TREAFET: Temperature-Aware Real-Time Task Scheduling for FinFET based Multicores

SHOUNAK CHAKRABORTY, Department of Computer Science, Norwegian University of Science and Technology (NTNU), Norway
YANSHUL SHARMA and SANJAY MOULIK, Department of Computer Science and Engineering, Indian Institute of Information Technology (IIIT) Guwahati, India

The recent shift in the VLSI industry from conventional MOSFET to FinFET for designing contemporary chip-multiprocessor (CMP) has noticeably improved hardware platforms' computing capabilities, but at the cost of several thermal issues. Unlike the conventional MOSFET, FinFET devices experience a significant increase in circuit speed at a higher temperature, called temperature effect inversion (TEI), but higher temperature can also curtail the circuit lifetime due to self-heating effects (SHEs). These fundamental thermal properties of FinFET introduced a new challenge for scheduling time-critical tasks on FinFET based multicores that how to exploit TEI towards improving performance while combating SHEs. In this work, *TREAFET*, a temperature-aware real-time scheduler, attempts to exploit the TEI feature of FinFET based multicores in a time-critical computing paradigm. At first, the overall progress of individual tasks is monitored, tasks are allocated to the cores, and finally, a schedule is prepared. By considering the thermal profiles of the individual tasks and the current thermal status of the cores, hot tasks are assigned to the cold cores and vice-versa. Finally, the performance and temperature are balanced on-the-fly by incorporating a prudential voltage scaling towards exploiting TEI while guaranteeing the deadline and thermal safety. Moreover, *TREAFET* stimulates the average runtime frequency by employing an opportunistic energy-adaptive voltage spiking mechanism, in which energy saving during memory stalls at the cores is traded off during the time slice having the spiked voltage. Simulation results claim *TREAFET* maintains a safe and stable thermal status (peak temperature below 80 °C) and improves frequency up to 17% over the assigned value, which ensures legitimate time-critical performance for a variety of workloads while surpassing a state-of-the-art technique. The stimulated frequency in *TREAFET* also finishes the tasks early, thus providing opportunities to save energy by power gating the cores, and achieves a 24% energy delay product (EDP) gain on average.

CCS Concepts: • **Computer systems organization → Real-time systems**; **Embedded and cyber-physical systems**; • **Hardware → Thermal issues**.

Additional Key Words and Phrases: real-time systems, scheduling, FinFET, multicores, thermal and energy efficiency, TEI, SHE, dynamic thermal management, memory stalls, core frequency, voltage scaling

ACM Reference Format:
Shounak Chakraborty, Yanshul Sharma, and Sanjay Moulik. 2018. TREAFET: Temperature-Aware Real-Time Task Scheduling for FinFET based Multicores. *ACM Trans. Embedd. Comput. Syst.* 00, 0, Article 000 (2018), 29 pages. https://doi.org/XXXXXXX.XXXXXXX

Authors' addresses: Shounak Chakraborty, Department of Computer Science, Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 7491, shounak.chakraborty@ntnu.no; Yanshul Sharma, yanshul.sharma@iiitg.ac.in; Sanjay Moulik, sanjay@iiitg.ac.in, Department of Computer Science and Engineering, Indian Institute of Information Technology (IIIT) Guwahati, Guwahati, Assam, India, 781015.

1539-9087/2018/0-ART000 $15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The recent shift from conventional MOSFET to FinFET for contemporary chip-multiprocessor (CMP) designs has noticeably improved hardware platforms' computing capabilities by lowering the short channel effects of MOSFET, but at the cost of several thermal issues specifically caused due to confined 3D geometrical shape of the FinFETs [41, 53, 60]. Due to the shape of FinFET, devices have become special in terms of their power and performance characteristics [34, 71, 78, 80]. Unlike the conventional MOSFET, FinFET devices experience a significant increase in circuit speed at a higher temperature, called temperature effect inversion (TEI) [20, 50]. This TEI might incorporate timing and synchronization issues during execution, which introduces new challenges for scheduling time-critical applications. Additionally, encapsulation of FinFET channel within a thermal insulator clogs heat dissipation that can potentially lead to circuit failure due to self-heating effect (SHE) [6]. Hence, developing novel scheduling strategies for FinFET based multicores that prudentially balance TEI and SHE is a prime requirement for the time-critical computing paradigm.

Scheduling real-time tasks has been becoming challenging in the case of multicore platforms over the years with a gradual increase in workload as well as computational resources. Towards legitimate distribution and utilization of the computational resources, semi-partitioned scheduling [8] has been proposed for multicore platforms having low task migration overheads. In such schedulers, the execution timeline is split into a batch of time slots known as *intervals*, and execution in systems happens interval by interval. The proposed scheduler also allows missing task deadlines within a stipulated bound but can offer efficient task utilization. In another approach [38], the authors proposed an optimal static scheduler to schedule soft real-time sporadic tasks based on the Earliest Deadline First (EDF) notion. A two-phase strategy was proposed by Casini et al. [22], where the first phase employs an approximation scheme to split the tasks, and the next phase is a load-balancing algorithm that limits the number of task migrations. A cluster-based scheduler was also devised [5], which initially partitions the tasks into clusters, each of which may contain a set of processors. Next, the tasks are scheduled using global EDF in every cluster.

Advancement in VLSI technology further drives real-time system researchers to include thermal management while developing efficient schedulers for modern multicore platforms [54, 82], where tasks are statically allocated based upon their thermal characteristics. Unfortunately, prior offline approaches are mostly based on the conventional MOSFET based system, hence did not consider the thermal characteristics of the FinFET. Over a decade, TEI in FinFET has been investigated, which increases the operational speed at higher temperatures even in the super-threshold voltage region [20, 21, 47, 49, 50]. Kim et al. [47] have explored this phenomenon by analyzing the circuit- and device-characteristics. By scaling supply voltage dynamically, Lee et al. [50] also proposed a thermal management technique for the FinFET devices while considering TEI. However, these prior techniques mostly focused on the TEI, but its performance impacts on the multicores were first evaluated by Cai and Marculescu [21]. Later, Neshatpour et al. [63] devised a TEI-aware DVFS that scales the cores' voltage/frequency (V/F) by exploiting on-chip thermal sensors. However, these earlier techniques did not consider SHE, which needs to be taken care of while exploiting TEI in FinFET devices, which can also be an interesting design choice for time-critical environments. In earlier work, we proposed *RESTORE* [72], which is a temperature-aware real-time scheduler for FinFET based multicores that govern the voltage/frequency of the cores by considering TEI and SHE phenomena of the FinFET. To the best of our knowledge, *RESTORE* is the first technique that considered TEI and SHE phenomena of the FinFET based multicores in the spectrum of time-critical environments. However, as a reactive mechanism, *RESTORE* considers the current temperature of the cores and regulates the voltage accordingly so that a legitimate frequency can be met. As core temperature depends upon the task's runtime characteristics, fine-grained thermal management of

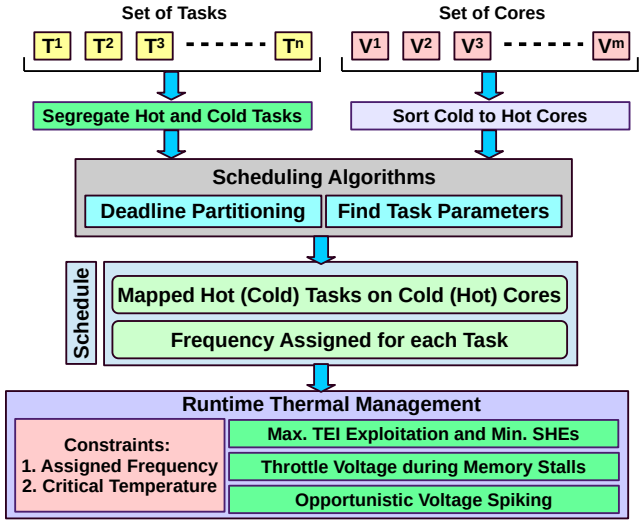


Fig. 1. Process Overview: *TREAFET*

the FinFET based time-critical system must account for the change in temporal thermal status of the individual tasks during execution, which was not included in *RESTORE*.

In this paper, we propose *TREAFET*, a two-phase temperature-aware real-time scheduling strategy for the multicore platforms that first schedules tasks at design time by assigning a particular runtime frequency for individual tasks. The entire strategy of *TREAFET* is depicted in Figure 1 (detailed in Sec. 3). At first, the task-to-core allocation is performed by considering the thermal characteristics of the individual tasks. On the other hand, the present thermal status of the cores is also analyzed, and the cores are also sorted as per their temperature values. Our proposed scheduler is based on the semi-partitioned approach and, hence, can offer high resource utilization with a limited number of migrations. The task execution is divided into several intervals, where the interval boundaries act as pseudo-deadlines for each task. This feature not only helps the scheduler maintain a steady rate of progress for all tasks but also meets their final task deadlines. Overall, our proposed semi-partitioned scheduler considers the thermal characteristics of individual tasks and the current thermal status of each core and maps hot (cold) tasks to cold (hot) cores. For each task, the scheduler also assigns a particular frequency so that the deadline is not violated. During execution, *TREAFET* will prudentially apply the dynamic voltage scaling (DVS) mechanism by considering the core temperature, with an objective of balancing TEI and SHE properties of the FinFET. As the tasks are known beforehand, *TREAFET* detects the different execution phases of each task and the runtime temperature of the core to apply DVS strategically so that exploitation of TEI benefits can be enhanced while maintaining a safe temperature and meeting the real-time constraint. Additionally, during costly memory stalls at the cores, the supply voltage is throttled to save power, while an opportunistic energy-adaptive voltage spike is applied just after the memory stall to improve performance. To the best of our knowledge, *TREAFET is the first real-time scheduling approach that accounts for TEI and SHE phenomena of the FinFET based multicore along with an intense focus on the runtime task-characteristics.*

The contributions of *TREAFET* can be listed as follows:

- In **Scheduling Phase** (detailed in Sec. 3.2)-

  - the interval based independent tasks are allocated to the cores based on the tasks' thermal profiles;
  - the current temperature of the individual cores is considered at the beginning of each interval to allocate hot tasks to the cold cores and vice versa;
  - each task is also assigned a specific frequency and the minimum frequency value at which the task needs to be executed to meet the deadline.
- For each task, the **Runtime Thermal Management** (detailed in Sec. 3.3)-
  - intends to execute the hotter execution phase with increased TEI exploitation while limiting the SHEs by employing DVS prudentially towards maintaining an average preset frequency so that deadlines are not violated;
  - detects and exploits the costly memory stalls induced by the last level cache (LLC) misses and prudentially throttles the core-voltage towards balancing TEI exploitation vs. mitigating SHEs while guaranteeing the performance;
  - employs an energy-adaptive opportunistic voltage spiking just after completion of the memory stall interval to improve the performance;
  - exploits slacks generated online due to performance improvement for energy saving by turning off the cores.

Simulation based results (detailed in Sec. 5) show that *TREAFET* is able to maintain a safe and stable peak temperature of 80 °C even with 100% system utilization. Overall, by employing TEI-aware and runtime energy-adaptive voltage scaling mechanism, *TREAFET* improves core frequency up to 17% over the assigned frequency while scheduling the tasks. By stimulating core frequency, *TREAFET* reduces the task execution time that generates slacks within the interval, which are further exploited to improve energy efficiency by power gating the processor cores, and thus a 24% average gain in energy delay product (EDP) is achieved while surpassing a prior art [63].

2 BACKGROUND AND SYSTEM MODEL

This section will briefly discuss the semi-partitioned scheduling, and thermal characteristics of FinFET based CMPs, along with the system model used in this work.

2.1 Semi-Partitioned Scheduling

Multicore schedulers are broadly classified as either *global* or *partitioned*. In global scheduling [10, 28, 70], all ready tasks are added in a single priority queue and at each scheduling point, the highest priority m tasks are scheduled on m available cores of the system. Although such schedulers have several advantages like automatic load balancing among cores, simple implementation, etc., they also suffer from a high number of migrations [14, 29]. This is attributed to the fact that at each scheduling point, there might be a migration. In partitioned scheduling [30, 33, 55], there are separate priority queues for each available core and once a task is allotted to a core, it is not permitted to migrate. Hence, there is no migration cost involved in such scheduling. Further, each core may use an optimal single core scheduling strategy like Earliest Deadline First, Rate Monotonic Scheduling, etc., which may be different from the strategy followed by other cores in the system. However, the problem of optimal task-to-core allocation is NP-Hard [12, 44]. These schedulers use various heuristics like First Fit, Next Fit, Worst Fit, etc., to perform the task-to-core allocation. Further, these schedulers also suffer from low resource utilization, which is a side-effect of not allowing migrations. Hence, hybrid strategies are employed for preparing multicore scheduling strategies, which can be categorized as *semi-partitioned* [9, 43]. In such schedulers, the execution timeline is split into batches of time slots known as intervals, using some heuristics. In each such split, a single schedule is prepared for the interval at the beginning, and the number of migrations

is restricted by a preset threshold. But if an interval is short, the scheduler will behave as a global one. On the other hand, a large interval will indicate a partitioned scheduler. Further, if a task has a deadline within an interval, which is very likely, the scheduler must consider it as a constraint. In *TREAFET*, we will use a heuristic known as *deadline partitioning* [66, 67] to achieve such time splits, which have the advantage of having task deadlines only at the interval boundaries.

2.2 Thermal Aspects of FinFET CMPs

The thermal status of any on-chip component obeys the basic *superposition and reciprocity* principle of the heat transfer, which is driven by three prime factors: *(1)* the component's own power consumption, *(2)* heat abduction by ambient, and *(3)* conductive heat transfer with its peers [76]. Out of these three factors, the power consumption of the component plays the most vital role in determining its thermal status, especially those built in sub-14nm technology, due to encapsulation of the FinFET channel by the insulator [6, 7, 45, 53, 60, 80]. Hence, to ensure the thermal safety of these devices, prudential control of power consumption can be a potential optimization knob.

We represent the dynamic power consumption of a core as Pow_{Dyn}, which is proportional to the supply voltage, V_{dd}, and the operational frequency, F, of the core. Pow_{Dyn} can be represented as:

$$Pow_{Dyn} = K \cdot V_{dd}^2 \cdot F \tag{1}$$

where K is a circuit related constant. The leakage power of a core depends on the supply voltage (V) and current core temperature ($Temp$), which can be expressed as:

$$Pow_{Leak}(V_{dd}, Temp_{die}) = V_{dd} \cdot (c_1 \cdot Temp_{die}^2 \cdot e^{(\frac{c_2 \cdot V_{dd} + c_3}{Temp_{die}})} + c_4 \cdot e^{(c_5 \cdot V_{dd} + c_6)}) \tag{2}$$

where c_1 to c_6 are technology dependent parameters[1] and $Temp_{die}$ represents the die temperature [50]. We employ Kirchhoff's equation for the RC-circuit thermal model to track the rate of change in temperature [50]:

$$\frac{dTemp_{die}}{dt} = (Pow_{circuit} - \frac{Temp_{die} - Temp_{amb}}{R_{die-amb}})/C_{die} \tag{3}$$

where C_{die}, $R_{die-amb}$, and $Temp_{amb}$ are the thermal capacitance of the die, thermal resistance between the die and ambient, and the ambient temperature, respectively. $Pow_{circuit}$ is the total power (i.e., summation of dynamic and leakage) consumed by the die and can be written as:

$$Pow_{circuit} = Pow_{Dyn} + Pow_{Leak} \tag{4}$$

By using a prior measurement for ARM-cortex A8 processor, built-in 14nm FinFET technology nodes [59], we set C_{die} and $R_{die-amb}$, values of which are given in Table 1[2]. We set $Temp_{amb}$ as 40 °C. TEI in FinFET reduces circuit delay at the increased temperature, which can directly impact the core frequency (F) that can be written as follows:

$$F = d_0 \cdot V_{dd}^2 + d_1 \cdot V_{dd} \cdot Temp_{core} + d_2 \cdot Temp_{core} + d_3 \cdot V_{dd} + d_4 \tag{5}$$

where d_0 to d_4 are the constants, and values of which are decided empirically [21] and are given in Table 1. The maximum temperature limit for our die is set as 80 °C.

Table 1. **System-wide Constants (based on 14nm FinFET Technology node) and their Values** [21, 50]

Parameters	d_0	d_1	d_2	d_3	d_4	C_{die}	$R_{die-amb}$
Values	−4.27	0.0042	0.0052	10.6	−2.66	9.0 J/K	35.8 K/W

[1] which are internally set in McPAT-monolithic's power model [56], used in our simulation (Sec. 4)

[2] These values might be determined and updated for other technology nodes empirically detailed in prior works [20, 21]

2.3 System Model

The considered system comprises a periodic task set $\mathbb{T} = \{\mathbb{T}^1, \mathbb{T}^2, \ldots, \mathbb{T}^n\}$ comprising n tasks, that needs scheduling on a multicore platform $\mathbb{V}$ with m cores, i.e., $\mathbb{V} = \{\mathbb{V}^1, \mathbb{V}^2, \ldots, \mathbb{V}^m\}$. Each core is able to run on a normalized set of frequencies $\mathbb{F} = \{\mathbb{F}^1, \mathbb{F}^2, \ldots, \mathbb{F}^{mx}\}$, where $\mathbb{F}^{mx}$ refers to the highest (lowest) available normalized frequency of 1 (0) and other frequencies range between 0 and 1. Each occurrence of $\mathbb{T}^i$ is associated with a deadline/period d^i, an execution requirement e^i (while performing on $\mathbb{F}^{mx}$), utilization $u^i = \frac{e^i}{d^i}$, and steady state temperature $\mathbb{T}^i_{ss}$. The steady state temperature of a task on a core is defined as the core's achieved temperature when the task performs without interruption on the core at a specific frequency for a prolonged stretch of time, possibly over multiple instances. At any moment, the remaining deadline/period of a task $\mathbb{T}^i$ is represented as rd^i.

3 *TREAFET*: PROPOSED TECHNIQUE

The working mechanism of *TREAFET* can be represented in a hierarchical manner that comprises five algorithms. In the outer layer (Algorithm 1), the execution of tasks in the system is broken down into multiple chunks of time slots called *intervals*, and the algorithm keeps track of the progress of each task across the intervals. In the intermediate layer (Algorithm 2), a heuristic based temperature-aware schedule is prepared for the given tasks on the available cores. In the inner layer (Algorithm 3), a FinFET specific online technique is used by the scheduler to further manage the temperature and energy consumption of the individual cores in the system. To carry out the whole thermal management process, Algorithm 3 employs an energy-adaptive frequency determination strategy (Algorithm 4) and a slack exploitation technique (Algorithm 5) for energy saving. In the subsequent subsections, we describe the working of each layer in detail.

3.1 Overall Progress Tracking

The overall progress tracking mechanism is given in Algorithm 1. In this layer, the scheme keeps track of the overall progress for each task in the system using the technique of *deadline partitioning* [32] (line 2). The algorithm sorts the remaining deadlines of the given tasks in the list $\mathbb{T}$. Next, the algorithm determines the duration of the next interval (say $\mathbb{I}_k$), which is the time duration from the current time slot to the nearest deadline among all the tasks. It can be represented as:

$$\mathbb{I}_k = min\{rd^1, rd^2, \ldots, rd^n\} \tag{6}$$

*TREAFET*employs a number of temperature-aware heuristics at different layers. In the first layer (Algorithm 1), it applies a basic temperature aware strategy. The algorithm needs to sort the tasks based on their temperature characteristics. However, if the sorting is done based on steady state temperatures, it may not be a realistic scenario because the steady state temperature can only be reached if the task keeps on running on a core for a very long time. In our algorithm, the task execution is broken into several parts, and a task $\mathbb{T}^i$ can only execute for a duration er^i_k in $\mathbb{I}_k$ (explained next). So, a core may not reach the steady state temperature in the interval. Further, the actual task-to-core assignment happens in the next layer. Therefore, the algorithm uses the concept of *virtual core*, which is used for initialization purposes. Since the algorithm does not know on which core the task $\mathbb{T}^i$ will be assigned, it computes $Temp_V^{avg}$ by computing the average of temperatures of the available m cores (line 4) and assigns it to the virtual core. It assumes that each task will execute on this core separately with the starting temperature $Temp_V^{avg}$ and check the hotness of the core $Temp_{\mathbb{V}}^{\mathbb{T}^i}$ when they finish. Depending on this final temperature $Temp_{\mathbb{V}}^{\mathbb{T}^i}$, a sorted task list $\wedge_1$ will be prepared. This final temperature presents a more realistic scenario because it is based on the actual execution requirement in the interval. Further, in a system having a decent

workload, it is highly improbable that some cores are totally idle and have a significantly lower temperature than other cores of the system. Hence, the consideration of $Temp_V^{avg}$ as the initial temperature of the virtual core is a reasonable assumption.

With the onset of the interval $\mathbb{I}_k$, it considers each task $\mathbb{T}^i$ in the list $\mathbb{T}$ individually (line 5 to 8). For each task $\mathbb{T}^i$, the algorithm determines the execution requirement (er_k^i) of the task (line 6) for the ensuing interval $\mathbb{I}_k$ by using the following equation:

$$er_k^i = \lceil e^i \times \mid \mathbb{I}_k \mid / d^i \rceil \tag{7}$$

Algorithm 1: *TREAFET*: Determination of Intervals and Workload

Input: $\mathbb{T}$, $\mathbb{V}$
Output: A set of schedules for the interval set
1 Let, *i.* Set of intervals, $\mathbb{I} = \{\mathbb{I}_1, \mathbb{I}_2, \dots\}$, *ii.* $\wedge_1$ be sorted list (in non-increasing order) based on the temperature of a core with an initial temperature $Temp_V^{avg}$ if it runs $\mathbb{T}^i$, and *iii.* $ST_k[m][n]$ be the generated schedule table for $\mathbb{I}_k$
2 Find a set of intervals $\mathbb{I}$ using *Deadline Partitioning*
3 **for** ***each*** *interval* $\mathbb{I}_k \in \mathbb{I}$ **do**
4 Compute average core temperature, $Temp_V^{avg} = \frac{Temp_{V1} + Temp_{V2} + \cdots + Temp_{Vm}}{m}$
5 **for** $i \leftarrow 1 : |\mathbb{T}|$ **do**
6 Find share er_k^i using Equation 7
7 Find core temperature $Temp_{\mathbb{V}}^{\mathbb{T}^i}$ if it runs $\mathbb{T}^i$ using Equation 3
8 $\wedge_1 \leftarrow \wedge_1 \cup \{\langle i, er_k^i, Temp_{\mathbb{V}}^{\mathbb{T}^i} \rangle\}$
9 Call Algorithm 2

If the scheme can execute each task $\mathbb{T}^i$ for er_k^i time-slots in every interval, $\mathbb{I}_k$, then all tasks will definitely meet their deadlines. Further, such execution of tasks will guarantee that the tasks maintain a steady rate of progress at the boundary of the intervals. The algorithm maintains a non-increasing ordered sorted list of tasks $\wedge_1$ based on the temperature reached by the tasks if they execute on the virtual core from the beginning of the interval. To prepare the list $\wedge_1$, the algorithm determines the temperature of the virtual core if the task under consideration, i.e. $\mathbb{T}^i$, runs on the core for its required time-slots er_k^i in the interval using Equation 3 (line 7). Based on this value, the task $\mathbb{T}^i$ is added to the list $\wedge_1$ at an appropriate position (line 8). Once all the tasks have been added to the list $\wedge_1$, the algorithm moves to the next phase by calling Algorithm 2 (line 9).

3.2 Scheduling Phase

Algorithm 2, the heart of the scheduling phase, prepares the basic schedule for the tasks on the available cores. The algorithm uses a heuristic where it tries to schedule the hottest task on the coolest core and the coolest task on the hottest core in an alternate iteration. Such a heuristic has proved useful to reduce core temperatures in prior literature [27, 88]. It iterates over the task list $\wedge_1$ by considering one appropriate task-core pair at a time (line 4 to 20). To alternatively extract hot and cool tasks from the list $\wedge_1$ and assign them to an appropriate core, it uses a binary variable $flag$. If the $flag$ is currently set (line 5), the hottest task (say $\mathbb{T}^i$) is chosen from the front of the list $\wedge_1$ (line 6), and the core (say $\mathbb{V}^j$) having the lowest temperature is chosen, which can meet the requirement of the chosen task (line 7). If no such core is found for the chosen task $\mathbb{T}^i$, it is added to the front of the list of migrating task $\wedge_{mgr}$ (line 8 to 10). Alternatively, suppose the current value of the $flag$ is 0. In that case, the task having the lowest steady-state temperature is chosen from the back of the list $\wedge_1$ (line 12) and is scheduled on the core having the highest current temperature (line 13). Subsequently, the core capacity $\mathbb{V}_c^j$ is updated (line 18) and the temperature at which the core $\mathbb{V}^j$ will reach if it executes the task $\mathbb{T}^i$ is computed using Equation 3 (line 19). It may be noted that each core $\mathbb{V}^j$ can execute for a maximum $|\mathbb{I}_k|$ time-slots in an interval $\mathbb{I}_k$ at the maximum

available normalized operating frequency $\mathbb{F}^{mx}$ (which is equal to 1). Hence, at the start of every interval, each core capacity $\mathbb{V}_c^j$ is set to the value $|\mathbb{I}_k|$ (line 3). Next, the $flag$ is set to the alternate value (line 20). Using such a strategy helps *TREAFET* to prepare a basic temperature-aware schedule which keeps the temperature of the cores balanced. At the end of Algorithm 2, all the tasks present in the list $\wedge_{mgr}$ are scheduled on available cores using *Next Fit Bin Packing* strategy (line 21). Tasks that could not be scheduled on any single core are executed among the available set of cores without considering the thermal status of the cores. When all the tasks have been scheduled on available cores, Algorithm 2 computes the operating frequency for each core $\mathbb{V}^j$ in the interval $\mathbb{I}_k$ where each core can execute for a maximum $|\mathbb{I}_k|$ time-slots. The assigned workload on $\mathbb{V}^j$ for the interval can be determined as $\sum_{i=1}^{n} er_k^i, \forall \mathbb{T}^i$ having $ST_k[j][i] > 0$. Therefore, the required operating frequency $\mathbb{F}_{opt}^j$ for the core $\mathbb{V}^j$ can be computed as a ratio of assigned workload to available time slots (line 22):

$$\mathbb{F}_{opt}^j = \lceil \frac{\sum_{i=1}^{n} er_k^i}{|\mathbb{I}_k|} \rceil, \qquad \forall \mathbb{T}^i \in \mathbb{T} \mid ST_k[j][i] > 0 \tag{8}$$

Since the computed frequency may not always be available in the discrete frequency set $\mathbb{F}$, we used the ceiling notation to denote the next available frequency, which is greater than or equal to this ratio. Further, it may be noted that the scaling down of the operating frequency will lead to the execution of tasks at a slower speed; however, the assigned workload to a core can never exceed its capacity in an interval, which means that the value of $\mathbb{F}_{opt}^j$ will always be less than or equal to 1. Therefore, running each core $\mathbb{V}^j$ at $\mathbb{F}_{opt}^j$ ensures that the workload finishes by the interval boundary of $\mathbb{I}_k$. For the dynamic thermal management (explained in Sec. 3.3), the algorithm requires separate operating frequencies for each task. Therefore, we assigned $\mathbb{F}_{opt}^j$ as the base frequency $F\_Base[i]$ for each task $\mathbb{T}^i$ which has been assigned to the core $\mathbb{V}^j$. Algorithm 3 is called next to manage the temperature during execution (line 23).

3.3 Runtime Thermal Management

Once the tasks are scheduled, the individual tasks are executed at the assigned processor cores. The variation in the counts of different types of instructions over the execution phases of a task changes the core temperature over time. As per Equation 5, the core frequency at any time-stamp of a FinFET based processor core depends upon the supply voltage and the core temperature due to TEI. Although at higher temperatures, the cores run faster, which can be leveraged to enhance the performance, a safe temperature needs to be maintained, so that circuit failure caused by SHE can be prevented beforehand. Hence, our runtime thermal management first considers the live core temperature periodically and tries to prudentially exploit the TEI to speed up the execution, whereas thermal safety is guaranteed by fine-grained dynamic voltage scaling (FG-DVS).

Runtime thermal management of *TREAFET* can be presented as an integration of the following three modules:

- Dynamic Exploitation of TEI
- FG-DVFS during Memory Stalls
- Shutdown the core during Slacks

The first one attempts to exploit TEI by considering the current-voltage magnitude and temperature. The supply voltage is governed to ensure a core frequency equal to the assigned value for the current task. Moreover, the power consumption must not violate the underlying core's thermal design power (TDP) to ensure thermal safety. As this voltage management is performed periodically, we must select a moderate period length during which thermal status can be assumed unchanged [21]. Moreover, the period length should be sufficiently large so that voltage switching should not

Algorithm 2: Task Scheduling

Input: $\wedge_1$, $\mathbb{V}$, ST_k
Output: Schedule for current interval

1 Let *i.* $\wedge_{mgr}$ be the task list having tasks which need migration in the current interval, *ii.* $\mathbb{V}_c^j$ be the spare capacity of $\mathbb{V}^j$ in $\mathbb{I}_k$, and *iii.* $flag \in \{0, 1\}$
2 Set $flag \leftarrow 1$ and sort cores in non-increasing order of their temperature
3 Set $\mathbb{V}_c^j = |\mathbb{I}_k|$ for $j = 1, 2, \ldots, m$
4 **while** $\wedge_1 \neq empty$ **do**
5 **if** $flag = 1$ **then**
6 Fetch $\mathbb{T}^i$ from the front of $\wedge_1$
7 Choose the core (say $\mathbb{V}^j$) having the lowest temperature and $\mathbb{V}_c^j \geq er_k^i$
8 **if** *No such core found* **then**
9 Add $\mathbb{T}^i$ to front of $\wedge_{mgr}$
10 continue;
11 **else**
12 Fetch $\mathbb{T}^i$ from the end of $\wedge_1$
13 Choose the core (say $\mathbb{V}^j$) having the highest temperature and $\mathbb{V}_c^j \geq er_k^i$
14 **if** *No such core found* **then**
15 Add $\mathbb{T}^i$ to end of $\wedge_{mgr}$
16 continue;
17 Schedule $\mathbb{T}^i$ on $\mathbb{V}^j$ for er_k^i time-slots by updating $ST_k[j][i]$
18 $\mathbb{V}_c^j \leftarrow \mathbb{V}_c^j - er_k^i$
19 Compute $Temp_{\mathbb{V}^j}$ using Equation 3 and update position of $\mathbb{V}^j$ in core list
20 $flag = (flag + 1)\%2$
21 Schedule all tasks of $\wedge_{mgr}$ on cores using *Next Fit* bin packing with respect to shares of tasks and remaining capacities of cores by updating ST_k
22 Compute operating frequency $\mathbb{F}_{opt}^j$, for each core $\mathbb{V}^j$ using Equation 8
23 Call Algorithm 3

frequently occur, which might incur significant power and performance issues at the voltage regulators (VRs) along with the transient faults. The second approach considers individual memory stalls at the cores during a period and reduces the core's V/F to the lowest possible value to save energy. The V/F will be scaled up just before the data arrives from the memory. The saved energy is next traded off by scaling the voltage to a larger value to maintain a higher core frequency for a stipulated time. The time span will be determined by accounting for the energy saved by DVFS during the memory stall. Prudential exploitation of TEI and energy-adaptive FG-DVFS during memory stalls result in improved performance, thus generating slacks, during which the core will be power gated to reduce energy usage and temperature, which is our last strategy.

3.3.1 Dynamic Exploitation of TEI. The entire process of runtime TEI exploitation and frequency management for the underlying FinFET based cores is given in Algorithm 3. The execution span of a task $\mathbb{T}^i$ on a core is broken into $\lceil er_k^i/\Delta \rceil$ parts, each of which is called *frame*. We consider Δ as the maximum allowed frame size, which is given as an input to the algorithm. To periodically monitor the core-temperature, Algorithm 3 first considers the length of a frame ($min\{\Delta, re_k^i\}$) and initial core temperature ($Temp\_Init$), where re_k^i denotes the remaining execution share of $\mathbb{T}^i$ in $\mathbb{I}_k$. Note that $Temp\_Init$ is basically the current core temperature (can be determined through on-chip thermal sensors) just before the task execution. The higher and lower temperature thresholds are also inputs to the algorithm ($Temp_{thr}^{Hi}$ and $Temp_{thr}^{Low}$) along with the operational voltage levels for the cores and the derived base frequencies for the individual tasks. Once initialization of the required parameters is done, the scheduled tasks are fetched for execution. Next, the supply voltage (V_{in}) is set to a certain level so that the assigned base frequency is guaranteed, i.e. $Freq(V_{in}, Temp) \geq F\_Base[i]$ (line 3), and the task execution will be started.

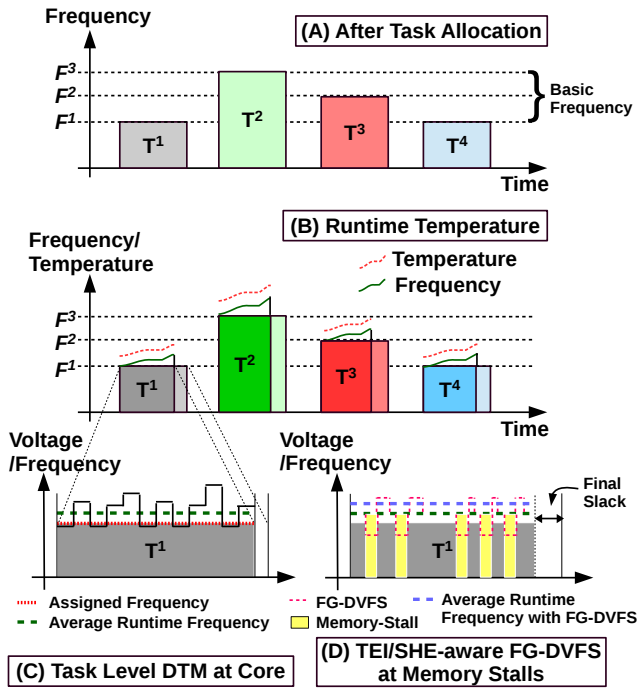


Fig. 2. Online TEI/SHE-Cognizant DTM

During execution, our algorithm collects the core temperature at the end of each frame, where the frame length is determined by $min\{\Delta, re_k^i\}$ (line 7). Once the core temperature at the end of the last frame is higher than $Temp_{thr}^{Hi}$, the V_{in} is set at the lowest possible level, $V_{dd}[1]$ (line 8 to 9). Lowering supply voltage will reduce the core power consumption, but higher temperatures will be able to maintain a suitable frequency, thanks to TEI. Note that we need to set the lowest voltage magnitude at some optimal value, which at the maximum allowed temperature, can maintain a certain frequency by exploiting TEI so that the deadline can be met. In RESTORE [72], we have shown how our considered lowest voltage value can still maintain a sufficiently high frequency when the temperature is higher or the same as $Temp_{thr}^{Hi}$. Once the temperature is lower than $Temp_{thr}^{Low}$, V_{in} will be set to the highest possible value $V_{dd}[L]$ (line 10 to 11), where L implies the number of available voltage levels.

If the current temperature of the core is within the predefined thresholds, the voltage level is set to a minimum possible magnitude which can maintain an average core frequency at least $F\_Base[i]$ (line 12 to 17). After setting V_{in}, the core frequency is updated (line 18). During a frame, the core executes a task normally, and the number of completed clock cycles is tracked by employing a counter, $cycle\_cntr$ (line 22). Figure 2[C] illustrates the idea of our dynamic TEI exploitation technique at the task level granularity with an example where three (base) frequencies (f_1, f_2 and f_3) are considered with four tasks (T^1 to T^4). The figure magnifies how our technique monitors the temperature and exploits it periodically to apply DVFS while guaranteeing the task deadlines. Once the frequency is set for a period, our algorithm also checks for the costly memory stalls at the individual cores, during which FG-DVFS can be applied in an energy-adaptive manner so that TDP will be maintained (line 23). The details of energy-adaptive FG-DVFS will be discussed next.

Algorithm 3: *TREAFET*: Dynamic Exploitation of TEI

Input: $F\_Base[1:|\mathbb{T}|]$, $Temp\_Init$, Δ, $dispatch\_table$, $V_{dd}[1:L]$, $Temp_{thr}^{Hi}$, $Temp_{thr}^{Low}$, re_k^i
Output: Thermal Safety with maintained base frequency

```
1  Temp = Temp_Init
2  for each task 𝕋^i do
3      # Fetch task 𝕋^i along with its assigned base frequency F_Base[i], current temperature (Temp) of the core, and set re_k^i = er_k^i
4      # Set V_in to fix the frequency, so that, Freq(V_in, Temp) ≥ F_Base[i], and start execution and cycle_cntr = 0
5      while 𝕋^i is executed do
6          if cycle_cntr == min{Δ, re_k^i} then
7              # Get the frequency and temperature for 𝕍^j in the last frame
8              if Temp ≥ Temp_thr^Hi then
9                  V_in = V_dd[1]
10             if Temp ≤ Temp_thr^Low then
11                 V_in = V_dd[L]
12             if Temp_thr^Hi > Temp > Temp_thr^Low then
13                 for p = 2 to (L − 1) do
14                     F_Next = getFreq(V[p], Temp)
15                     if (F_Curr + F_Next)/2 ≥ F_Base[i] then
16                         V_in = V_dd[p]
17                         break
18             F_Curr = getFreq(V_in, Temp)
19             cycle_cntr = 0
20             re_k^i = re_k^i - min{Δ, re_k^i}
21         else
22             cycle_cntr++
23             cycle_ctr = EnergyAdaptive(F_Curr, V_in, Temp) (call Algorithm 4) ;
24     Call Slack_Exploitation algorithm (Algorithm 5);
```

3.3.2 FG-DVFS during Memory Stalls.

Analyzing Memory Stalls. Prior literature claims that a significant portion of the entire execution time of modern applications is spent on accessing the off-chip memory (both data and instruction blocks) [4, 16, 73], and individual memory accesses are costly in terms of access-time and energy. In case of an instruction miss in an OoO core, the instruction dispatch is stalled once the front end is depleted by the instructions. On the other hand, upon a load miss (i.e. a miss for a data block), due to memory-level parallelism (MLP), only the very first miss, or an isolated miss, of a set of potential in-flight memory accesses to the same memory location (i.e. cache block), will observe the entire duration of the memory access. Towards employing FG-DVFS at the core, we need to detect which loads cause isolated misses. These isolated load misses can be identified by looking at the miss status holding register (MSHR) of the respective cores that have requested the data. Once a load (LLC-) miss does not have an allocated MSHR, and its respective requester core has zero pending memory accesses at present, then this miss can be tagged as an isolated miss. We executed eight PARSEC applications in gem5 [17], a cycle-accurate full system simulator, for $100M$ cycles (in Region of Interest (RoI)) on a single core OoO $x86$ processor, equipped with two levels of caches (64KB 4W L1 (D/I) and 1MB L2) and observed the percentage of entire execution time the stalls take place while accessing memory. We segregated the isolated and instruction misses for each benchmark application and showed the results in Figure 3. The result portrays that two memory-intensive applications, *Ded* and *Stream*, spend up to 60% of their whole execution time in accessing off-chip memory. Overall, for all applications, on an average of 25% of the total execution time, a core remains stalled for accessing memory. This is significantly high and hence can be utilized to reduce the power and temperature of the core without any performance impact.

Energy-Adaptive FG-DVFS vs. Individual Stalls. Now, we will discuss two different scenarios during which FG-DVFS can be applied at the requester core during the memory stalls: the first one is an instruction miss, and the second one is an isolated miss. Let us assume that each of our

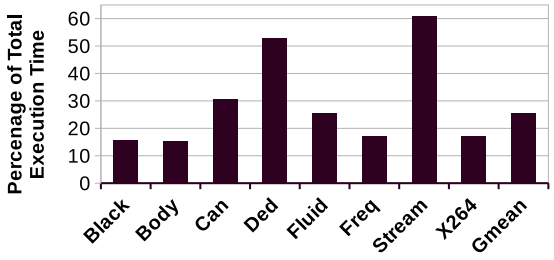


Fig. 3. Percentage of Total Execution-time Spent for Accessing Off-chip Memory

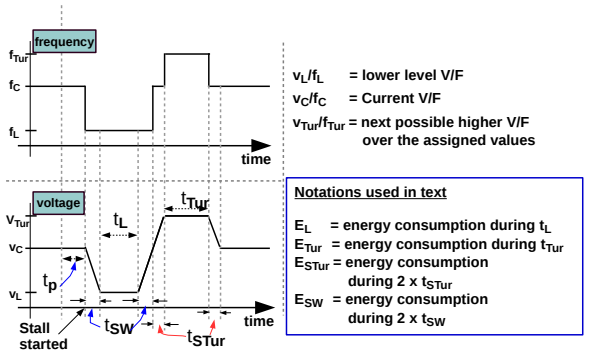


Fig. 4. FG-DVFS Time-line: During Memory Stall

cores dispatches D instructions per clock cycle. Once an LLC miss is detected due to the absence of an instruction block, it requires L cycles before the dispatch stops, where L implies the front-end pipeline depth of the core. Meanwhile, access to the off-chip memory is being performed, and on completion, instructions will be fetched. However, the instruction dispatching will be resumed only L cycles later. Therefore, for an instruction miss, the stall span for applying FG-DVFS is equal to the off-chip memory access latency. Our energy-adaptive mechanism scales down the V/F once an instruction LLC-miss is detected and scales the V/F up on completion of the memory access. The V/F scaling effectively changes the core frequency abruptly when applying FG-DVFS, which consequently elongates the depletion of the front-end pipeline. However, such temporal overhead will not incur any performance impact as it will be hidden by the off-chip memory access.

An isolated (load) miss can be identified by accessing the MSHR of the respective requester core. However, scaling down V/F on detection of an isolated miss can lead to performance aggravation. Basically, the dispatch is stalled during a load miss if one of the following situations occurs: *(A)* the Re-Order Buffer (ROB) is filled up, *(B)* the CPU-registers get exhausted, or *(C)* the issue queue is filled up with all instructions that have a dependency on the currently missed block. As our considered system handles time-critical applications, we decided to wait until the dispatch stalls. On detection of a dispatch stall, the V/F is scaled down, and the dispatch resumes after scaling the V/F up once the data arrives from the memory. In fact, waiting till dispatch stalls during an LLC miss before scaling down V/F can also handle the overlapped misses of the independent blocks. Waiting for dispatch stalls might compromise the benefits of FG-DVFS but can safeguard real-time applications from deadline violation.

The stall spans for individual memory accesses depend upon several timing overheads, which can result in variation in the stall span for individual LLC misses. To keep our problem simple and straightforward, in *TREAFET*, we assume that the time span for accessing off-chip memory is

uniform. This might not be a realistic choice, but studying the variation of the memory access time in the spectrum of real-time systems is another research avenue [61] and is out of the scope of this paper. In *TREAFET*, we consider a fixed DRAM access latency of 70ns [25]. However, our FG-DVFS mechanism can also be extended to tackle variation in memory delay without incorporating significant changes[3].

Figure 4 elaborates the (individual) timing diagram while FG-DVFS is applied. Conventional systems usually experience a time gap between an LLC-miss detection and dispatch stalls at the requester core, which is defined here as t_p in this figure. Then, the FG-DVFS controller reduces the frequency without any delay to a lower level and will concurrently start reducing the voltage (from v_C to v_L). The controller will start scaling up the voltage for a stipulated time span before the stall completes so that upcoming instructions can be executed at the assigned V/F after the stall is completed. Once the voltage is scaled up, the frequency is set to the respective higher level, which incurs no delay.

In our work, we considered the OoO x86 core as having the highest voltage setting (Turbo voltage) of 0.85V and a lower voltage setting of 0.65V, whereas our operational voltage is considered as 0.75V. Our considered on-chip VR has a switching speed of 20mV/ns, and each VR consumes 0.09W and 0.12W, for 0.65V and 0.85V outputs, respectively [31]. For analyzing energy savings, we considered that our core temperature remains stable at 77 °C, for which frequency values are obtained by employing Equation 5 for 0.85V, 0.8V, 0.75V, 0.7V and 0.65V, which are 4.1GHz, 3.75GHz, 3.5GHz, 3.3GHz, and 3.0GHz, respectively. During a stall span of 70ns, the core (OoO $X86$) can consume 174nJ of energy at 0.75V, which is our baseline. By applying voltage downscaling to 0.65V from 0.75V and subsequent upscaling to 0.75V, we can save a significant amount of energy which is up to 20% on an average, while including the power consumed by the VR during switching [4]. From a timing perspective, each of the scaling up and scaling down processes of voltage can take up to $5ns$ if it has to switch between 0.65V and 0.75V while considering VR's voltage switching speed as 20 mV/ns [31]. We also considered the configuration of a core (detailed in Sec. 4) and derived the longest time taken before the dispatch stalls from the detection of an LLC-miss to be around 8ns (i.e. t_p in Figure 4). This implies the core can be operated at the lowest voltage level for $52ns$ in the worst case, whereas the entire duration of memory stall is considered as 70ns [25]. This indicates the core will have sufficient time-spans both for switching operations as well as maintaining the lowest possible voltage level.

We further analyzed the overall reduction in the dynamic energy of the core gained by FG-DVFS. In order to do so, we executed eight PARSEC applications for $100M$ cycles (in RoI) in gem5 [17] cycle accurate full system simulator. Subsequently, we derived the magnitudes of the energy usages for individual applications by simulating the architecture and by feeding the performance traces (collected from gem5's output) in McPAT simulator [56] along with due consideration to Equation 5. The thermal simulation has been performed by using HotSpot 6.0 [85] simulator for modeling TEI. The entire closed-loop simulation setup used in *TREAFET* is detailed in Sec. 4. The energy savings for individual benchmark applications by employing FG-DVFS during memory stalls is depicted in Figure 5 that shows a significant reduction of 17% (up to 31%) in the core's dynamic energy. As

[3]Note that the following parameters might be helpful to gain insights regarding variation in the value of $stall\_span$-memory access latency, memory bus bandwidth, outstanding memory requests, data transfer size and memory-level parallelism (MLP). As memory access patterns vary over different execution phases, an in-depth phase-based analysis for these parameters will be helpful in the determination of $stall\_span$, which can be done in a separate sub-routine and upon detection of memory stall at line 15 in Algorithm 4, this sub-routine will be called before applying FG-DVFS. However, detailed analysis regarding this concept is out of the scope of *TREAFET*.

[4]We calculated energy usage at the nano-second precision level by discretizing. The energy consumed by each VR during the switching processes between different levels of voltage is around 2nJ.

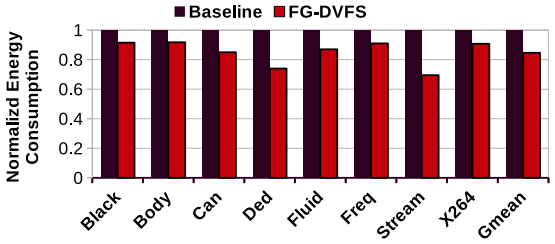


Fig. 5. Energy savings by employing FG-DVFS during Memory Stall

in the case of the core, dynamic energy shares a significant amount of the total energy usage [48]; this noticeable saving in dynamic energy potentially motivates us to employ FG-DVFS towards improving the energy efficiency of the cores. We derived all of these values by simulating the architecture in our simulation setup (detailed in Sec. 4), along with due consideration to Equation 5 and McPAT simulator [56].

Trading off Energy Saving by FG-DVFS. Now, we will elaborate on the entire idea of exploitation of the energy gains of running the core at lower voltage during memory stalls by considering Figure 4. Basically, Figure 4 depicts the timing diagram during an individual LLC-miss and the frequency switching up on resolving the miss. As a stable temperature is assumed (77 °C), TEI's impact on the frequency due to temperature will also remain the same. However, the impact of voltage scaling is considered. Our baseline V/F setting is assumed as v_C/f_C, which has been assigned by our constrained scheduling. During a stall incurred due to an LLC miss, the V/F setting of the core will be stepped down to v_L/f_L. While increasing the V/F on completion of a stall interval, the V/F will be set to v_{Tur}/f_{Tur} (higher than v_C/f_C). The respective energy consumption while staying at C, L and Tur levels are E_C, E_L and E_{Tur} (see Figure 4). The switching processes consume E_{SW} and E_{STur} during switching between C and L levels and C and Tur levels, respectively (see Figure 4). As violating power budgets might entail severe thermal issues, the energy saving by FG-DVFS over the baseline (i.e., $E_{sav} = E_C - (2 \times E_{SW} + E_L)$) during $2 \times t_{sw} + t_L$ should be the highest limit while running the core at the turbo (Tur) mode. Now, towards maintaining the power budget or on-chip thermal safety, $E_{sav} \geq E_{Tur} + 2 \times E_{STur}$. As the power consumption at v_{Tur}/f_{Tur} level is known beforehand, the time span (t_{Tur}) can be determined dynamically. Note that t_L and t_{Tur} denote the time spans the core can be operated at L and Tur voltage levels, respectively. The overall performance improvement and slack generation, in addition to its exploitation towards improving energy efficiency, will be discussed in Sec. 5.

Energy-Adaptive FG-DVFS Algorithm (Algorithm 4). Our energy-adaptive mechanism for employing FG-DVFS is written in Algorithm 4. During a frame ($min\{\Delta, er_k^i\}$), Algorithm 3 first determines a particular voltage level that needs to be maintained to meet the base frequency requirement. As our application is executed with a strict time and power/thermal constraint, our proposed energy-adaptive algorithm exploits the costly memory stalls to reduce energy and exploits the saved energy for improving performance. Towards that, Algorithm 4 is employed per core at the task level granularity and is being called by Algorithm 3 while executing a task. Algorithm 4 considers current frequency (F_{Curr} as f_C), current voltage (V_{in} as v_C), and temperature ($Temp$) as inputs which are passed by Algorithm 3 (line 23 in Algorithm 3). The VR's switching speed (VR_Speed), span for memory stall ($Stall\_Span$), v_L and v_{Tur} are also considered as inputs to Algorithm 4. To maintain the functional correctness, Algorithm 4 maintains the following three flags: $dvfs\_enabled$, $scaled\_down$ and $scaled\_up$.

Algorithm 4: EnergyAdaptive(F_{Curr}, V_{in}, $Temp$)

```
   Input: VR_Speed, Stall_Span, v_L, v_Tur
   Output: g_cycles
1  scaled_down = 0, scaled_up = 0
2  v_C = V_in, f_C = F_Curr
3  if (dvfs_enabled = 0) and (LLC-miss is detected) then
4      if (dispatch stalls on a data miss) or (an instruction miss is detected) then
5          f_L = getFreq(v_L, Temp)
6          t_SW (in cycles) = ((v_C − v_L) / VR_Speed) × f_L
7          t_L (in cycles) = (Stall_Span − 2 × ((v_C − v_L) / VR_Speed)) × f_L
8          t_L^A (actual cycle counts during t_L at v_C) = Stall_Span × f_C
9          # Set frequency to f_L, and start reducing voltage to set at v_L and stop increasing g_cycles
10         scale_down = 1
11         dvfs_enabled = 1
12         # Set counter to the duration of the reduced V/F setting (i.e. t_L)
13         counter = t_L
14     else
15         # Block is already being handled by an earlier request, so execute as normal
16 if (counter == 0) and (scale_down == 1) then
17     f_Tur = getFreq(v_Tur, Temp)
18     # Start increasing voltage to v_C
19     if v_Tur > current_voltage ≥ v_C then
20         # set current frequency at f_C
21     start increasing g_cycles (i.e. g_cycles + +)
22     t_STur (in cycles) = ((v_Tur − v_C) / VR_Speed) × f_C
23     t_Tur (in cycles) = ((2×E_SW + E_L − 2×E_STur)) / CorePow_Tur) × f_Tur
24     # Start increasing voltage to v_Tur
25     if current_voltage == v_Tur then
26         # set current frequency at f_Tur
27     scaled_down = 0
28     scaled_up = 1
29     # Set counter to the duration of the increased V/F setting (i.e. t_Tur)
30     counter = t_Tur
31 if (counter == 0) and (scaled_up == 1) then
32     dvfs_enabled = 0
33     scaled_up = 0
34     # Set frequency to f_C, and start reducing voltage to v_C
35     # Since DVFS (scaled down and up) is applied for some certain lengths,
36     # a fixed number of cycles can be added to g_cycles that accounts for the frequency difference
37     g_cycles + = t_L^A
38 # counter is an unsigned saturating decrementer
39 counter--
40 return g_cycles
```

Once an LLC-miss (data or instruction miss) is detected, and DVFS is not enabled, Algorithm 4 starts preparing for applying DVFS, and eventually DVFS will be applied (line 3 to 15). Before lowering the voltage to v_L, f_L is calculated through the $getFreq()$ function that also considers temperature. Subsequently, the number of cycles during t_{SW} and t_L are also computed so that the total number of cycles can be tracked during the process. As frequency can be changed abruptly, while the voltage switching takes time, the frequency is set at the lower level at the beginning of the voltage scaling down process. During the voltage scaling-up process, frequency is changed upon completion of the voltage scaling. However, as reducing frequency to f_L will finish a lesser number of cycles than in f_C, to maintain the correct cycle count of the task, t_L^A is also calculated that keeps track of the number of cycles which can be completed if frequency would be at f_C. Next, the frequency is set at f_L, and the voltage switching process is initiated by setting $scale\_down$ and $dvfs\_enabled$ flags. The cycle counter, g_cycles, is also stopped increasing (line 8 to 11). To keep track of the duration (t_L), an unsigned saturating decrementer ($counter$) is employed, which is set

at t_L at this point (line 12). However, if the dispatch does not stall during an LLC miss, the execution will be continued normally.

Once the counter reaches 0, and the *scale_down* flag is set, the algorithm starts the voltage scaling up process to set the frequency at the Tur level in an energy-adaptive manner (line 16 to 30). At first, the f_{Tur} is computed, and the voltage scaling-up process is initiated. Once the voltage reaches v_C, the frequency will be set at f_C, and subsequently, g_cycles will start increasing (line 17 to 21). Our algorithm next computes the voltage switching time to turbo mode (t_{STur}) and derives t_{Tur} during which scaled voltage will be maintained (line 22 to 23). Towards that, respective energy usages are also derived so that overall TDP is maintained. However, upon deriving t_{Tur}, scaling up of voltage to v_{Tur} will be initiated, and once it will reach v_{Tur}, the frequency will be set at f_{Tur} and the $counter$ is initialized to t_{Tur} (line 24 to 30). Once the $counter$ reaches 0, the frequency is set at f_C, and voltage is scaled down to v_C. Finally, to maintain the correctness in the cycle counting process, g_cycles is updated with t_L^A and returned (line 31 to 40).

Algorithm 5: Slack_Exploitation

```
   Input: Break_Even_Time
1  Slack_after_T_i = (Cyc_Ext_End_T_i − Curr_Time) ;
2  if Slack_after_T_i > Break_Even_Time then
3  |   # Turn off the core ;
4  |   while Slack_after_T_i > Break_Even_Time do
5  |   |   Slack_after_T_i--;
6  |   #Turn on the core ;
```

3.3.3 Slack Detection and Exploitation. Upon completion of the execution of T_i, the slack interval is checked, as TEI and our energy-adaptive FG-DVFS can potentially increase the effective frequency of the tasks assigned during scheduling. To determine the slack interval, we introduce another parameter, called extended end time of T_i ($Cyc\_Ext\_End\_T_i$). $Cyc\_Ext\_End\_T_i$ of a task is the start time-stamp of the next task at the same processor or the deadline of the frame if T_i is the last task of the current frame. After completion of T_i, Algorithm 5 is called to detect and exploit the slack spans for improved energy efficiency. At first the slack-span ($Slack\_after\_T_i$) is derived (line 1) and if $Slack\_after\_T_i$ is higher than the $Break\_Even\_Time$ of the processor core (line 2), the processor core will be power gated (line 3). The core will be further turned on once the slack-span ($Slack\_after\_T_i$) is over (line 4 to 6).

3.3.4 Implementation. Our scheduling mechanisms (Algorithm 1 and 2) can be executed on some accelerators associated with main CPU cores, which is common in practice now-a-days [1, 2]. Towards implementing Algorithm 3 and 4, thermal sensors are needed to track the current temperature. Although the temperature sensing mechanism is different in the case of FinFET than the conventional MOSFET, there exists a plethora of options for the FinFET based cores [58]. On the other hand, for voltage scaling, conventional VR can be integrated at each core, which is common in practice for modern CMPs [18]. Integration of on-chip VRs might incur their own power and thermal overheads, which can be mitigated by employing techniques like ThermoGater [46]. Additionally, *TREAFET* can potentially incur frequent changes in voltage, which might lead to a transient fault in the core-circuitry, that can, however, be mitigated by incorporating some prior techniques [74]. The functional logic for dynamic exploitation TEI and FG-DVFS (i.e. Algorithm 3 and 4) can be implemented at the respective core's controller, which will orchestrate the associated VR to scale V/F setting of the core on LLC-miss induced stalls. Additionally, Algorithm 5 can also be

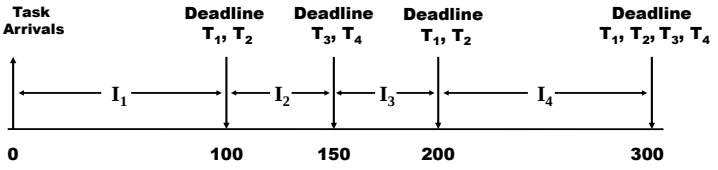


Fig. 6. Example: Deadline Partitioning

implemented at each core's controller to exploit the per-core-power-gating technique [23]. Overall, the implementation of *TREAFET* will not include any additional hardware support.

3.4 Analysis of the algorithms

In this section, we explain the working principle of the deadline partitioning technique and how it helps in meeting the task deadlines with the help of an example. The optimality of the technique for multiprocessor platforms has been proven earlier [32]. Further, we will analyze the time complexity of the proposed algorithms.

3.4.1 Deadline Partitioning Technique. To ensure that tasks are not violating deadlines, let us consider the following example. Consider a task set $\mathbb{T} = \{\mathbb{T}^1, \mathbb{T}^2, \mathbb{T}^3, \mathbb{T}^4\}$ having four tasks with execution parameters shown (execution requirements (e^i) and deadline (d^i)) in Table 2.

Table 2. Example: Task Parameters

Task	$\mathbb{T}^1$	$\mathbb{T}^2$	$\mathbb{T}^3$	$\mathbb{T}^4$
e^i	20	40	30	60
d^i	100	100	150	150

Initially, the remaining deadlines of the tasks are as follows: $rd^1 = 100$, $rd^2 = 100$, $rd^3 = 150$, and $rd^4 = 150$. Hence, the first interval $\mathbb{I}_1$ can be computed from Equation 6 as: $\mathbb{I}_1 = min\{rd^1, rd^2, rd^3, rd^4\} = 100$. The execution requirement of tasks for $\mathbb{I}_1$ can be computed from Equation 7 as: $er_1^1 = 20$, $er_1^2 = 40$, $er_1^3 = 20$, and $er_1^4 = 40$. Within the interval $\mathbb{I}_1$, tasks will be assigned and scheduled on cores using Algorithm 2. At the end of the interval $\mathbb{I}_1$, second instances of $\mathbb{T}^1$ and $\mathbb{T}^2$ will start their execution as tasks are periodic in nature. Hence, the updated remaining deadlines of the tasks are updated as $rd^1 = 100$, $rd^2 = 100$, $rd^3 = 50$, and $rd^4 = 50$.

The length of the second interval will be $\mathbb{I}_2 = min\{rd^1, rd^2, rd^3, rd^4\} = 50$. The execution requirements of the tasks for the second interval will be $er_2^1 = 10$, $er_2^2 = 20$, $er_2^3 = 10$, and $er_2^4 = 20$. Again, the tasks will be scheduled using Algorithm 2. At the end of the interval $\mathbb{I}_2$, second instances of $\mathbb{T}^3$ and $\mathbb{T}^4$ will start their execution. The remaining task deadlines will again be updated. The execution requirements of the tasks for four intervals are shown in Table 3, and the first four intervals corresponding to task deadlines are shown in Figure 6.

Table 3. Example: Task execution requirements

Task	er_1^i	er_2^i	er_3^i	er_4^i	**Task**	er_1^i	er_2^i	er_3^i	er_4^i
$\mathbb{T}^1$	20	10	10	20	$\mathbb{T}^3$	20	10	10	20
$\mathbb{T}^2$	40	20	20	40	$\mathbb{T}^4$	40	20	20	40

As we can observe, the execution of the first instance of $\mathbb{T}^3$ is broken into two intervals $\mathbb{I}_1$ and $\mathbb{I}_2$. The sum of execution requirements for $\mathbb{T}^3$ across these intervals can be computed as $er_1^3 + er_2^3 = 20 + 10 = 30$, which is equal to the original requirement $e^3 = 30$. This phenomenon holds true for all task instances. Therefore, we may conclude that the intervals will act as pseudo-deadlines for the tasks, and if the tasks complete their execution requirements for each interval, they will not miss their execution deadlines. One other advantage of the deadline partitioning technique is that

tasks can have their deadlines only at some interval boundaries. So, at the start of every interval, the proposed scheduler can prepare a schedule till the boundary of that interval without worrying about individual task deadlines between the interval boundaries.

Once the tasks are scheduled, they are deployed for execution with a particular frequency. This frequency is called *base frequency* (Equation 8), and Algorithm 3 ensures that the operating core frequency will never be lesser than this base value for individual tasks. Thus, the task deadlines are not violated.

3.4.2 Time-Complexity Analysis. In this section, we analyze the time-complexity of the proposed work *TREAFET* algorithm-wise for an interval $\mathbb{I}_k$ using a bottom-up approach starting from Algorithm 5 to 1.

- Algorithm 5 computes the generated slack after completion of the execution requirement for each task in an interval. If the generated slack is greater than the break-even time, the corresponding core is turned off for the slack duration. All these computation steps in Algorithm 5 take a constant time. Hence, the time complexity of the algorithm is $\mathbb{O}(1)$.
- Algorithm 4 applies energy adaptive techniques during memory stalls. There are no iterative operations in this algorithm. Therefore, each operation in this algorithm may be associated with a constant amount of time which brings the time complexity of the algorithm to be $\mathbb{O}(1)$.
- Algorithm 3 deals with dynamic exploitation of TEI during execution for every task at each core in an interval $\mathbb{I}_k$. It keeps track of task execution with the help of the *cycle_ctr* variable. At each frame boundary, it performs a set of operations of constant time depending upon the core temperature at the frame boundary. The summation of the *cycle_ctr* for all tasks at a core will be equal to the interval length $|\mathbb{I}_k|$. It may be noted that the summation of the *cycle_ctr* may be less if a task finishes before its computed requirement because of the TEI, but we are considering the worst case. As there are m available cores in the system, the time complexity of Algorithm 3 in an interval is $\mathbb{O}(m \times |\mathbb{I}_k|)$.
- Algorithm 2 assigns task onto cores and prepares the actual schedule. It considers each task sequentially for assignment between line 4 and line 20. All operations in the loop take a constant amount of time, except updating of the core list in line 19, which takes $\mathbb{O}(m)$ time. The loop runs for each task. As there are n tasks, the loop has $\mathbb{O}(nm)$ time complexity. The preparation of the schedule for the migrating tasks may take $\mathbb{O}(nm)$ time. Further, Algorithm 3 is called which takes $\mathbb{O}(m \times |\mathbb{I}_k|)$ time. Therefore, the time complexity of Algorithm 2 comes to be $\mathbb{O}(m(n + |\mathbb{I}_k|))$.
- Algorithm 1 is the main algorithm for *TREAFET* which calls other algorithms. Computation of a set of intervals in line 2 takes $\mathbb{O}(n\ logn)$ time and is done only once at the start of the system. Within each interval $\mathbb{I}_k$, the computation of the starting temperature for the virtual core takes $\mathbb{O}(m)$ time. For each task, the computation of execution requirements and the final temperature for the virtual core take a constant amount of time. However, insertion of the task in the sorted list $\wedge_1$ takes $\mathbb{O}(n)$ time. Next, Algorithm 2 with time complexity $\mathbb{O}(m(n + |\mathbb{I}_k|))$ is called. As there are n tasks, the final time complexity of *TREAFET* comes to be $\mathbb{O}(n^2 + mn + m|\mathbb{I}_k|)$.

As the number of cores is much lesser than the number of tasks running in a system, the number of cores can be neglected, which brings the time complexity to be $\mathbb{O}(n^2 + |\mathbb{I}_k|)$. The preparation of the list $\wedge_1$ is done only once per interval, and the respective time complexity can further be brought down to $\mathbb{O}(n\ logn)$ from $\mathbb{O}(n^2)$ if priority queues are used. Further, for the $|\mathbb{I}_k|$ part, the time complexity is a nominal value for any online algorithm and is required to keep track of core temperatures.

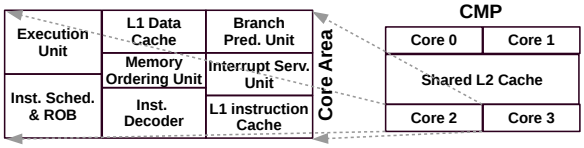


Fig. 7. Floorplan of the four core based CMP

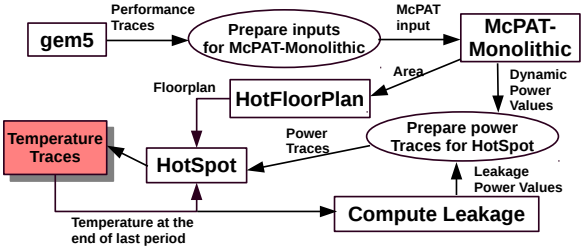


Fig. 8. Simulation Framework

4 SIMULATION INFRASTRUCTURE

We simulated a homogeneous CMP having 4 $x86$ OoO cores (that resembles Intel Xeon CMP [3]), as shown in Figure 7, in the gem5 [17] full system simulator. In addition to a core, each of these tiles contains data and an instruction local L1 cache. The L2 cache is shared among all the cores and located centrally in the CMP. A 2D-mesh-NoC connects the tiles and the L2 cache. Hence, individual tiles and the L2 cache are equipped with routers. Towards analyzing complete performance-power-thermal status, the periodic performance traces of the multithreaded PARSEC benchmarks [16], collected from gem5, are fed to McPAT-monolithic [34] for simulating power and the power values are subsequently sent to HotSpot 6.0 [85] to generate temperature. Note that we are only considering dynamic power from McPAT, whereas leakage is separately computed by employing our own leakage model that considers the temperature of individual components. Once both dynamic and leakage power values are ready, the total power of all components is derived and sent to HotSpot 6.0. To improve the accuracy in generating thermal traces, we adopt the thermal properties of 14nm FinFET technology node [21] in HotSpot 6.0. HotSpot 6.0 also considers the floorplan of the entire chip, which is generated at the beginning of the simulation by considering the area of each component derived from McPAT. The power values, chip floorplan and temperature of each component during the last period are used next to generate the current temperatures of the cores. Our performance-power-thermal simulation framework generates power and temperature traces based on the number of operations performed at each individual on-chip component, and hence, it is agnostic to the thread counts of the task being executed. With an interval of 1ms, we collected the periodic *performance traces* from gem5. Although the TEI effect in FinFET makes the frequency no longer fixed at various temperatures, for our simulation, we assume a fixed temperature during the whole span of a *PERIOD* (frame), i.e., 1ms [21]. The default parameters used in our simulations are listed in Table 4, and our closed loop performance-power-temperature simulation framework is illustrated in Figure 8.

We consider a prior TEI induced frequency model [21] to set the threshold values used in Algorithm 3 as follows: $Temp_{thr}^{Hi} = 80\,^{\circ}\text{C}$, $Temp_{thr}^{Low} = 77\,^{\circ}\text{C}$, $V_{dd}[High] = 0.75v$ and $V_{dd}[1] = 0.65v$. Note that we further assumed a maximum safe temperature of 82 °C. Hence, we set $Temp_{thr}^{Hi} = 80\,^{\circ}\text{C}$ to limit the thermal overshoot beyond 82 °C. Khan et al. have shown that the operating temperature of FinFET can also reach as high as 80–85 °C, which can be considered a hotspot [45]. By considering various technology nodes and process variations, both temperature values for determining hotspots

Table 4. System Parameters

Parameters	Values	Parameters	Values
Number of Cores	4	Core Model (Technology)	$x86$ (14nm FinFET)
Nominal Frequency	3.5GHz	$V_{dd}[1]$, $V_{dd}[High]$, v_{Tur} (at cores)	0.65v, 0.75v, 0.85v
Private L1 D/I Cache	64KiB, 4W SA, LRU	Shared L2 Cache	8MiB, 16W SA, LRU
DRAM	8GiB	Ambient Temperature	40 °C

and the threshold temperatures can be adjusted, which we intend to explore in our future work. However, to maintain an average frequency of 3.7GHz on-the-fly, we set these values so that we can maintain the lowest and the highest frequencies at 3.0GHz (for $Temp_{thr}^{Low}$, V_{Lo}) and 3.9GHz (for $Temp_{thr}^{Hi}$, V_{Hi}), respectively. Note that all of our threshold values can be tuned and can also be adjusted by considering the technical specifications of the underlying circuitry and/or technology nodes. However, our employed on-chip VR is assumed to be installed at the individual core, and each VR has a switching speed of 20 mV/ns [31], and the respective area and power overheads are based on a prior regulator-power model [81]. The calculation of timing overhead of each VRs considers t_{SW} and t_{Tur} of Figure 4, and the respective power overhead is also derived [26].

We have evaluated the following core-based techniques:

- **Baseline** – the default model with system parameters according to Table 4;
- **TREAFET** – implementation of the techniques as described in Sec. 3.3;
- **ENPASS** – a recent prior DVFS technique for the FinFET based CMPs [63].

We considered eight PARSEC applications to frame the tasks. Each of our tasks is framed with a PARSEC application running multi-threaded with a large input set. In our task model, 20 different tasks are considered, as was in our prior work, RESTORE [72]. The task-set is detailed in Table 5. In our simulation, to avoid the complexity of inter-core communication, we assumed all threads of a particular task are executed on the same core [23]. Each task set is characterized by the term *System Utilization* U, which is the ratio of the sum of task utilization u^i and the number of available cores m in the system. In other words, it denotes the ratio of the workload that has to be handled by the system and the capacity of the system. The distribution with standard deviations ranging from $\sigma_u = 0.1$ to $\sigma_u = 0.5$ and the mean $\mu_u = 0.4$ has been used to create the task utilization based on the total number of tasks n and the necessary System Utilization U. It's possible that the total of the randomly generated task utilization will not be precisely equal to U. On the other hand, keeping the summation of task utilization constant facilitates comparing and assessing the algorithms. The individual task utilization has been scaled to ensure that the total task utilization equals the necessary system utilization U. For each distinct set of input parameters, we ran 50 experiments and took their average as the final result.

Table 5. Task Details: <Task ID: PARSEC Benchmark (# Threads)>. The abbreviation details for the benchmarks are as follows: Black (Blackscholes), Body (Bodytrack), Can (Canneal), Ded (Dedup), Fluid (Fluidanimate), Stream (Streamcluster), Swap (Swaptions), X264 (X264).

T^0: Body (4)	T^1: Can (2)	T^2: X264 (4)	T^3: Fluid (2)	T^4: Stream (2)
T^5: Swap (2)	T^6: Black (2)	T^7: Body (2)	T^8: Can (4)	T^9: Ded (4)
T^{10}: Fluid (4)	T^{11}: Stream (4)	T^{12}: Swap (4)	T^{13}: Black (4)	T^{14}: Can (6)
T^{15}: Ded (6)	T^{16}: Fluid (6)	T^{17}: X264 (6)	T^{18}: Swap (6)	T^{19}: Black (6)

5 EVALUATION

Before showcasing the overall efficacy of *TREAFET*, we will first discuss the changes in frequency and instruction per second (IPS) for different benchmark applications, which are used to construct

our task set. Next, we will show how the changes in frequency and IPS finish the tasks earlier within an interval and generate slacks which are further used to improve energy saving. We will also discuss the peak temperature of the processor cores for different system utilization, and overall EDP gains will also be discussed before concluding the section.

5.1 Changes in Frequency and IPS

By employing Algorithm 3, TEI is exploited during execution, named as *ExploitTEI[C]*, whereas [C] represents scenario Figure 2[C], which improves the performance. The performance is further improved by incorporating energy-adaptive DVFS, given in Algorithm 4, just after the memory stalls at the cores, named as *ExploitTEI_MemStall[D]*, where [D] indicates scenario Figure 2[D]. By employing Algorithm 4, memory-intensive benchmarks are more benefited over the CPU-intensive and mixed ones. The higher number of memory stalls in the case of *Stream*, *Ded*, *Body*, etc., offers more scopes to run these applications at turbo frequency for a large portion of execution.

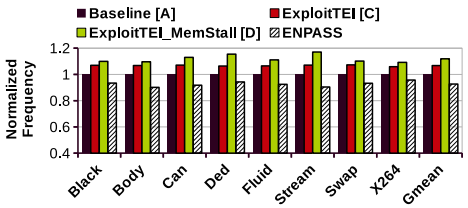


Fig. 9. Change in Frequency: *TREAFET* vs. ENPASS

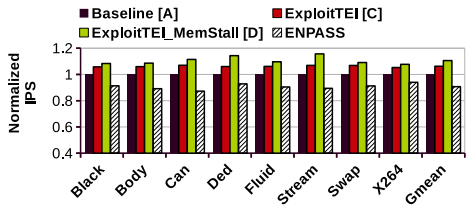
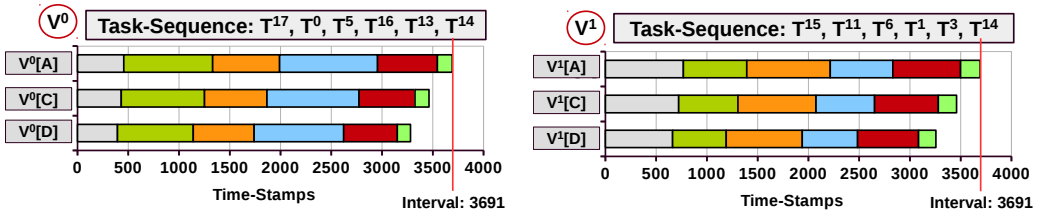


Fig. 10. Change in IPS: *TREAFET* vs. ENPASS

From our closed loop simulation framework, we periodically extracted the core temperature and the frequency for the next interval is thus updated by considering the supply voltage for all of our configurations. Finally, the average frequency across the periods is derived at the end of execution and is plotted in Figure 9. We capture the improvements in frequency for *ExploitTEI[C]* and *ExploitTEI_MemStall[D]* in Figure 9, where the frequency of *Baseline[A]* (scenario Figure 2[A]) implies the frequency assigned by Algorithm 1 and 2. However, by exploiting TEI dynamically while being thermally safe, the average frequency during execution is improved by 6 to 7.5%, with an average increase of around 6.7%. However, our energy-adaptive performance stimulation boosts up this gain further and results in a frequency increase of around 12% on average over *Baseline[A]*, with a range between 9 to 17.1%. We also capture the respective changes in IPS (instructions per seconds) by considering total instruction counts and execution time from gem5's output for all of our considered benchmarks, where *ExploitTEI[C]* shows 6.2% improvement on average over *Baseline[A]*, which is around 10.6% on average for *ExploitTEI_MemStall[D]*. The changes in IPS is plotted in Figure 10.

5.2 Impacts of Frequency Boosting in Scheduling

TREAFET first schedules the tasks by considering system-wide constraints. The hot (cold) tasks are mapped to the cold (hot) cores, and the run-time frequencies for each task are also assigned. During execution, the dynamic exploitation of TEI, along with our opportunistic energy-adaptive voltage spiking, reduces the execution span of each task. The reduction in task execution time means that tasks are finished early and generate slacks at the end of each interval. We show our schedule for an interval at two of our processor cores, $\mathbb{V}^0$ and $\mathbb{V}^1$, and the scenario is shown in Figure 11, where [A], [C] and [D] represent the scenarios depicted in Figure 2. The span of the interval is considered at 3691 units of time, where the task sequences for $\mathbb{V}^0$ and $\mathbb{V}^1$ are shown in Figure 11a and 11b,



(a) Processor P0 (b) Processor P1

Fig. 11. *TREAFET*: Task Schedule at Processors $\mathbb{V}^0$, and $\mathbb{V}^1$ in Frame 1, with 100% System Utilization. [A], [C] and [D] represent the schedule after task allocation, the modified schedule after exploiting TEI and the final schedule with TEI/SHE-aware FG-DVFS at memory stalls. Note that [A], [C] and [D] resemble [A], [C] and [D] concepts of Figure 2

respectively, and the execution times for each task is derived from gem5's output for the benchmark applications associated for the respective task. The tasks are scheduled here by considering 100% system utilization. Initial schedules (*[A]*) in both cores do not have any slacks, but both of our online techniques (*[C]* and *[D]*) reduced the execution spans of each task, thus generated slacks at the end of each schedule. The exploitation of TEI along with energy-adaptive voltage scaling in [D] trivially shows better improvement, in terms of frequency, than [C].

By considering all of our 20 tasks, scheduled on 4 processor cores, we also observed the slack spans for three consecutive intervals with different system utilization. The average percentages of time spans shared by slacks within an interval are plotted in Figure 12. Our *Baseline [A]* schedule does not have any slack with 90% or higher system utilization. However, slack is still lesser than 2% (of the entire duration of *Interval*) with 80 and 85% system utilization. With 75% system utilization, *Baseline [A]* experienced a 6% slack span in an interval, on average. While employing *ExploitTEI[C]* and *ExploitTEI_MemStall[D]*, the amount of slack is increased for all system utilization, and *ExploitTEI_MemStall[D]* shows the highest amount of slack percentages for all cases. The slack span increases while reducing the system utilization. At 75% workload, *ExploitTEI_MemStall[D]* shows an average slack span of more than 20% of an interval, which is slightly higher than 15% in the case of *ExploitTEI[C]*. The slack percentage reduces to around 7% and 10% at the higher system utilization for *ExploitTEI[C]* and *ExploitTEI_MemStall[D]*, respectively. Note that from gem5's output, we calculated the slacks for the tasks and average slack percentages for different magnitudes of system utilization are subsequently derived.

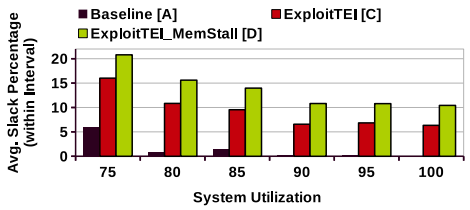


Fig. 12. Average Slack Percentages for different System Utilization and Changes due to Online Policies

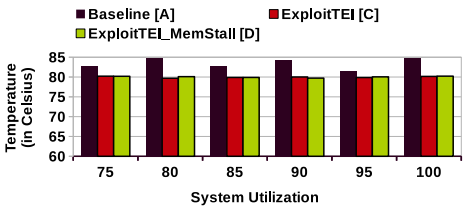


Fig. 13. Peak Temperature at different System Utilization and Reduction due to online policies

5.3 Impacts on Peak Temperature

Improving frequency can only be beneficial if the core is being operated at some safe temperature. To showcase the thermal efficiency of *TREAFET*, we also observed the peak temperature of our

cores where the temperature values are collected from HotSpot's output. The peak temperature values for different system utilization are traced during task execution, at the end of each 0.5ms [26], periodically. The values of peak temperature for different system utilization for *Baseline [A]*, *ExploitTEI[C]* and *ExploitTEI_MemStall[D]* are shown in Figure 13. The peak temperature is highest for *Baseline [A]* for all system utilization, which reaches up to 85 °C, but the reduction can be noticed for both *ExploitTEI[C]* and *ExploitTEI_MemStall[D]* than *Baseline [A]*. However, our algorithm maintains a safe peak temperature of 80 °C for *ExploitTEI[C]* and *ExploitTEI_MemStall[D]* for all processor cores with different system utilization. Note that, to exploit TEI, we need to maintain a sufficiently high but safe temperature, which in our case should not be more than 80 °C.

5.4 Energy Savings and EDP Gains

TREAFET exploits the generated slacks in *ExploitTEI[C]* and *ExploitTEI_MemStall[D]* for energy saving by power gating the cores (Algorithm 5). We further derived the energy savings for different system utilization by considering the respective slack time spans and energy values are derived by considering the execution time of each task and power consumption generated by McPAT. As slack spans are more in cases of lower system utilization, the energy saving is more than the energy saving at the higher system utilization. For all of our considered system utilization values (75% to 100%), energy savings are almost same for both *ExploitTEI[C]* and *ExploitTEI_MemStall[D]*, which are plotted in Figure 14. Actually, the energy usage in *ExploitTEI[C]* is trimmed due to generated slacks caused by shortening the runtime of tasks through TEI exploitation. For *ExploitTEI_MemStall[D]*, the energy is further saved by employing FG-DVFS during memory stalls, but this saving is further traded off to improve performance. Hence, both *ExploitTEI[C]* and *ExploitTEI_MemStall[D]* experience almost the same energy savings, which is almost close to 20% on average at 75% system utilization and close to 10% on average at full system utilization. However, to show the overall efficacy of our algorithms, we derived and plotted EDP in Figure 15. For lower system utilization of 75%, *ExploitTEI[C]* and *ExploitTEI_MemStall[D]* show EDP gains of 21% and 24%, respectively, whereas, at 100% system utilization, these values are around 11% and 14%, respectively. Overall, exploiting generated slacks improves the energy saving for both *ExploitTEI[C]* and *ExploitTEI_MemStall[D]*, and our energy-adaptive mechanism in *ExploitTEI_MemStall[D]* also keep the overall energy usage in check, while boosting up core performance.

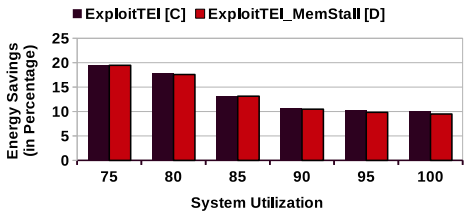


Fig. 14. Energy savings at different System Utilization over scheduling mechanism of *TREAFET*

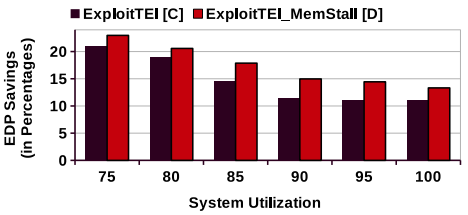


Fig. 15. EDP gains at different System Utilization over scheduling mechanism of *TREAFET*

6 STATE-OF-THE-ART

Reducing energy in modern CMP-based real-time systems has become a research topic of paramount importance in the recent past [62, 64]. Executing real-time tasks on state-of-the-art CMP platforms by considering the energy/power budget is gradually becoming challenging with technology scaling [36]. Over the recent years, researchers have been attempting to develop energy-aware

scheduling strategies for real-time task sets with different system-wide constraints [15, 37, 42]. Shrinking in process technology is gradually replacing MOSFETs with a confined 3D device, FinFET, having different power/thermal characteristics. To the best of our knowledge, none of these prior scheduling strategies considered FinFET's power/thermal characteristics. In this section, we will discuss recent thermal/energy efficient scheduling mechanisms followed by prior techniques for managing safe temperature in FinFET based CMPs.

6.1 Thermal/Energy Efficient Scheduling

Minimizing energy and temperature is essential for modern CMP-based real-time systems because reducing temperature impacts time-critical system performance, reliability, and cost-effectiveness. In a recent exploration [69], a power and thermal-aware task scheduling on multiprocessor systems has been proposed. The scheduling algorithm tries to maintain the maximum power limit and thus avoids the generation of hotspots. A multilevel scheduler, *TheSPoT* [39], has been proposed later that considers spatial and temporal thermal variations. Core consolidation and deconsolidation are performed based on peak temperature and power constraints, whereas the operating frequencies are then determined by employing a convex optimization problem. A mixed integer linear programming (MILP) model for mapping and scheduling real-time applications on CMP platforms was proposed [57], with an objective to minimize energy usage while satisfying temperature and lifetime reliability constraints. The MILP model also considers processor voltage/frequency assignment using the DVFS technique.

Zhao et al. [86] proposed a novel task scheduling framework for 3D CMP to address power supply noise and thermal issues. The framework includes power delivery network (PDN) stimuli extraction, an efficient PDN solver, and a heuristic algorithm for task scheduling. Another strategy, *VFSM* [65], has been proposed to address thermal issues at the CMPs by incorporating task migration and task swapping to reduce energy consumption and meet the high-priority task deadlines while alleviating hotspots. In another attempt [40], the authors proposed resource mapping and thread-partitioning of applications with online optimization to manage the thermal and energy efficiency of heterogeneous mobile systems. Bashir et al. [13] proposed a thermal-aware load balancing technique for CMPs by considering ambient, as well as spatial and temporal temperature gradients during execution. The technique estimates the time taken by a task set to reach temperature threshold values using offline recorded thermal profiles of datasets.

Zhou et al. [87] proposed a two-level scheduling algorithm for real-time tasks on DVFS-enabled heterogeneous MPSoC systems. In this work, at the processor level, a multi-processor model supporting DVFS is transformed into a virtual multi-processor model having only one fixed frequency level. At the core level, real-time tasks are assigned to individual cores of the virtual processor under the constraints of task precedence and peak temperature. The work by Taheri et al. [77] presents a solution to the challenges posed by CMOS technology scaling, including issues of temperature, reliability, performance and leakage power. The authors employed Stochastic Activity Networks (SANs) to model and evaluate the power consumption of a multicore system with respect to thermal constraints. The DVFS technique is used to dynamically control the temperature of cores by assigning lower voltage/frequency to the core with higher temperatures.

6.2 Thermal Management in FinFET based CMPs

Over the last decade, the industry has been gradually shifting from conventional planar MOSFET to 3D FinFET devices while designing state-of-the-art CMPs, due to lower leakage, higher circuit speed, better scalability and lower power consumption of FinFET over the MOSFET [35, 68, 75, 79]. Specifically, FinFET devices have become the prevalent choice in CMP design to alleviate the short channel effects in sub-20nm CMOS devices [6, 11]. In fact, the reduced channel length of FinFET

leads to better area efficiency. However, the power density of these FinFET devices has become a design concern with the reduced channel length [24]. Such higher power density restricts the designers and architects from fully taking advantage of the higher circuit speed of these devices. The upsides of reduced gate delay i.e. higher circuit speed in FinFETs, can be achieved if the circuit is operated at a higher temperature [24, 51, 52]. However, thermal safety needs to be guaranteed to prevent the device from breaking down. DVFS, in combination with online state-destroying caches, are widely used in managing the temperature of the high-end computer systems [26, 48, 84]. Prior arts mostly focused on MOSFET based designs, where reduced temperature improves both the performance and the energy efficiency of the system. On the contrary, the FinFET based designs benefit from an increased temperature due to TEI property, while energy and reliability get worse [63]. Over a decade, TEI in FinFET is investigated, which significantly lowers circuit delay in higher temperatures even at the super-threshold voltage region [19–21, 47, 49, 50, 83]. Kim et al. [47] have analyzed several circuit and device characteristics in detail to understand TEI in-depth. Through runtime scaling of the supply voltage, Lee et al. [50] proposed a thermal management technique for the FinFETs, while exploiting TEI. However, these prior techniques focused on the TEI effects, but its impacts on the performance of multicores were first evaluated by Cai and Marculescu [21], but time-critical applications were not considered.

Electro-thermal issues in the current generation of FinFETs, also known as SHE, have become a serious design concern for the CMPs, especially those built in sub-14nm FinFET technology [80]. In the recent past, researchers tried to reduce SHEs to maintain thermal safety in FinFET based CMPs [6, 7, 45, 53, 60]. Authors in [45] focused on the SHE and reliability issues, where the temperature of the FinFET cores can potentially be more than 80 °C due to an increase in the gate and drain temperature. The confined geometry of FinFET devices is the most significant cause of SHE. Hence, it needs to be modeled with due consideration to the 3D geometric shape in addition to the power consumption to reduce the aging process [6, 53].

6.3 *TREAFET* over Prior Art

Prior scheduling mechanisms that considered power and/or temperature as a constraint(s) are mostly based on the conventional MOSFET based CMPs. The earlier thermal efficient techniques proposed at the architecture level also targeted MOSFET based systems, where lowering temperature is always beneficial to combat the leakage power issue. Due to lower leakage in FinFET based CMPs along with the presence of TEI property, lowering the temperature of the cores might not be a viable option as long as thermal safety is guaranteed. However, maintaining thermal safety while exploiting TEI in real-time scenarios has remained a fundamental challenge, which has not yet been addressed by the prior arts. In *TREAFET*, we proposed a semi-online scheduling mechanism that schedules a set of real-time tasks at the beginning, and by employing runtime mechanisms, TEI is exploited while combating SHEs. In fact, effective frequency is further stimulated by an opportunistic energy-adaptive voltage spiking mechanism applied just after the memory-induced stalls at the cores. One such energy-adaptive performance improvement strategy has been proposed recently by Chakraborty et al. [23]. However, this prior art did not consider FinFET's thermal characteristics, and hence, the time span for spiked V/F was determined statically. Due to TEI, in *TREAFET*, we consider the current core temperature, and accordingly, the time-span for spiked V/F is determined dynamically by taking TEI into account. Additionally, the overall performance gain at the cores reduces the execution span of the tasks and thus generates slacks at the cores, which are further utilized to improve energy usage by putting the cores in sleep mode. To the best of our knowledge, *TREAFET* is the first real-time scheduling strategy that considers TEI and SHE properties of the FinFET based CMPs to boost overall performance and energy efficiency without

violating system-wide constraints. We have further summarized our related works in Table 6 to showcase how *TREAFET* is different in comparison with its prior arts.

Table 6. **Summary of Related Works**

Related Work	Energy Aware	Temperature Aware	Time Criticality	Platform
H. Salamy [69]	✓	✓	X	Homogeneous CMP(MOSFET)
A. Iranfar et al. [39]	✓	✓	X	Homogeneous CMP(MOSFET)
Y. Zhao et al. [86]	✓	✓	X	Homogeneous CMP(MOSFET)
Q. Bashir et al. [13]	✓	✓	X	Homogeneous CMP(MOSFET)
G. Taheri et al. [77]	✓	✓	X	Homogeneous CMP(MOSFET)
W. Liu et al. [57]	✓	✓	✓	Homogeneous CMP(MOSFET)
D. Rupanetti and H. Salamy [65]	✓	✓	✓	Homogeneous CMP(MOSFET)
S. Isuwa et al. [40]	✓	✓	X	Heterogeneous CMP(MOSFET)
J. Zhou et al. [87]	✓	✓	✓	Heterogeneous CMP(MOSFET)
S. Kim et al. [47]	✓	✓	X	Single core (FinFET)
W. Lee et al. [50]	✓	✓	X	Single core (FinFET)
E. Cai and D. Marculescu [21]	✓	✓	X	Homogeneous CMP (FinFET)
K. Neshatpour et al. [63]	✓	✓	X	Homogeneous CMP (FinFET)
TREAFET	✓	✓	✓	**Homogeneous CMP (FinFET)**

7 CONCLUSIONS

Rapid progress in contemporary real-time systems leads the researchers not only to focus on scheduling the competing tasks but also to concentrate on the energy/thermal aspects of the cores. Stagnation in process technology is gradually shifting the VLSI industry from conventional MOSFET to faster FinFET based multicore design. These FinFET based multicores have brought up new challenges for real-time system designers, as FinFET's performance increases with temperature, known as TEI. However, a higher temperature can accelerate the circuit aging due to self heating effects (SHEs). The existing diversity in instruction types at different execution phases of the applications changes the core temperature over time, hence the core frequency in FinFET based cores due to TEI. As core frequency plays the most pivotal role in the real-time paradigm, modern time-critical systems must be designed by considering TEI and SHEs properties of the FinFET based multicores. In this work, we propose a temperature cognizant real-time scheduler, *TREAFET*, that, as a first study, exploits the TEI feature of FinFET based multicore platforms in the context of time-criticality to meet other design constraints of real-time systems. By considering the overall progress of individual tasks along with the thermal characteristics of each of the tasks and of the cores, *TREAFET* derives a task-to-core allocation and prepares a schedule. *TREAFET* attempts to assign hot tasks to the cold cores and vice-versa. During execution, *TREAFET* analyzes the trade-offs between performance and temperature by incorporating a prudential temperature cognizant V/F scaling to exploit TEI while guaranteeing deadline and combating SHEs. Moreover, *TREAFET* stimulates the average runtime frequency by employing an opportunistic energy-adaptive voltage spiking mechanism, whereas energy saving during memory stalls is traded off by the energy usage during the execution span having spiked voltage. The stimulated frequency in *TREAFET* also finishes the tasks early, thus providing opportunities to save energy by power gating the cores, and that leads to average gains in EDP by up to 24%.

ACKNOWLEDGMENT

This work is funded by *Marie Curie Individual Fellowship (MSCA-IF), EU (Grant Number 898296).*

REFERENCES

[1] [n. d.]. AMD Ryzen Processor with Accelerator. https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/white-papers/amd-cdna-3-white-paper.pdf. Accessed: 2024-03-20.
[2] [n. d.]. Intel Xeon Phi Processor Cores. https://www.intel.com/content/dam/develop/external/us/en/documents/intel-xeon-phi-quick-start-developers-guide-mpss-3-4.pdf. Accessed: 2024-03-20.
[3] 2022. Intel Xeon Processor. https://www.intel.com/content/www/us/en/products/details/processors/xeon.html Accessed: 2024-03-20.
[4] S. Achour and M. C. Rinard. 2015. Approximate Computation with Outlier Detection in Topaz. *SIGPLAN Not.* (2015).
[5] S. Ahmed and J. H. Anderson. 2020. A Soft-Real-Time-Optimal Semi-Clustered Scheduler with a Constant Tardiness Bound. In *RTCSA*.
[6] W. Ahn et al. 2018. Integrated modeling of Self-heating of confined geometry (FinFET, NWFET, and NSHFET) transistors and its implications for the reliability of sub-20nm modern integrated circuits. *Microelectronics Reliability (Elsevier)* (2018).
[7] H. Amrouch et al. 2019. Reliability Challenges with Self-Heating and Aging in FinFET Technology. In *IOLTS*.
[8] J. H. Anderson et al. 2005. An EDF-based scheduling algorithm for multiprocessor soft real-time systems. In *ECRTS*.
[9] J. H. Anderson et al. 2016. Optimal semi-partitioned scheduling in soft real-time systems. *Journal of Signal Processing Systems* (2016).
[10] J. H. Anderson and A. Srinivasan. 2000. Early-release fair scheduling. In *Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000.*
[11] A. Bansal et al. 2006. Compact thermal models for estimation of temperature-dependent power/performance in FinFET technology. In *ASPDAC*.
[12] S. Baruah and N. Fisher. 2005. The partitioned multiprocessor scheduling of sporadic task systems. In *26th IEEE International Real-Time Systems Symposium (RTSS).*
[13] Q. Bashir et al. 2018. A scheduling based energy-aware core switching technique to avoid thermal threshold values in multi-core processing systems. *Microprocessors and Microsystems* 61 (2018).
[14] M. Bertogna et al. 2009. Schedulability Analysis of Global Scheduling Algorithms on Multiprocessor Platforms. *IEEE TPDS* (2009).
[15] A. Bhuiyan et al. 2018. Energy-efficient real-time scheduling of DAG tasks. *ACM TECS* (2018).
[16] C. Bienia et al. 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *PACT*.
[17] N. Binkert et al. 2011. The gem5 Simulator. *ACM SIGARCH CAN* (2011).
[18] E. A. Burton et al. 2014. FIVR — Fully integrated voltage regulators on 4th generation Intel® Core™ SoCs. In *APEC*.
[19] E. Cai et al. 2016. Exploring aging deceleration in FinFET-based multi-core systems. In *ICCAD*.
[20] E. Cai and D. Marculescu. 2015. TEI-Turbo: temperature effect inversion-aware turbo boost for FinFET-based multi-core systems. In *ICCAD*.
[21] E. Cai and D. Marculescu. 2017. Temperature Effect Inversion-Aware Power-Performance Optimization for FinFET-Based Multicore Systems. *IEEE TCAD* (2017).
[22] D. Casini et al. 2020. Task Splitting and Load Balancing of Dynamic Real-Time Workloads for Semi-Partitioned EDF. *IEEE Trans. on Comp.* (2020).
[23] S. Chakraborty et al. 2021. Prepare: Power-Aware Approximate Real-Time Task Scheduling for Energy-Adaptive QoS Maximization. *ACM TECS* (2021).
[24] S. Chakraborty et al. 2022. STIFF: Thermally safe temperature effect inversion aware FinFET based multi-core. In *Proceedings of the 19th ACM International Conference on Computing Frontiers*. 21–29.
[25] S. Chakraborty and H. K Kapoor. 2019. Exploring the role of large centralised caches in thermal efficient chip design. *ACM TODAES* (2019).
[26] S. Chakraborty and M. Själander. 2021. WaFFLe: Gated Cache-Ways with Per-Core Fine-Grained DVFS for Reduced On-Chip Temperature and Leakage Consumption. *ACM TACO* (2021).
[27] J. Choi et al. 2007. Thermal-aware task scheduling at the system software level. In *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*. 213–218.
[28] R. I. Davis and A. Burns. 2011. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems* (2011).
[29] R. I. Davis and A. Burns. 2011. A Survey of Hard Real-Time Scheduling for Multiprocessor Systems. *ACM Comput. Surv.* (2011).

[30] D. de Niz and L. T. X. Phan. 2014. Partitioned scheduling of multi-modal mixed-criticality real-time systems on multiprocessor platforms. In *RTAS*.
[31] S. Eyerman and L. Eeckhout. 2011. Fine-Grained DVFS Using on-Chip Regulators. *ACM TACO* (2011).
[32] S. Funk et al. 2011. DP-Fair: a unifying theory for optimal hard real-time multiprocessor scheduling. *Real-Time Systems* 47 (2011), 389–429.
[33] G. Gracioli et al. 2013. Implementation and evaluation of global and partitioned scheduling in a real-time OS. *Real-Time Systems* (2013).
[34] A. Guler and N. K. Jha. 2020. McPAT-Monolithic: An Area/Power/Timing Architecture Modeling Framework for 3-D Hybrid Monolithic Multicore Systems. *IEEE TVLSI* (2020).
[35] Shaofeng Guo et al. 2017. Towards reliability-aware circuit design in nanoscale FinFET technology: new-generation aging model and circuit reliability simulator. In *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press, 780–785.
[36] Z. Guo et al. 2017. Energy-Efficient Multi-Core Scheduling for Real-Time DAG Tasks. In *ECRTS*.
[37] Z. Guo et al. 2019. Energy-Efficient Real-Time Scheduling of DAGs on Clustered Multi-Core Platforms. In *RTAS*.
[38] C. Hobbs et al. 2019. Optimal Soft Real-Time Semi-Partitioned Scheduling Made Simple (and Dynamic). In *RTNS*.
[39] A. Iranfar et al. 2017. Thespot: Thermal stress-aware power and temperature management for multiprocessor systems-on-chip. *IEEE TCAD* (2017).
[40] S. Isuwa et al. 2019. TEEM: Online Thermal- and Energy-Efficiency Management on CPU-GPU MPSoCs. In *DATE*.
[41] M. Jin et al. 2016. Reliability characterization of 10nm FinFET technology with multi-VT gate stack for low power and high performance. In *IEDM*.
[42] K. Kanoun et al. 2014. Online energy-efficient task-graph scheduling for multicore platforms. *IEEE TCAD* (2014).
[43] S. Kato et al. 2009. Semi-partitioned Scheduling of Sporadic Task Systems on Multiprocessors. In *21st Euromicro Conference on Real-Time Systems*.
[44] O. R. Kelly et al. 2011. On Partitioned Scheduling of Fixed-Priority Mixed-Criticality Task Sets. In *IEEE International Conference on Trust, Security and Privacy in Computing and Communications*.
[45] M. I. Khan et al. 2014. Self-heating and reliability issues in FinFET and 3D ICs. In *ICSICT*.
[46] S. Khatamifard et al. 2017. ThermoGater: Thermally-aware on-chip voltage regulation. In *ISCA*.
[47] S. Kim et al. 2007. Temperature Dependence of Substrate and Drain–Currents in Bulk FinFETs. *IEEE T-ED* (2007).
[48] J. Kong et al. 2012. Recent Thermal Management Techniques for Microprocessors. *ACM Comput. Surv.* (2012).
[49] C. Lee and N. K. Jha. 2011. CACTI-FinFET: An integrated delay and power modeling framework for FinFET-based caches under process variations. In *DAC*.
[50] W. Lee et al. 2014. Dynamic thermal management for FinFET-based circuits exploiting the temperature effect inversion phenomenon. In *ISLPED*.
[51] W. Lee et al. 2014. Dynamic thermal management for FinFET-based circuits exploiting the temperature effect inversion phenomenon. In *International Symposium on Low Power Electronics and Design*. 105–110.
[52] W. Lee et al. 2017. TEI-power: Temperature Effect Inversion–Aware Dynamic Thermal Management. *ACM Trans. Des. Autom. Electron. Syst.* 22, 3, Article 51 (2017).
[53] Y. Lee et al. 2016. Consideration of BTI variability and product level reliability to expedite advanced FinFET process development. In *IEDM*.
[54] Y. Lee et al. 2019. Thermal-Aware Scheduling for Integrated CPUs–GPU Platforms. *ACM TECS* (2019).
[55] B. Lesage et al. 2012. PRETI: Partitioned Real-Time Shared Cache for Mixed-Criticality Real-Time Systems. In *International Conference on Real-Time and Network Systems*.
[56] S. Li et al. 2009. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *MICRO*.
[57] W. Liu et al. 2019. Energy-Efficient Application Mapping and Scheduling for Lifetime Guaranteed MPSoCs. *IEEE TCAD* (2019).
[58] K. A. A. Makinwa. 2018. *Temperature Sensor Performance Survey*. http://ei.ewi.tudelft.nl/docs/TSensor_survey.xls
[59] A. Mandke et al. 2011. Adaptive Power Optimization of On-chip SNUCA Cache on Tiled Chip Multicore Architecture Using Remap Policy. In *WAMCA*.
[60] S. Mei et al. 2016. New understanding of dielectric breakdown in advanced FinFET devices — physical, electrical, statistical and multiphysics study. In *IEDM*.
[61] O. Mutlu and T. Moscibroda. 2007. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In *MICRO*.
[62] S. Narayana et al. 2016. Exploring Energy Saving for Mixed-Criticality Systems on Multi-Cores. In *RTAS*.
[63] K. Neshatpour et al. 2018. Enhancing Power, Performance, and Energy Efficiency in Chip Multiprocessors Exploiting Inverse Thermal Dependence. *IEEE TVLSI* (2018).
[64] S. Pagani et al. 2015. Energy and peak power efficiency analysis for the single voltage approximation (SVA) scheme. *IEEE TCAD* (2015).

[65] D. Rupanetti and H. Salamy. 2019. Thermal-Constrained. Energy-Aware Load Management on MPSoC Architectures. In *UEMCON*. IEEE.
[66] S Saha et al. 2015. Scheduling Dynamic Hard Real-Time Task Sets on Fully and Partially Reconfigurable Platforms. *IEEE Embedded Systems Letters* (2015).
[67] S. Saha et al. 2018. Real-Time Application Processing for FPGA-Based Resilient Embedded Systems in Harsh Environments. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*.
[68] Tarun Sairam et al. 2007. Optimizing FinFET technology for high-speed and low-power design. In *Proceedings of the 17th ACM Great Lakes Symposium on VLSI*. 73–77.
[69] H. Salamy. 2015. Task scheduling on multicore embedded systems under power and thermal constraints. *International Journal of Electronics* (2015).
[70] A. Sarkar et al. 2011. A Low-Overhead Partition-Oriented ERfair Scheduler for Hard Real-Time Embedded Systems. *IEEE Embedded Systems Letters* (2011).
[71] A. Shafaei et al. 2014. FinCACTI: Architectural Analysis and Modeling of Caches with Deeply-Scaled FinFET Devices. In *ISVLSI*.
[72] Y. Sharma et al. 2022. RESTORE: Real-Time Task Scheduling on a Temperature Aware FinFET based Multicore. In *DATE*. IEEE.
[73] J. Shun and G. E. Blelloch. 2013. Ligra: A Lightweight Graph Processing Framework for Shared Memory. In *PPoPP*.
[74] M. O. Simsir et al. 2010. Fault modeling for FinFET circuits. In *IEEE/ACM International Symposium on Nanoscale Architectures*.
[75] J. Singh et al. 2018. 14-nm FinFET Technology for Analog and RF Applications. *IEEE Transactions on Electron Devices* 65, 1 (2018), 31–37.
[76] K. Stavrou and P. Trancoso. 2005. TSIC: Thermal Scheduling Simulator for Chip Multiprocessors. In *Proceedings of the Panhellenic Conference on Advances in Informatics*. Springer-Verlag.
[77] G. Taheri et al. 2018. Temperature-aware dynamic voltage and frequency scaling enabled MPSoC modeling using stochastic activity networks. *Microprocessors and Microsystems* (2018).
[78] A. Tang et al. 2015. McPAT-PVT: Delay and Power Modeling Framework for FinFET Processor Architectures Under PVT Variations. *IEEE TVLSI* (2015).
[79] T. Uemura et al. 2016. Investigation of logic circuit soft error rate (SER) in 14nm FinFET technology. In *IEEE International Reliability Physics Symposium (IRPS)*. 3B–4–1–3B–4–4.
[80] S. Venkateswarlu et al. 2018. Ambient Temperature-Induced Device Self-Heating Effects on Multi-Fin Si n-FinFET Performance. *IEEE T-ED* (2018).
[81] W. Kim et al. 2008. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *HPCA*.
[82] Z. Wang et al. 2015. Efficient task partitioning and scheduling for thermal management in multicore processors. In *ISQED*.
[83] X. Huang et al. 2001. Sub-50 nm P-channel FinFET. *IEEE T-ED* (2001).
[84] W. Zang and A. Gordon-Ross. 2013. A Survey on Cache Tuning from a Power/Energy Perspective. *ACM Comput. Surv.* (2013).
[85] R. Zhang et al. 2015. HotSpot 6.0: Validation, Acceleration and Extension.. In *University of Virginia, Tech. Report CS-2015-04*.
[86] Y. Zhao et al. 2018. Power Supply Noise Aware Task Scheduling on Homogeneous 3D MPSoCs Considering the Thermal Constraint. *Journal of Computer Science and Technology* (2018).
[87] J. Zhou et al. 2018. Thermal-aware correlated two-level scheduling of real-time tasks with reduced processor energy on heterogeneous MPSoCs. *JSA* (2018).
[88] X. Zhou et al. 2010. Thermal-Aware Task Scheduling for 3D Multicore Processors. *IEEE TPDS* 21, 1 (2010), 60–71.

Received 00 June 2023; revised 00 March 2024; accepted 07 May 2024