalize our solution to include implicit deadline periodic tasks. In Section IX we present the results of simulations and we conclude the paper in Section X.

## II. Related Work

In the past, much research has focused on power, energy and thermal management for multi-core systems [10], [17], [24], [25]. In [10] the global thermal-aware scheduling of sporadic tasks is analyzed to minimize the peak temperature using DVFS as a knob. For a multi-core system where cores share voltage and frequency islands, the work in [24], [25] presents and analyses an energy-efficient task partitioning scheme. Likewise, in [17] optimal procrastination interval for each task with real-time constraints is derived to minimize the energy consumption. In these works the scheduling decisions are intended to reduce the energy or temperature by controlling the average power consumption. Therefore, these cannot be effectively modified to control the *peak* power consumption to remain within the TDP constraint, as is the case in our work. Moreover we cater for heterogeneity of cores and tasks as well.

The work in [23], [26], [27] focuses on *maximizing performance under a power constraint, e.g., TDP*. In [23], a control-based framework is proposed to obtain the optimal trade-off between power and performance of asymmetric multi-core systems under a specific power budget (TDP). The work in [26] exploits the process variations between the cores in a homogeneous multi-core system to pick the more suitable cores for an application to improve performance. Their results show that the performance efficiency can be increased along with the increasing dark silicon area, due to the proportional increment of the process variations. However, in both of these works [23], [26], the performance is not guaranteed, making it unsuitable for real-time tasks. In our work, we tackle the dual problem, in which we focus on *minimizing the peak power consumption* while considering the schedubility

of the *hard real-time tasks as constraints*, i.e., delivering a guaranteed performance. In [27] Sartori et al strive to boost the performance while guaranteeing that power consumption of the chip does not shoot beyond a threshold. However, the performance is again not guaranteed. In contrast, we provide a simple, polynomial time, online admissibility test for hard real-time task sets.

Another work in high correlation to ours is [19], which develops a new scheduling algorithm that minimizes the peak power consumption for real-time tasks. However, the complexity of the method is so high that it can only be used for offline design. In comparison, we present polynomial time algorithms, that can be used for online scheduling.

## III. System Model and Problem Definition

We employ the commonly used system model for a heterogeneous multi-core system [2] as follows: Given a set of software processes (tasks), a set of processing entities (cores) upon which these tasks can execute, and the rate at which the processing cores execute the tasks and their corresponding power consumption values, our goal is to determine a mapping of the tasks onto the cores and determine a time schedule for the cores in such a way that the peak power consumption is minimized.

### A. Task and Hardware Model

This paper focuses on a set of $n$ periodic tasks $\mathbf{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ which share the same arrival time and deadline i.e., *frame-based tasks*. All tasks have the same period $T$, and in each period all tasks have the same arrival time 0. This task model is later extended to include the more general *implicit deadline periodic tasks* in Section VIII where tasks do not necessarily share the same period. We consider *partitioned scheduling*, in which each task is assigned onto a core, i.e., task migration among cores is not allowed. We also assume that all jobs are independent. That is, they do not share resources, they do not have data dependencies, and there is no interprocess communication. This model is not as restrictive as it appears, since there are ways to transform a set of dependent tasks to independent ones [18].

We focus on a multi-core system with $m$ heterogeneous cores, that is, $\mathbf{C} = \{c_1, c_2, \ldots, c_m\}$. Having a system with heterogeneous cores implies that tasks will have different execution times and power consumptions, depending on the core in which each task is mapped to. The power consumption for execution consists of a dynamic and static component. Moreover, cores can be put to sleep mode by gating the clock. We assume that changing the execution frequency of cores or putting them to sleep mode, and vice versa, takes negligible amount of time, as it is accomplished through clock gating. During the sleep mode cores consume only the static power, which might be different for each core. However, since the static power is continuously being consumed, it only adds a constant offset to the peak power consumption of tasks (considering the tiled-architecture, all cores belonging to tiles in which at least one core is active consume static power, whereas the cores in other tiles can be turned off/power gated). Hence, without loss of generality, we focus on dynamic power consumption and consider leakage as a constant offset. Formally, we assume that the peak dynamic power consumption of task $\tau_i$ executing on core $c_j$ is denoted as $p_{i,j}$, resulting in a peak power matrix $\mathbf{P} = [p_{i,j}]_{n\times m}$. Normally, during different execution phases a task might have different power values, and considering only the peak power is a safe approach.

Similarly, we define $u_{i,j}$ as the utilization of task $\tau_i$ executing in core $c_j$. That is, assume that task $\tau_i$ requires $x_{i,j}$ amount of time in the worst case when executing on core $c_j$ to meet its deadline. The period of $\tau_i$ is $T$ for all $i$ (frame-based tasks), thus, $u_{i,j} = \frac{x_{i,j}}{T}$. This results in an utilization matrix $\mathbf{U} = [u_{i,j}]_{n\times m}$. It holds that $0 \leq u_{i,j} \leq 1$, when $x_{i,j} \leq T$. If $x_{i,j} > T$, then we set $u_{i,j}$ to $\infty$ to guarantee that core $c_j$ is not considered for placement of task $\tau_i$.

### B. The Studied Problem

For $n$ frame-based tasks and a heterogeneous many-core platform with $m$ cores, the objective of this paper is to find a schedule and mapping for executing all the $n$ tasks without violating their deadlines, while minimizing the total peak power consumption.

## IV. Solution Overview

To fulfil the aforementioned objective, we propose a peak power management scheme (PPM). The solution presented in this paper consists of two independent steps as discussed in the subsequent sections. The first step deals with task partitioning into the cores and decides the individual schedule of each core. In the second step we decide a sleep cycle for individual cores in such a way that the peak power is minimized without affecting the individual schedules of cores. The second step is the key challenge targeted in this paper.

### A. Step 1: Task Partitioning

In the first step, as we consider partitioned scheduling, the mapping of the tasks into cores has to be decided. Performing partitioned scheduling to ensure the timing constraints is a well-studied topic, in which the recent survey by Davis and Burns [7] provides a comprehensive study. However, as this part is more related to the feasibility of task partitioning, we only sketch the key concepts.

Deciding whether there exists a feasible task assignment for a set of tasks with real-time constraints into multiple cores is an $NP$-hard problem in the strong sense [2]. However, for example, the approximation algorithms by Graham [12] and Baruah [2] can be adopted to provide efficient and effective task partitioning for homogeneous and heterogeneous many-core systems, respectively.

The output of the task partitioning algorithms is a partition matrix $\mathbf{K} = [k_{i,j}]_{n\times m}$. For every element in the matrix, $k_{i,j}$ is set to 1 if task $\tau_i$ is partitioned to be executed on core $c_j$ and 0 otherwise.

It is not necessary to use all the given $m$ cores for assigning the tasks, that is, the task partitioning may group the tasks into less than $m$ cores. If the time complexity is tolerable, the whole process (*Step 1* and *Step 2*) can be iteratively called to decide the number of cores for assigning the tasks. For the simplicity of presentation, we only focus on one iteration in which the number of cores for allocating tasks is fixed, and any known algorithm, e.g., [2], [12], for task partitioning is adopted.

Moreover, the derived task partitioning also guarantees that the utilization of the tasks assigned on one core is less than or equal to $100\%$, in which adopting any workload-conserving scheduling policy in one core individually ensures that the tasks can meet the deadlines due to the assumption of frame-based tasks. If the derived task partitioning has a core with utilization larger than $100\%$, either another more powerful task partitioning algorithm should be adopted or we should consider more cores to partition the tasks.

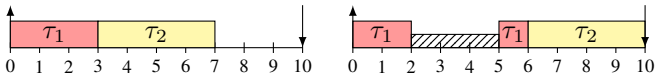
### B. Step 2: Sleep Schedule Decision

After the tasks are mapped onto the cores and every core's internal schedule is decided in *Step 1*, this phase decides the sleep schedule for each core such that the internal schedule of the core remains unaffected and the peak power is minimized.

*Theorem 1:* If a frame-based, synchronous task set with period $T$ and cumulative utilization $U$, can be feasibly scheduled on a single core, it can also be feasibly scheduled if the core is halted for $\lfloor T(1-U) \rfloor$ time in every $T$ interval.

*Proof:* The system is either idle or executing jobs in time interval $[0, T)$. Since we focus on frame-based real-time tasks here, any schedule is feasible if the core is halted only by at most $T(1-U)$ amount of time, as the remaining time is used for executing jobs. ■

**Explanatory Example**: Suppose that after partitioning, one of the processing cores has two tasks $(\tau_1, \tau_2)$ with the period of 10 ms and worst case execution times of 3 ms and 4 ms, respectively. The cumulative utilization is $\frac{3}{10} + \frac{4}{10} = 0.7$. As per Theorem 1, the processing core can be put to sleep for 30% of time and the task set can still be feasibly scheduled. A sample schedule is shown in Figure 2.




(a) Original schedule (b) Schedule with sleep cycle

Fig. 2: An example of a modified schedule with sleep cycle.

Based on the chip's architecture and task structure, we consider two observations.

1) *Task set heterogeneity*: The power consumption of a core caused by executing a task can either be task independent, or dependent. We call the former case *homogeneous* task set and the latter *heterogeneous* task set. For homogeneous task set, it holds that $p_{i,j} = p_{i+1,j}$ for all $i = 1, 2, \ldots, n-1$ (all the rows of column $j$ in matrix $\mathbf{P}$ are equal).
2) *Core heterogeneity*: Like tasks, we classify the cores into two disjoint sets: *homogeneous* and *heterogeneous*. Homogeneous cores are those in which the power consumption of task $\tau_i$ is independent of the core where it is executed, i.e., $p_{i,j} = p_{i,j+1}$ for all $j = 1, 2, \ldots, m-1$ (all the columns of row $i$ in matrix $\mathbf{P}$ are equal). Furthermore, this implies that all cores are equal in their capabilities, hence task $\tau_i$ has the same utilization on any core, i.e., $u_{i,j} = u_{i,j+1}$ for all $j = 1, 2, \ldots, m-1$ (all the columns of row $i$ in matrix $\mathbf{U}$ are equal). Those cores that do not follow the above condition are heterogeneous cores.

These two conditions results in four possible cases. We discuss these in the following sections and provide efficient solutions for each.

## V. Homogeneous Tasks on Homogeneous Cores

In this section we consider that all the tasks are identical in their power consumption and only differ from each other in the utilization requirements. Also, all the cores have identical behavior for power consumption and computational performance output. This is the simplest case and will form the foundation for solving the complex cases that follow.

As we consider only homogeneous tasks being executed on homogeneous cores, it holds that $p_{i,j} = p_{\text{const}}$ for all $p_{i,j} \in \mathbf{P}$. For this case, a time schedule with minimum peak power can be obtained by extending the McNaughton's wrap-around rule [22] using core utilization values. Note that the wrap-around rule was designed for other scheduling purposes (i.e., to minimize the maximum completion time).

Suppose that the core utilizations, for all $m$ cores in the system, are represented through the row matrix $\mathbf{W} = [w_j]_m$. Using the partition matrix $\mathbf{K}$ and the utilization matrix $\mathbf{U}$, we can fill matrix $\mathbf{W}$, such that $w_j = \sum_{i=1}^{n} k_{i,j} \cdot u_{i,j}$ for all $j = 1, 2, \ldots, m$. Clearly, for a feasible schedule, it should hold that $w_j \leq 1$ for all $j = 1, 2, \ldots, m$.

Using $\mathbf{W}$, we apply the wrap-around rule as follows. Assume that there are $m$ bins of time $(b_1, b_2, \ldots, b_m)$, each of size $T$. The starting time for each is 0. Iteratively, we assign $T \cdot w_j$ time from bin $b_k$ to core $j$, starting from $b_1$ and $w_1$. When core $j$ is assigned time from bin $b_k$, the value of $b_k$ is updated to $b_k - T \cdot w_j$. If for core $j$, the time requirement cannot be fully satisfied from bin $b_k$, then we assign as much as possible from $b_k$ so that $b_k$ becomes zero, and the rest is assigned from $b_{k+1}$. As $w_j \leq 1 \; \forall w_j \in \mathbf{W}$, a core gets time slices from at most two bins. The time slices assigned by this algorithm forms the schedule of the core.

An example of assigning 3 cores having utilizations $0.5, 0.9$ and $0.5$ into 3 bins is presented in Figure 3. Here, $c_1$, having a utilization of $0.5$, is completely assigned to first half of $b_1$. In terms of per-core DPM schedule, this allocation means that $c_1$ is only turned on in the beginning half of the frame, let us say $T$, then, from 0 to $0.5T$. $c_2$, with a utilization of $0.9$, cannot be fully assigned to $b_1$, so it is partially assigned to the last half of $b_1$ and the rest is "wrapped-around" to the initial $40\%$ of $b_2$. Correspondingly, for its DPM schedule, $c_2$ can be turned on in the initial $40\%$ of the frame $(0-0.4T)$ and then in the last half of the frame $(0.5T - T)$. Similarly, $c_3$ is assigned time slot from $0.4T$ to $0.9T$ fulfilling its requirement of $0.5$ utilization.

This is a polynomial-time algorithm, with a time complexity of $\mathcal{O}(m)$ for $m$ cores. Moreover, due to the wrap-around policy, it is also clear that at any time instant, the algorithm will activate at most $c^*$ cores at the same time, where $c^* = \left\lceil \sum_{i=1}^{n} \sum_{j=1}^{m} k_{i,j} \cdot u_{i,j} \right\rceil$.

Clearly the peak power consumption, $\pi$, of the task scheduled using above presented rule is given as $\pi = p_{\text{const}} \cdot c^*$

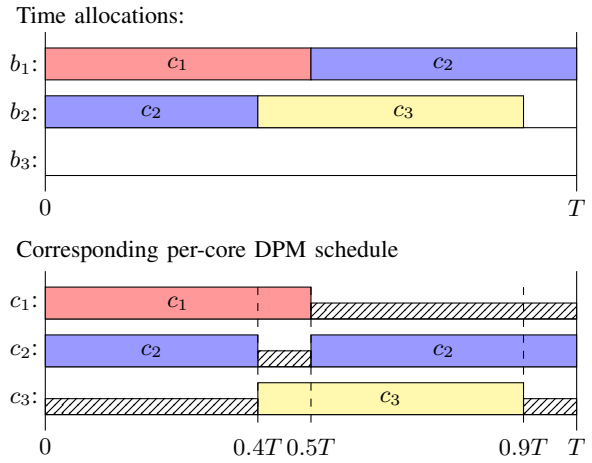



Fig. 3: Wrap-around example for 3 cores.

Consequently, for testing the schedubility of the task set, if it holds that TDP $\geq \pi$, then this task set is feasible.

*Theorem 2:* At least $\lceil U \rceil$ processing cores must be simultaneously powered on to feasibly schedule a set of frame-based real-time tasks with a cumulative utilization of $U$.

*Proof:* Suppose that for a task set with cumulative utilization $U$, we have a feasible schedule in which the number of simultaneously activated cores is less than $\lceil U \rceil$. Then, it follows from the pigeon-hole principle that at least one of the cores must have a utilization more than 1, hence the task set is infeasible. This is a contradiction. ■

A trivial indication from Theorem 2 is that the wrap-around rule minimizes the number of simultaneously activated processing cores. Hence, it gives an optimal solution for the peak power minimization for homogeneous tasks scheduled on homogeneous cores, regardless of the task partitioning scheme used in *Step 1*.

**Remark:** Furthermore, this algorithm can also be used to approximate the cases in which the power consumption of the task set is non-homogeneous or the case in which the capabilities of the cores are heterogeneous.

## VI. Homogeneous Tasks on Heterogeneous Cores

In this case we assume that cores differ in their power consumption. This can be the result of manufacturing variations or complexity of the core due to added hardware accelerators. However, the power consumption is independent of the tasks. That is, $\forall i \in \{2, \ldots, m\}\ p_{i,j} = p_{1,j}$ for every core $j$.

This case is particularly relevant to heterogeneous architectures where the cores differ so much from each other due to their capabilities that the power consumption profile of the cores is practically dependent only on the core being used. An example of such architecture is ARM big.LITTLE [13].

To solve this case we present a greedy approach called Least Density First (LDF). This approach works as follows. Firstly, we calculate the individual core utilizations, $\mathbf{W}$, as in Section V. That is, using the partition matrix $\mathbf{K}$ and the utilization matrix $\mathbf{U}$, we can fill matrix $\mathbf{W}$, such that $w_j = \sum_{i=1}^{n} k_{i,j} \cdot u_{i,j}$ for all $j = 1, 2, \ldots, m$. We start with the core that has the highest power consumption and assign it $\lceil w_j \cdot q \rceil$ slots from $q$ total slots. After this, we update a density state vector, which is null initialized, by summing the total power consumption for all slots. For the next core, we again assign $\lceil w_j \cdot q \rceil$ slots with the lowest density so far, and update the density state vector, again. This is done iteratively for each core in $\mathbf{W}$. The pseudo code for this scheme is presented in Algorithm 1.

**Algorithm 1:** Least Density First (LDF)

**Input**:
Individual peak power usage: $\mathbf{R} = [r_s]_{1\times m}$,
Per-Core Utilizations: $\mathbf{V} = [v_s]_{1\times m}$,
Total time slots: $q$.
**Output**:
Net peak: $\pi$
Schedule assignment: $\mathbf{A} = [a_{s,t}]_{m\times q}$,
s.t. $a_{s,t} = \begin{cases} 1 & \text{if } s^{th} \text{ resource is to be used at } t \\ 0 & \text{otherwise.} \end{cases}$

**Algorithm:**

Density state: $\mathbf{D} = [d]_{1\times q} \leftarrow [0]_{1\times q}$
$\mathbf{A} = [a_{s,t}]_{m\times q} \leftarrow [0]_{m\times q}$
**while** $\mathbf{R}$ *not empty* **do**
  Sort $\mathbf{D}$ ascending;
  Select and remove largest $r_s \in \mathbf{R}$;
  Assign first $\lceil v_s \times q \rceil$ slots to be used for $r_s$;
  Mark corresponding $a_{s,t} \in \mathbf{A}$ =1;
  Update $\mathbf{D} : \mathbf{D} \leftarrow \mathbf{D} + r_s \cdot \{a_{s,t} : \forall t \in (1, \ldots, q)\}$;
$\pi = \max(\mathbf{D})$;

Formally, we set $\mathbf{R}$ and $\mathbf{V}$ as follows to use Algorithm 1:

- $\mathbf{R}$ = per-core power consumption, i.e., $\mathbf{R} = [e_j]$ where $e_j = \sum_{i=1}^{n} k_{i,j} \cdot p_{i,j}\ \forall j \in \{1, \ldots, m\}$. The length of $\mathbf{R}$ is $m$.
- $\mathbf{V}$ = per-core utilization, i.e., $\mathbf{V} = [w_j]$ where $w_j = \sum_{i=1}^{n} k_{i,j} \cdot u_{i,j}\ \forall j \in \{1, \ldots, m\}$.

The output matrix, $\mathbf{A}$, from Algorithm 1 gives the schedule for the cores with the added sleep cycle. This schedule is feasible if the peak power consumption, $\pi$, is within the TDP. Here, $\pi$ can be used as the basis for an offline, sufficient feasibility test for the task set. The problem of generating a schedule that minimizes the peak power is $NP$-hard, as shown in Theorem 3.

The working of this algorithm is explained in the example shown in Figure 4. Here we have three cores with $c_1$, $c_2$ and $c_3$ with utilizations of $0.6, 0.5, 0.9$ and power consumption values of $3, 4, 2$ Watts. First, we choose the core with the highest power consumption, $c_2$ and assign it the required slots and update the density vector $\mathbf{D}$. Here, the height of the bars represents the density. After this, we choose the next highest power consuming core and assign it the required number of lowest density slots and so on. This results in the per-core schedule shown in the third iteration in Figure 4, and the peak in power consumption is 7 Watts.

LDF is a polynomial time algorithm. Since there is one multiplication and two sorting operations involved, one for the cores' utilization vector ($\log m$) and another one for the

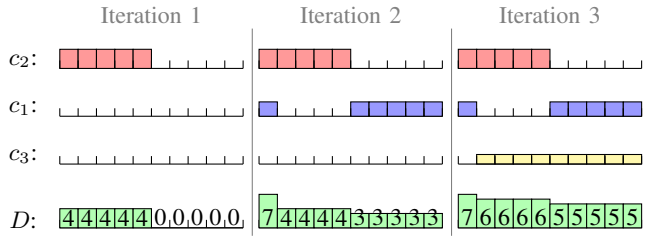



Fig. 4: Example for least density first algorithm

density vector ($q \log q$) inside a loop that iterates $m$ times, the worst case time complexity of the algorithm is bounded by $\mathcal{O}(m(mq + \log m + q \log q))$.

*Theorem 3:* For a given task partitioning of homogeneous tasks onto heterogeneous cores, deriving a schedule for cores to optimally minimize the peak power is an $NP$-hard problem.

*Proof:* We reduce from the optimization version of the partitioning problem. Given a set of $m$ numbers, the optimization version of the partitioning problem is to divide them into two disjoint subsets, $A$ and $A'$, such that the difference of sum of the numbers in each subset is minimized. Such a problem is NP-hard [11].

The reduction works as follows. Consider the special case of our problem in which we have $m$ tasks with utilization $50\%$, every task has a different peak power consumption, and the deadline for the frame-based tasks of 1. The objective is to decide whether a core is executed in the window of $(0, 0.5]$ or $(0.5, 1]$, such that the final peak power consumption is minimized, which is the same as minimizing the difference between the peak power of the windows.

This problem is equivalent to the optimization version of the partitioning problem, where the peak power consumption of the tasks represents the numbers to partition. Deciding to execute a core in the window of $(0, 0.5]$ is equivalent to putting the number of its power consumption to set $A$. Similarly, deciding to execute a core in the window of $[0.5, 1)$ is equivalent to putting the number of its power consumption to set $A'$.

Therefore, an optimal solution to the optimization version of the partitioning problem is equivalent to an optimal solution to this special case of our studied problem and vice versa. Hence, we conclude the $NP$-hardness of our studied problem. ■

## VII. Heterogeneous tasks on homogeneous/heterogeneous cores

In this section we present an algorithm for two cases, that is, (i) the power consumption is dependent on the task but not on the core it executes on, and, (ii) the power consumption is dependent on the task as well as the core on which the task executes.

The former case applies to the wide variety of multi-core chips currently available in the market, where several identical cores are available on the chip and the power consumption profile of tasks differ from each other due to resource access pattern. Whereas, the latter case is a generalization to include many-core chips such that a core might consume less power for one task and more power for another task, and there might exist another core in the system whose power consumption values are reversed for the same two tasks. This is the most general case, with all the previous cases being a special form of this case.

**Algorithm 2:** LDF with occupancy check

**Input**:
Power matrix: $\mathbf{P} = [p_{i,j}]_{n \times m}$,
Partition matrix: $\mathbf{K} = [k_{i,j}]_{n \times m}$,
Utilization matrix: $\mathbf{U} = [u_{i,j}]_{n \times m}$,
Maximum number of time slots: $q$.
**Output**:
Peak power consumption: $\pi$,
Schedule assignment: $\mathbf{A} = [a_{j,t}]_{m \times q}$,
s.t. $a_{j,t} = \begin{cases} i & \text{if } i^{th} \text{ task scheduled at core } j \text{ for } t \\ 0 & \text{otherwise.} \end{cases}$
**Algorithm:**

Density state: $\mathbf{D} = [d]_{1 \times q} \leftarrow [0]_{1 \times q}$
$\mathbf{A} = [a_{j,t}]_{m \times q} \leftarrow [0]_{m \times q}$
**while** $\mathbf{P}$ *not empty* **do**
  Sort $\mathbf{D}$ ascending;
  Select and remove largest $p_{i,j} \in \mathbf{P}$;
  Assign $\lceil k_{i,j} \cdot u_{i,j} \cdot q \rceil$ globally least dense and locally free slots in core $j$ to task $n$;
  Set corresponding $a_{i,j} \in \mathbf{A} = i$;
  Update $\mathbf{D} : \mathbf{D} \leftarrow \sum_{j=1}^{m}$ power usage at slot $t \; \forall t \in (1, \ldots, q)$;
$\pi = \max(\mathbf{D})$;

We present a variation of Least Density First algorithm to solve both these cases. The variation from the earlier presented Least Density First algorithm is explained as follows. In the normal version of LDF the only decision criteria is based on density. However, in this algorithm we also check the assignment of the slots. We start with the highest power consuming task and assign it the lowest density *free* slots in the core in which this task has been partitioned, and this process is repeated for all tasks. The extra step of checking the free slots is necessary as the globally lowest density slots might already be occupied in the core of interest. Since we are only considering partitioned scheduling, i.e., each task is assigned to only one core, hence we do not need to check for the condition in which the same task is concurrently scheduled at two cores. The pseudo code for the procedure is presented in Algorithm 2.

Like in the previous section, the matrix $\mathbf{A}$ gives us the schedule of the cores. This schedule is feasible if $\pi$ is less than or equal to the TDP.

This is also a polynomial time algorithm. Here, we sort the $\mathbf{P}$ array of size $n \times m$, and for every iteration of the loop we sort the $\mathbf{D}$ array of $q$ numbers. The worst case time complexity of this algorithm is given as $\mathcal{O}(nm(mq + \log nm + q \log q))$.

## VIII. Implicit Deadline Periodic Tasks

The solutions presented in Sections V, VI and VII are only applicable to frame-based task sets. In this section, we present a method to extend the previous results to include

implicit deadline periodic task sets. We include the following tasks types in the task model. Each task $\tau_i$ releases an infinite number of task instances (jobs) with period $T_i$ and relative deadline $D_i$, where $D_i = T_i$. We assume that the tasks are synchronized, that is, the first job of each task arrives at the same instant. Frame-based tasks are a special form of implicit deadline periodic tasks.

The core idea here is that an implicit deadline real-time task set can be feasibly scheduled until its utilization on each core does not exceed 100%. Therefore, the slack left after task partitioning can be reclaimed to halt the processing cores in a coordinated manner to decrease the peaks in power consumption.

*Theorem 4:* A feasible, implicit deadline task set with periods $\{T_1, T_2, \cdots, T_n\}$ and with cumulative utilization $U$ can be feasibly scheduled with *Earlier-Deadline-First* (EDF) policy, if the processing core is put to sleep for $(1-U)$ fraction of time in every $\Delta$ interval, where $\Delta$ is the greatest common divisor (GCD) of $(T_1, T_2, \cdots, T_n)$ and is synchronized with the tasks.

*Proof:* Consider an implicit deadline task set with periods $\{T_1, \cdots, T_n\}$ and worst case execution times $\{C_1, \cdots, C_n\}$ which is feasibly schedulable with EDF policy. Consider that the sleep time of the processing core is an additional implicit deadline task, $\tau_s$, with period $\Delta$ and execution time of $\Delta(1-U)$, where $U = \sum_{j=1}^{n} \frac{C_j}{T_j}$.

Assume that after introducing $\tau_s$ the system cannot be feasibly scheduled using EDF policy and a job, $J_{i,k}$, of a task, $\tau_i$, misses its absolute deadline, $d_{i,k}$. Suppose that $t_0$ is the last instant before $d_{i,k}$ when the system was idle. If such an instant does not exist, then $t_0$ denotes the starting time of the system. Since the system cannot be feasibly scheduled with EDF, then it must hold that:

$$
\begin{aligned}
d_{i,k} - t_0 &< \sum_{j=1}^{n} \left\lfloor \frac{d_{i,k} - t_0}{T_j} \right\rfloor C_j + \left\lfloor \frac{d_{i,k} - t_0}{\Delta} \right\rfloor \Delta(1-U) \\
\implies 1 &< \sum_{j=1}^{n} \frac{C_j}{T_j} + (1-U) \\
\implies 1 &< U + 1 - U \\
\implies 1 &< 1
\end{aligned}
$$

We reach a contradiction. Hence the assumption that a task set, originally feasibly schedulable with EDF policy, becomes infeasible with the addition of $\tau_s$ with period $\Delta$ and execution time $\Delta(1-U)$, is invalid and the theorem is proven. ■

Applicability of the earlier presented algorithms to implicit deadline periodic tasks is a more general result but an overhead can be expected due to the additional switching transitions to and from the sleep mode. Since, $\Delta$ is the greatest common divisor of the periods of all tasks, it can be small, and, as a sleep cycle has to be placed in each $\Delta$, this can make the system infeasible if there is high timing overhead associated with putting the system to sleep.

For periodic tasks, in case of homogeneous tasks on homogeneous cores, the feasibility of real-time constraints originating from peak power consumption can be verified by the same test as introduced in Section V. The peak power is given as $\pi = \lceil U \rceil \cdot p_{\text{const}}$, where $U$ is the cumulative utilization and $p_{\text{const}}$ is the power consumption of any core.

| Application | Period [ms] | Dynamic power usage (W) | |
|---|---|---|---|
| | | P54C | Alpha |
| x264 | 30 | 0.70 | 0.66 |
| bodytrack | 30 | 1.00 | 0.81 |
| swaptions | 450 | 0.60 | 0.74 |
| blackscholes | 900 | 0.50 | 0.70 |

TABLE I: Specs. of applications used for simulations

For the most general case, i.e., heterogeneity of either task set or both the task set and the processing cores, the test introduced in the Section VII for verifying the feasibility of real-time constraints for the second version of LDF (Algorithm 2) can be utilized repeatedly. Since there is a sleep period included in the schedule for every core in the system in every $\Delta$ interval, we can use LDF (Algorithm 2) within each $\Delta$ interval to coordinate the sleep periods of cores to minimizes the peak power. The peak power, $\pi$, for each $\Delta$ is known from Algorithm 2. To find the highest peak that can occur, the system must be analyzed for an interval equal to the hyper period of the system. A hyper period is that interval after which the system repeats itself. It is equivalent to the least common multiple of the periods of the task set. Concretely, we employ the task partitioning algorithm as discussed previously (Section IV-A) and use Algorithm 2 with $\mathbf{P}$, $\mathbf{K}$ and $\mathbf{U}$ matrices obtained from the partitioning. Here, we set the number of slots, $q$, equal to $\Delta$. Algorithm 2 returns the value of peak, $\pi$ for this $\Delta$. The same process is repeated for next $\Delta$ intervals till the total analyzed period equals a hyper period. The highest peak among all $\Delta$ intervals belonging to the hyper period, is used to decide the feasibility.

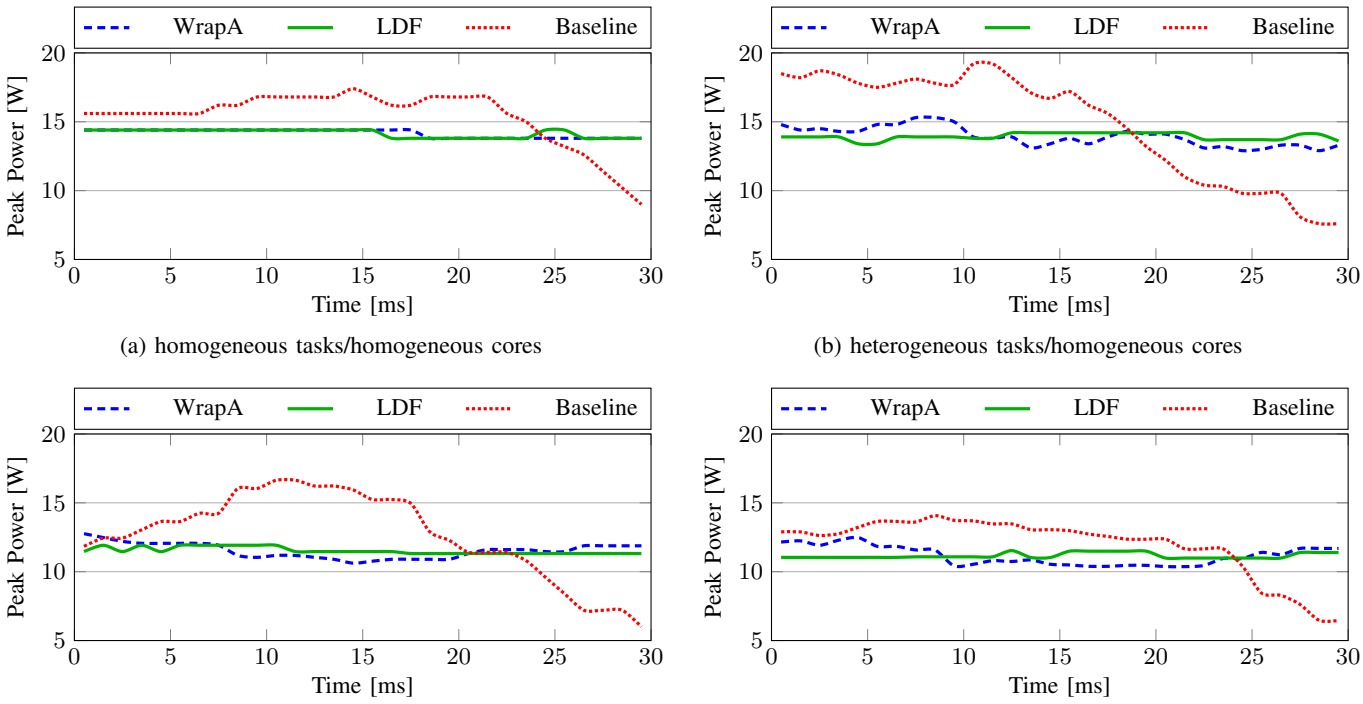
Similarly, for homogeneous tasks on heterogeneous cores Algorithm 1 can be utilized repeatedly for one hyper period and the highest value of $\pi$ can be compared against TDP for feasibility testing.

## IX. Results and Discussion

In this section we present the results of our simulations. To evaluate our scheme we used applications from *Parsec* benchmark suite [3] running on the SCC platform and on Alpha cores that we simulate using gem5 and McPAT infrastructure. To highlight the difference between peak power minimization and average power minimization, i.e., energy minimization, we compare the presented algorithms with a well known energy minimization scheme. The details of the setup and the results are presented below.

### A. Platform details

We use Intel's 48 core SCC platform [15] for power measurements. This platform is equipped with 48 Pentium (P54C) cores which are based on 45nm manufacturing process. The cores are distributed into evenly placed 24 tiles with 2 cores per tile. A network on chip in mesh topology allows the inter-core communication. We use four applications from *Parsec* benchmark suite [3] and individually obtain the peak power consumption for each, using the on-board instrumentation. Details of the applications used to collect the power traces are presented in the next section. SCC offers a tiled architecture




(a) homogeneous tasks/homogeneous cores

(b) heterogeneous tasks/homogeneous cores

(c) homogeneous tasks/heterogeneous cores

(d) heterogeneous tasks/heterogeneous cores

Fig. 5: Power consumption profile for each period (frame-based tasks)

with the so-called voltage and frequency islands. The voltage can only be changed at the granularity of 8 cores and frequency at the granularity of 2 cores. As a precursor to future many-core chips, it shows that DPM based power control will remain an essential control 'knob' in future chips.

Since SCC platform only has homogeneous cores, it cannot be used to measure the effect of core-heterogeneity. For this, we simulated a synthetic platform based on SCC's architecture and same dimensions, but with 24 Alpha cores [21] replacing 24 Pentium cores, one in each tile. Alpha cores are also based on 45nm manufacturing process and are simulated using gem5 [4]. The peak power measurements for the Alpha cores are obtained through simulation using McPAT. Alpha and Pentium cores differ in their power consumption as well as computational performance but are based on the same manufacturing technology. This makes them a good candidate for judging the efficacy of this work. We run the same applications on both types of cores to measure the power profiles for each application.

In the following discussions, the results based on the homogeneous cores (Figures 5a, 5b, 7 and 9) are from the SCC, while the ones based on the heterogeneous cores (Figures 5c, 5d, 6 and 8) are from the simulated platform.

### B. Real-time workload

We generate *synthetic* workloads for the simulations. Our scheme of workload generation is based on the widely used technique in real-time community presented in [1]. We iteratively generate a randomized task set, where we add a new task to the set at each iteration. Each task has three parameters: (1) utilization, (2) peak power and (3) period. These three parameters are obtained as follows. For each task, the utilization is assigned randomly in the interval $(0, u_l)$, where $u_l$ is the minimum of 1 and left over utilization in the platform. We stop adding more tasks to the task set when $u_l \leq 0.05$. The peak power consumption is chosen among the actual measurements that were performed using the SCC platform and the simulated Alpha core on gem5. We use the traces collected from four applications from the *Parsec* benchmark suite [3] to sample their peak power consumption as shown in Table I. Parsec benchmark suite has 13 applications. In order to curtail evaluation time, we randomly selected four out of these and executed their single core versions. Here, *x264* is an H.264 video encoder, *bodytrack* is computer vision application that tracks human body whereas *swaptions* and *blackscholes* are financial analyses applications. For the third parameter: the period of the real-time workload, in the first step, we select the period for all tasks to be 30ms, in order to simulate a frame-based task set. In the second step, we use the actual periods as mentioned in Table I for implicit deadline periodic tasks.

After generating the task set, we use the partitioning algorithm presented in [2] to partition the tasks into the available cores. Since this algorithm guarantees to find a feasible mapping if, at most, half of the processing capacity is utilized, we use the system utilization of 24 where not mentioned otherwise. Note that it is not possible to cause thermal throttling in case of SCC due to an overly pessimistically designed heat sink.

### C. Baseline Scheme

To compare our scheme, we use a well known energy minimization procrastination scheme presented in [17]. The

schedule obtained using [17] minimizes the energy consumption for tasks with real-time requirements. In essence, the energy minimization scheme procrastinates the tasks as much as possible without violating the performance constraints. In its original form, it activates all the cores towards the end of the schedule, causing a peak in the power consumption at the end of the period for frame-based tasks. To suppress this peak towards the end of the schedule, we *modify* the scheme by keeping $m^*$ cores activated through out the period, starting with the most power consuming cores, and activating more in the end to meet performance requirements. Here, $m^* = \left\lfloor \sum_{j=1}^{m} \sum_{i=1}^{n} u_{i,j} \cdot k_{i,j} \right\rfloor$, i.e., floor of the total system utilization. By keeping a subset of cores activated throughout the period, the power consumption is distributed over the length of whole period and the peaks in power consumption are suppressed. This provides the basis for a fair comparison.

### D. Results

Our focus in the paper has been on minimizing the peak power. We present two important results in this regard.

Firstly, we show a simple comparison of PPM against a well known baseline scheme for energy minimization for both varieties of tasks and cores. We generate the schedule required for the different combinations of applications considering their real-time requirements using our scheme and the baseline. Initially, we consider frame-based tasks with the period to be 30ms. To simulate the case of homogeneous task sets we only use *swaptions* on all cores. To simulate the case of homogeneous cores we use only the power data from the SCC.

*Frame-based Tasks:* For frame-based tasks, the results are summarized in Figure 5. Here we can see that in all four cases our scheme produces a more balanced power consumption profile as compared to the baseline. Since we prefer the least dense slots (LDF), the power consumption of our scheme gets distributed over the whole period, which helps in avoiding peaks.

The wrap-around method (WrapA) was basically designed for homogeneous tasks on homogeneous cores and it solves this case optimally. In Figure 5a it can be seen that peak produced by WrapA is not higher than that of LDF, although not at the same point in schedule. WrapA achieves peak power consumption of 3 Watts less than the baseline scheme. In the rest of the three cases, that is, when either tasks or cores are non-homogeneous in their power consumption (Figures 5b, 5c and 5d), it does not produce optimal results. Nevertheless, this method still fares better than the baseline scheme and its maximum deviation from the LDF remains less than 10%. The greedy approach employed in the LDF method achieves better results than both the baseline scheme and WrapA in this case.

In the second case, we evaluate the effect of increasing the load on the peak power. The results are summarized in Figure 6. In this case we only consider the general case of heterogeneous tasks on heterogeneous cores. As expected, it can be seen that as the workload increases, the peak in power starts to grow for all three schemes. Here again, we observed that maximum deviation of WrapA method does not exceed 10% (1.58 Watts) from the value achieved by LDF, whereas baseline scheme deviates up to 35.5% (6.08 Watts).

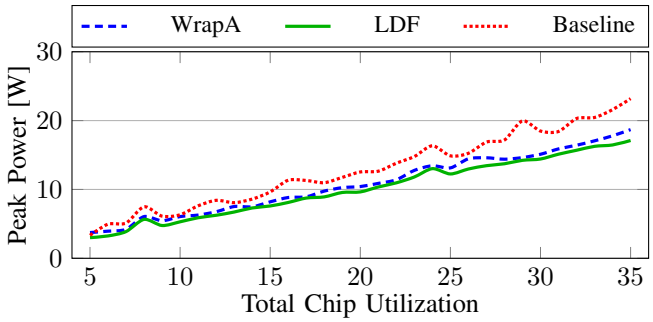



Fig. 6: Effect of increasing workload on peak power consumption

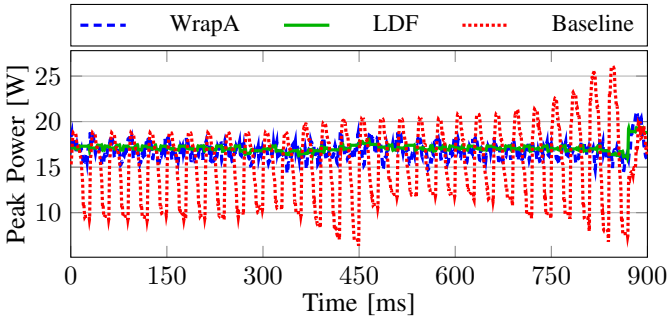



Fig. 7: Periodic workload: heterogeneous tasks on homogeneous cores

The maximum deviation for both WrapA and the baseline scheme was observed at the total chip utilization of 35, with a general trend of higher deviations with increasing utilizations for both.

*Non Frame-based, Periodic Tasks:* The results for the non frame-based periodic tasks are presented in Figures 7 and 8. Here we use the periods and power profiles for the tasks as mentioned in Table I. We assign the utilization randomly according to the methodology introduced in [1]. For periodic tasks we only present the results for non-homogeneous task set on both homogeneous and heterogeneous cores. Hence, we employ Algorithm 2 here. In this case, the difference between the *energy* minimization vs. *peak* power minimization becomes quite apparent. The baseline scheme used is designed with the perspective of energy minimization. In Figure 8, the baseline has a peak power consumption of 5 Watts more than LDF and 3 Watts more than WrapA, whereas the baseline consumes less

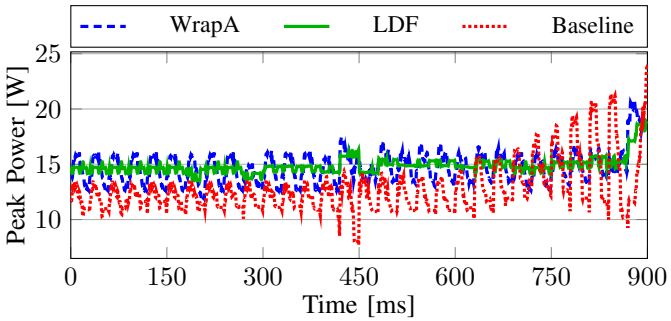



Fig. 8: Periodic workload: heterogeneous tasks on heterogeneous cores

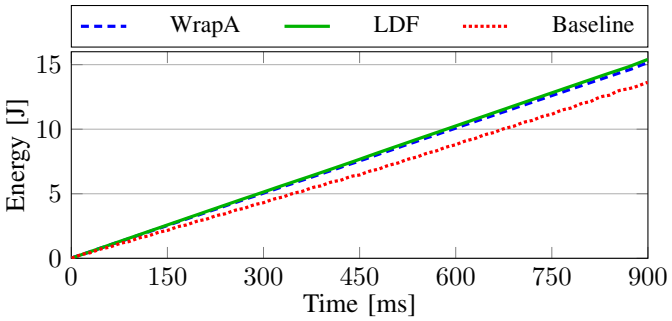



Fig. 9: Cumulative energy consumption: Periodic workload, heterogeneous tasks on homogeneous cores

energy as shown in Figure 9. In the case of homogeneous cores (Figure 7), the difference in peak power consumption is even bigger. In both cases, LDF and WrapA keep a more balanced power profile, like in the case of frame based tasks, and for this reason they are able to suppress the peaks, specially towards the end of the schedule where the baseline scheme causes a peak in order to fulfill the performance requirements of real-time tasks that have been back-logging during the procrastination. The peaks caused at every 30, 450 and 900 ms correspond to the periods of the tasks used for evaluation.

## X. Conclusions

In this paper we presented solutions to minimize the peak power consumption for executing real-time tasks on many core architectures which can help contain the power consumption within the TDP constraint. We argued that the peak power consumption of a real-time task set must also be verified when deciding its scheduling feasibility. The presented peak power management scheme follows a two step procedure: first the tasks are partitioned on to the available cores and the schedule for each core is decided. Afterwards, our solution minimizes the peak power consumption for systems with: *homogeneous tasks on homogeneous cores*, *either heterogeneous cores or tasks*, and with both *heterogeneous tasks on heterogeneous cores*. This is achieved by putting the cores to sleep mode at appropriate points in time, without affecting the tardiness of real-time tasks. For this, we presented algorithms with polynomial-time complexity. We simulated our scheme using power traces for two platforms; SCC and a heterogeneous core platform based on the SCC design. Our results show the efficacy of our scheme for peak power minimization.

## XI. Acknowledgment

This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre *Invasive Computing* (SFB/TR 89); http://invasic.de.

## References

[1] T. P. Baker. Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hard real time. Technical report, TR-050601, 2005.

[2] S. K. Baruah. Task partitioning upon heterogeneous multiprocessor platforms. In *RTAS '04*.

[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: characterization and architectural implications. In *PACT '08*.

[4] N. Binkert, B. Beckmann, et al. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2), Aug. 2011.

[5] J.-J. Chen, H.-R. Hsu, K.-H. Chuang, C.-L. Yang, A.-C. Pang, and T.-W. Kuo. Multiprocessor energy-efficient scheduling with task migration considerations. In *ECRTS '04*.

[6] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *RTAS '06*.

[7] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35, 2011.

[8] R. Dennard, F. Gaensslen, V. Rideout, E. Bassous, and A. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *Solid-State Circuits*, 9(5):256–268, Oct 1974.

[9] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, et al. Dark silicon and the end of multicore scaling. In *ISCA '11*.

[10] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele. Thermal-aware global real-time scheduling on multicore systems. In *RTAS '09*.

[11] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman and Co., 1979.

[12] R. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:263–269, 1969.

[13] P. Greenhalgh. Big. little processing with arm cortex-a15 & cortex-a7. *ARM White Paper*, 2011.

[14] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Toward dark silicon in servers. *Micro-44*, 2011.

[15] J. Howard, S. Dighe, S. R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. K. De, and R. F. V. der Wijngaart. A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and DVFS for performance and power scaling. *J. Solid-State Circuits*, 46(1):173–183, 2011.

[16] Intel Corporation. Desktop 3rd generation intel core processor family - thermal mechanical specifications and design guidelines, Jan 2013.

[17] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *DAC '04*.

[18] S. Kodase, S. Wang, Z. Gu, and K. G. Shin. Improving scalability of task allocation and scheduling in large distributed real-time systems using shared buffers. In *RTAS'03*.

[19] J. Lee, B. Yun, and K. Shin. Reducing peak power consumption in multi-core systems without violating real-time constraints. *Parallel & Distributed Sys. '13*.

[20] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: a power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO-42.*, 2009.

[21] E. J. McLellan and D. A. Webb. The alpha 21264 microprocessor architecture. In *ICCD '98*.

[22] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959.

[23] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin. Hierarchical power management for asymmetric multi-core in dark silicon era. In *DAC '13*.

[24] S. Pagani and J. Chen. Energy efficiency analysis for the single frequency approximation (SFA) scheme. In *RTCSA '13*.

[25] S. Pagani and J. Chen. Energy efficient task partitioning based on the single frequency approximation scheme. In *RTSS '13*.

[26] B. Raghunathan, Y. Turakhia, S. Garg, and D. Marculescu. Cherry-picking: exploiting process variations in dark-silicon homogeneous chip multi-processors. In *DATE '13*.

[27] J. Sartori and R. Kumar. Distributed peak power management for many-core architectures. In *DATE '09*.

[28] M. Shafique, S. Garg, J. Henkel, and D. Marculescu. The EDA challenges in the dark silicon era: Temperature, reliability, and variability perspectives. In *DAC '14*.