RESTORE: Real-Time Task Scheduling on a Temperature Aware FinFET based Multicore

Abstract—**With the advancement of technology for modern real-time systems, the designers not only have to focus on scheduling competing tasks, but also on the energy/thermal aspects of the cores. Modern FinFET based multicore platforms have unique thermal characteristics, where their speed increases with temperature. In this work, *RESTORE* as a first study, exploits this feature of FinFET based multicore platforms in the context of time-criticality to meet other design constraints of real-time systems. *RESTORE* is a temperature aware real-time scheduler for FinFET based multicore system, which comprises three phases. The first couple of phases monitor the overall progress of individual tasks and derive a task-to-core allocation, and a schedule is prepared. The last phase balances the performance and temperature on-the-fly by incorporating a prudential temperature cognizant voltage/frequency scaling while guaranteeing the schedule. Simulation results claim, *RESTORE* is able to maintain a safe and stable thermal status (peak temperature below $80\,^{\circ}$C), hence the frequency (3.7 GHz on an average) that ensures legitimate time-critical performance for a variety of workloads while surpassing state-of-the-arts.**

Index Terms—**Real-Time Systems, DVS, Temperature, FinFET, Frequency, Power/Energy**

I. Introduction and Prior Work

A system is classed as real-time if it has a dual sense of accuracy: logical and temporal. As hardware platforms' computing capabilities are steadily improving, real-time embedded systems' efficient allocation of computing resources to several competing applications continues to be a difficult job. This problem is attributable to the fact that these systems enforce strict performance, power, and time limitations, all of which must be appropriately addressed throughout the design phase for the functional correctness of the system. As modern embedded systems are constrained by strict power limits to tackle overheating, scheduling strategy must be designed considering the power/thermal issues. Additionally, modern FinFET based chip multiprocessors (CMPs) offer different thermal characteristics than their MOSFET based counterparts. The processing speed in FinFET based CMPs increases with temperature, known as temperature effect inversion (TEI), which can be exploited for improving performance. However, higher circuit temperature might accelerate the circuit aging process, known as the self-heating effect (SHE), resulting in permanent circuit failure. Hence, to meet the real-time constraints for such FinFET systems, a scheduling strategy should be amalgamated with a prudential online thermal management that considers both TEI and SHE.

Conventional schedulers for multicore follows either global [1] or partitioned [2] approach. By allowing migrations after every time slot, global schedulers offer high utilization and workload balance among the cores at the cost of higher migration overhead that proportionally scales with the number of cores. On the other hand, the partitioned approach do not allow migration after task-allocation to a core, which makes their design less complex than the global ones. However, task to core allocation is an NP-complete problem [3]. To blend the upsides of both approaches, researchers recently started developing hybrid/semi-partitioned schedulers that offer higher utilization than the partitioned category but with restricted migrations. A seminal work for semi-partitioned strategy was proposed in [4], where the scheduler allows missing of task deadlines within a stipulated bound but offers inefficient task utilization for the modern systems. In [5], 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 in [6] where the first phase employs an approximation scheme to split tasks, and the next phase is a load balancing algorithm that restricts the number of task migrations. A cluster-based scheduler was devised in [7] that initially partitions the tasks into clusters, each of which may contain a fractional processor. Next, the tasks are scheduled using global EDF in every cluster.

The rapid advancement in technology further has led the real-time system's research to include thermal management while developing efficient schedulers for modern multicore platforms [8], [9], where tasks are statically allocated based upon their thermal characteristics. Unfortunately, such offline approaches fail to consider the TEI property of FinFET based systems, where processing speed changes dynamically with temperature. Over a decade, TEI in FinFET is being investigated, which increases the operational speed of the FinFET devices at higher temperatures even in the super-threshold voltage region [10]–[14]. Kim et al. have explored this phenomenon by analyzing the circuit- and device-characteristics [13]. By scaling supply voltage dynamically, Lee et al. [12] 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 [10]. Later, Neshatpour et al. devised a TEI-aware DVFS [15] 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.

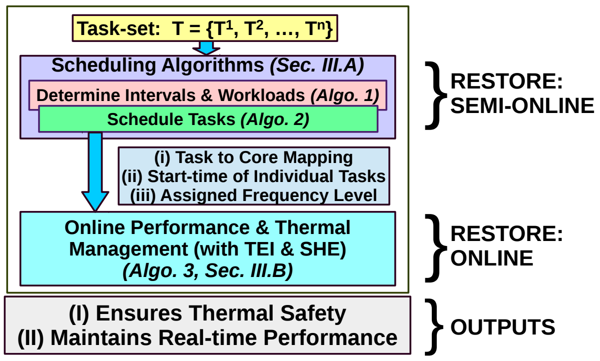
In this paper, we propose *RESTORE*, a real-time scheduling strategy for the multicore platforms that first schedule tasks at design time by assigning a particular runtime frequency for each task. Later, during execution, *RESTORE* will maintain the core temperature and frequency by prudentially balancing TEI and SHE properties of the FinFET. The proposed scheduler is based on the semi-partitioned approach and, hence, can offer high resource utilization with a limited number of migrations. The execution of tasks in the system is divided into several intervals, whose boundaries act as pseudo-deadlines for tasks. This feature not only helps the scheduler to maintain a steady rate of progress for all tasks, but also meets their final task deadlines. The runtime thermal management strategy periodically checks core temperatures and sets the processing speed for each individual core judiciously, ensuring thermal safety while meeting real-time constraints. To the best of our knowledge, *RESTORE is the first work that studies real-time systems with TEI and SHE phenomena of the FinFET based multicore.* Simulation results claim, *RESTORE* is able to maintain a safe and stable thermal status (peak temperature below 80 °C), hence the frequency (3.7 GHz on an average), that ensures legitimate time-critical performance for a range of workloads while surpassing the prior art [15].

II. System Model

The system under consideration is composed of a set of periodic tasks $\mathbb{T} = \{\mathbb{T}^1, \mathbb{T}^2, \ldots, \mathbb{T}^{|\mathbb{T}|}\}$ which is to be executed on a multicore platform $\mathbb{V} = \{\mathbb{V}^1, \mathbb{V}^2, \ldots, \mathbb{V}^{|\mathbb{V}|}\}$. Each core can run on a frequency which is chosen from the frequency set $\mathbb{F} = \{\mathbb{F}_1, \mathbb{F}_2, \ldots, \mathbb{F}_{max}\}$, such that, $\mathbb{F}_{max}$ represents the normalized frequency of 1 and all other frequencies lie between 0 and 1. Every task $\mathbb{T}^i$ needs to execute for e^i time-slots within a deadline/period of p^i. The dynamic power consumption of a core (Pow_{Dyn}) is proportional to the supply voltage (V_{dd}) and operational frequency (F) of the core, which can be written as $Pow_{Dyn} \propto V_{dd}^2 \cdot F$. Leakage power of a core is a function of the supply voltage and temperature and can be expressed as $Pow_{Leak}(V_{dd}, T_{die}) = V_{dd} \cdot (c_1 \cdot T_{die}^2 \cdot e^{(\frac{c_2 \cdot V_{dd} + c_3}{T_{die}})} + c_4 \cdot e^{(c_5 \cdot V_{dd} + c_6)})$, where c_1 to c_6 are technology dependent parameters and T_{die} represents die temperature [12]. The rate of change in temperature can be tracked by using Kirchhoff's equation for the RC-circuit thermal model: $\frac{dT_{die}}{dt} = (Pow_{circuit} - \frac{T_{die} - T_{amb}}{R_{die-amb}})/C_{die}$, where C_{die}, $R_{die-amb}$, and T_{amb} represent thermal capacitance of the die, thermal resistance between die and ambient, and ambient temperature, respectively. $Pow_{circuit}$ is the total power (i.e., summation of dynamic and leakage) consumed by the die. By using a prior measurement [16] for ARM-cortex A8 processor, we set C_{die}, and $R_{die-amb}$, as 9.0 J/K, 35.8 K/W, respectively, whereas T_{amb} is set as 40 °C. The salient presence of TEI in FinFET increases core frequency with the core temperature (T_{core}), which can be written as: $F = d_0 \cdot V_{dd}^2 + d_1 \cdot V_{dd} \cdot T_{core} + d_2 \cdot T_{core} + d_3 \cdot V_{dd} + d_4$, where d_0 to d_4 are the constants (values of which are decided empirically) [10]. The maximum temperature limit for our die is set as 80 °C.

III. *RESTORE*: Proposed Mechanism

After discussing the first two levels of our hierarchical scheduling strategy in this section, we will illustrate the dynamic temperature aware technique of *RESTORE*. The entire concept of *RESTORE* is depicted in Figure 1. Upon completion of the semi-online scheduling at the start of each interval, the following information will be produced: *(i)* task to core mapping, *(ii)* start-times of the individual tasks, and *(iii)* assigned frequency levels. This generated scheduling information will be stored in a dispatch table and will be used to execute the tasks. The tasks in this dispatch table are ordered according to their execution start-time, obtained from the scheduling policy. During execution, *RESTORE*-online looks up the dispatch table and eventually selects tasks to execute as per their start time along with the proposed dynamic thermal management. The online mechanism primarily attempts to maintain a stable frequency while considering the TEI of the underlying FinFET based cores to maintain the generated schedule.



Task-set: T = {T1, T2, ..., Tn}
Scheduling Algorithms (Sec. III.A)
Determine Intervals & Workloads (Algo. 1)
Schedule Tasks (Algo. 2)
(i) Task to Core Mapping
(ii) Start-time of Individual Tasks
(iii) Assigned Frequency Level
Online Performance & Thermal Management (with TEI & SHE) (Algo. 3, Sec. III.B)
(I) Ensures Thermal Safety
(II) Maintains Real-time Performance
RESTORE: SEMI-ONLINE
RESTORE: ONLINE
OUTPUTS

Fig. 1: *RESTORE*: Process Overview.

A. *RESTORE: Semi-online Scheduling*

In our hierarchical proposed scheduling mechanism, firstly, the tasks are executed interval by interval, and their execution demand is monitored so that they do not miss their required execution in any interval. Next, the tasks are scheduled on the available cores within each interval. In this step, we have used a semi-partitioned scheduling approach, which offers high resource utilization with a limited number of migrations.

1) Determination of Interval and Workload: Algorithm 1 maintains the overall execution progress of tasks in the system. For this purpose, execution in the system is divided into multiple intervals belonging to the interval set $\mathbb{I}$ (line 2), where an individual interval is a group of consecutive time-slots addressing a span between two sequential deadlines corresponding to the set of ready tasks [17]. Within an ensuing interval, the algorithm computes the required workload/share of each task (line 5). For the k^{th} interval ($\mathbb{I}_k$), the share of a task $\mathbb{T}^i$ can be computed as:

$$sh^i = \lceil \frac{e^i \times |\mathbb{I}_k|}{p^i} \rceil \qquad (1)$$

The interval boundaries may be considered pseudo-deadlines for the tasks. Hence, the execution of the computed shares in individual intervals helps the algorithm in maintaining a steady progress rate at the boundary of each interval and thus meet task deadlines. The computed task shares are kept in the sorted list $\wedge_1$ (line 6). Then Algorithm 2 will prepare the schedule for the interval $\mathbb{I}_k$.

Algorithm 1: DETERMINATION OF INTERVALS & WORKLOAD

```
Input: $\mathbb{T}$, $\mathbb{V}$
Output: A set of schedules for the interval set
1 Let i. Set of intervals, $\mathbb{I} = \{\mathbb{I}_1, \mathbb{I}_2, \ldots\}$, ii. $\wedge_1$ be sorted list based on shares
    of $\mathbb{T}^i$ in non-decreasing order
2 Find a set of intervals $\mathbb{I}$ using Deadline Partitioning
3 for each interval $\mathbb{I}_k \in \mathbb{I}$ do
4     for $i \leftarrow 1 : |\mathbb{T}|$ do
5         Find share $sh^i$ using Equation 1
6         $\wedge_1 \leftarrow \wedge_1 \cup \{\langle i, sh^i \rangle\}$
7     Call Algorithm 2
```

2) Schedule Tasks: Algorithm 2 prepares the basic schedule for the tasks on available cores for the interval $\mathbb{I}_k$. It starts by computing the normalized operating frequency $\mathbb{F}_{\mathbb{I}_k}$ required to handle the task set for the interval (line 1). It is the ratio of the task workload to the system capacity in the interval, which can be represented as:

$$\mathbb{F}_{\mathbb{I}_k} = \lceil \frac{\sum_{i=1}^{|\mathbb{T}|} sh^i}{\mathbb{V} \times |\mathbb{I}_k|} \rceil \tag{2}$$

If there is a significant variation in requirements of the tasks, there may be a few tasks that will not be able to complete their execution in the interval operating at $\mathbb{F}_{\mathbb{I}_k}$. It happens because the algorithm does not allow parallel execution of any task on more than one core at any particular instant. Such tasks are added to the sorted list $\wedge_{high}$, and are executed at the highest operating frequency $\mathbb{F}_{max}$. The remaining tasks of $\wedge_1$ are added to the list $\wedge_{low}$ which may be executed at the frequency $\mathbb{F}_{opt}$.

$$\mathbb{F}_{opt} = \lceil \frac{\sum_{i=1}^{|\wedge_{low}|} sh^i}{(\mathbb{V} \times |\mathbb{I}_k|) - \sum_{i=1}^{|\wedge_{high}|} sh^i} \rceil \tag{3}$$

Then it uses McNaughtons's wrap-around rule [18] to assign the prepared task sequence across available cores in the system. This rule helps the scheduler to achieve a bounded number of migrations in an interval discussed in Sec. III-C. The schedule prepared for each task is revisited next by calling Algorithm 3 (line 6) to make them energy and temperature efficient.

Algorithm 2: SCHEDULE TASKS

```
Input: $\wedge_1$, $\mathbb{V}$
Output: Schedule for current interval
1 Find $\mathbb{F}_{\mathbb{I}_k}$, the required frequency for the set of tasks in $\mathbb{I}_k$ using Equation 2
2 Divide $\wedge_1$ into two lists, $\wedge_{high}$ and $\wedge_{low}$ based on whether required
    frequency of a task is greater than or lower than $\mathbb{F}_{\mathbb{I}_k}$
3 Schedule tasks of $\wedge_{high}$ at the highest available frequency $\mathbb{F}_{max}$ using
    McNaughton's rule on first $|\mathbb{V}_{high}|$ cores
4 Let $\mathbb{F}_{opt}$ be the required frequency for rest of tasks
5 Schedule tasks of $\wedge_{low}$ at frequency $\mathbb{F}_{opt}$ using McNaughtons's rule on
    remaining $|\mathbb{V}| - |\mathbb{V}_{high}|$ cores
6 Each task $\mathbb{T}^i (\in \mathbb{T})$ will now be executed by calling Algorithm 3
```

B. RESTORE: Managing Temperature and Performance Online

The entire process of our runtime temperature and frequency management for underlying FinFET based cores is given in Algorithm 3. The algorithm breaks the execution of a task $\mathbb{T}^i$ on a core into $\lceil sh^i/\Delta \rceil$ parts, which have been called *frames* in this paper. In the beginning, the first frame of each task starts its execution at least at the computed frequency $\mathbb{F}_{opt}$ (or $\mathbb{F}_{max}$). The TEI characteristic in FinFET based cores varies the operational frequency during execution, as in FinFET cores, the frequency depends on the current core temperature and the supply voltage (V_{dd}). Basically, processing frequency increases with the temperature, which can be exploited to enhance performance, thanks to TEI, but can be at the cost of SHE induced issues. In addition, the core temperature directly depends upon its V/F level, which needs to be managed carefully to avoid any thermal induced catastrophes. Therefore, our online strategy monitors the core temperature periodically and prudentially tackles the circular dependency between temperature and core frequency, with due consideration to the frequency assigned by our scheduling algorithm.

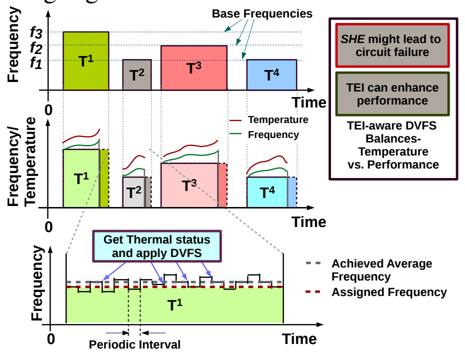

Base Frequencies
Frequency
f3
f2
f1
T1
T2
T3
T4
Time
0
Frequency/ Temperature
Temperature
Frequency
SHE might lead to circuit failure
TEI can enhance performance
TEI-aware DVFS Balances- Temperature vs. Performance
Get Thermal status and apply DVFS
Achieved Average Frequency
Assigned Frequency
Periodic Interval
Fig. 2: Online Mechanism.

Figure 2 illustrates the idea of our online technique. We have considered three frequency levels (f_1, f_2 and f_3) with four tasks (T^1 to T^4). These frequency levels are named as the base frequency for each task, determined by our scheduling strategy. As stated earlier, executing a task on a core leads to a rise in temperature that might affect its operational frequency. Hence, during the execution of individual frames for each task, the algorithm first sets a frequency having a magnitude of at least the assigned one, which is done by setting the voltage level by considering the current core temperature. As the execution progresses, the algorithm periodically checks the reached frequency and temperature of the core at the end of each frame. Once the core temperature reaches the preset higher threshold value, the voltage will be reduced so that the average frequency can be higher than the assigned one. As TEI assists us to maintain a minimum performance at higher temperatures with lower supply voltage, our runtime mechanism attempts to maintain a minimum core temperature, called as the lower threshold. Once the core temperature is reached at the lower threshold, the supply voltage will be increased to maintain a higher frequency that will complete extra execution cycles, which can be exploited while executing the task at a lower voltage/frequency.

Towards periodic monitoring of thermal status of the core, Algorithm 3 first takes the length of a frame ($min\{\Delta, sh^i_{rem}\}$) and initial core temperature ($Temp\_Init$), where sh^i_{rem} is the remaining execution share of $\mathbb{T}^i$ in the interval $\mathbb{I}_k$. In addition, the higher and lower temperature threshold values are also given as input to the algorithm ($Temp^{Hi}_{thr}$, and $Temp^{Low}_{thr}$). The viable voltage levels for the cores are also set as input to the algorithm, along with the derived base frequencies for individual tasks. After initialization of the required parameters, the algorithm fetches the scheduled tasks for execution. Once the base frequency is determined, the algorithm sets supply voltage

Algorithm 3: ONLINE PERFORMANCE AND THERMAL MANAGEMENT

```
Input: $F\_Base[1 : |\mathbb{T}|]$, $Temp\_Init$, $\Delta$, $dispatch\_table$, $V_{dd}[1 : L]$,
       $Temp_{thr}^{Hi}$, $Temp_{thr}^{Low}$, $sh_{rem}^{i}$
Output: Thermal Safety with maintained base frequency
1  $Temp = Temp\_Init$
2  for each task $\mathbb{T}^i$ do
3      # Fetch the task $\mathbb{T}^i$ and get its assigned base frequency $F\_Base[i]$,
         current temperature ($Temp$) of the core, and set $sh_{rem}^{i} = sh^{i}$
4      # Set $V_{in}$ to fix the frequency, so that, $Freq(V_{in}, Temp) \geq$
         $F\_Base[i]$, and start execution and $cycle\_cntr = 0$
5      while $\mathbb{T}^i$ is executed do
6          if $cycle\_cntr == min\{\Delta, sh_{rem}^{i}\}$ then
7              # Get the frequency and temperature for $\mathbb{V}^j$ in the last frame
8              if $(Temp) \geq Temp_{thr}^{Hi}$ then
9                  $V_{in} = V_{dd}[1]$
10             if $(Temp) \leq Temp_{thr}^{Low}$ then
11                 $V_{in} = V[L]$
12             if $Temp_{thr}^{Hi} > Temp > Temp_{thr}^{Low}$ then
13                 for $m = 2$ to $(L - 1)$ do
14                     $F_{Next} = getFreq(V[m], Temp)$
15                     if $(F_{Curr} + F_{Next})/2 \geq F\_Base[i]$ then
16                         $V_{in} = V[m]$
17                         break
18             $F_{Curr} = getFreq(V_{in}, Temp)$
19             $cycle\_cntr = 0$
20             $sh_{rem}^{i} = sh_{rem}^{i}$ - $min\{\Delta, sh_{rem}^{i}\}$
21         else
22             $cycle\_cntr + +$
```

(denoted as V_{in}) to a certain level, so that the assigned base frequency is not violated, i.e. $Freq(V_{in}, Temp) \geq F\_Base[i]$ (line 3). Subsequently, the task execution will be started. At runtime, our algorithm will collect the core temperature at the end of each frame, the length of which is determined by $min\{\Delta, sh_{rem}^{i}\}$ (line 7). If the core temperature at the end of the last frame is more than $Temp_{thr}^{Hi}$, the algorithm sets V_{in} at the lowest possible level $V[1]$ (line 8 to 9). This will reduce the core power consumption, but a higher current temperature can maintain a suitable frequency, thanks to TEI. On the other hand, once the temperature is lower than the stipulated threshold value $Temp_{thr}^{Low}$, the supply voltage will be set to its highest possible value $V_{dd}[L]$ (line 10 to 11), where L is assumed to be the number of available voltage levels. When the current temperature is within the thresholds, the algorithm will set the voltage level to a minimum possible value that can maintain an average frequency of the core at least $F\_Base[i]$ (line 12 to 17). Once the V_{in} is set, the core frequency is updated (line 18). During a frame, the core will execute the task normally, and the number of completed clock cycles is counted by employing a counter, $cycle\_cntr$ (line 22).

C. Implementation Issues and Mitigations

In *RESTORE*, the tasks are scheduled on cores sequentially. When the core under consideration does not have sufficient capacity to handle the total share of a task, the task is scheduled on that core at the end of the interval for a duration equal to the core's remaining capacity. The remaining share of the task is scheduled with the onset of the interval on a new core. Such a scheduling strategy avoids parallel execution of any task on more than one core at any time slot. Hence, a core can have at most one migrating task in an interval, and the strategy limits the number of migrations to $|\mathbb{V}| - 1$ per interval.

Voltage	**Temperature (in °C)**			
	65	**70**	**75**	**80**
	Frequency (in GHz)			
0.65	2.94	2.98	3.02	3.06
0.7	3.19	3.23	3.27	3.32
0.75	3.43	3.47	3.51	3.55
0.8	3.64	3.68	3.73	3.77

TABLE I: V/F pairs for different temperatures.

A periodic task can arrive or depart at any time in a dynamic system. When a new task ($\mathbb{T}^i$) arrives in a system, *RESTORE* first checks the task period. If there is sufficient time to finish the new task's execution starting from the end of the current interval, the scheduling of $\mathbb{T}^i$ is then procrastinated till the start of the next interval. Otherwise, the system suspends its execution from the current time instant, recomputes new intervals by including consideration of $\mathbb{T}^i$, and subsequently prepares a new schedule for the tasks. On a task departure, its entry is deleted from the task set, and it is not considered for scheduling from the next interval.

To enable the per-core DVFS in a faster manner, we employ on-chip voltage regulators (VRs) (detailed in Sec. IV) having significantly smaller timing overhead than their off-chip counterparts [19]–[22]. We also analyzed the power consumption of these on-chip VRs, as they might pose their own challenges regarding power consumption and hotspots, which can be addressed by techniques like ThermoGater [23]. To implement Algorithm 3, a monitor is employed for scaling supply voltage at the cores. This monitor triggers DVS once the temperature crosses either of the thresholds, and there exists room for regulating the supply voltage as per Algorithm 3. However, our implementation incurs limited hardware costs.

IV. EVALUATION

A. Simulation Setup

We simulated a homogeneous tiled CMP having 4 Alpha 21364 OoO cores in gem5 [24] full system simulator. Each tile contains a core, data, instruction local/private L1 caches, and an L2 cache bank. The L2 cache, the last level cache (LLC) here, is logically shared yet physically distributed into 4 uniform-sized banks and connected through a 2D-mesh-NoC. For complete performance-power-thermal analysis, periodic performance traces of the tasks (consisted of PARSEC benchmarks with large input set [25]) are collected from gem5, and are fed to McPAT-monolithic [26] for power-simulation. The power traces are next sent to HotSpot 6.0 [27] for generating the thermal traces. Note that we adopt the thermal properties of FinFET [10] in HotSpot 6.0. With a frame of 1M cycles, the periodic *performance traces* are collected from gem5. Although the TEI effect in FinFET leads to varying frequencies with temperatures but for our simulation, we assume a fixed temperature during the whole span of a frame, i.e., 1M cycles like [10]. The default parameters used in our simulation are listed in Table II and the task details are given in Table III. We varied the system utilization from 0.8 to 1.0, and the period of the tasks are generated randomly to meet the desired system utilization.

Parameters	Values
Number of Cores	4
Core Model	Alpha 21364
Nominal Frequency	3.5 GHz
V_{Hi}, V_{Lo} (at cores)	0.8v, 0.65v
L1 D/I Cache	Private 64KB, 4W SA, LRU
Shared L2 Cache bank (4 banks)	512KB, 16W SA, LRU
Ambient Temperature	47 °C
Technology	14 nm (FinFET)

TABLE II: System Parameters.

T^0: Body (4)	T^1: Can (2)	T^2: Ded (2)	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}: Stream (6)	T^{18}: Swap (6)	T^{19}: Black (6)

TABLE III: Task Details. <Task ID: PARSEC Benchmark (# Threads)>.

By considering TEI induced frequency model of [10], we set the temperature threshold and voltage limits used in Algorithm 3, which are as follows: $Temp_{thr}^{Hi}$ = 80 °C, $Temp_{thr}^{Low}$ = 75 °C, $V[L] = 0.8v$ and $V[1] = 0.65v$. Based on a prior analysis on FinFET hotspots [28], we assume a maximum safe temperature of 82 °C, hence, we set $Temp_{thr}^{Hi}$ = 80 °C that can restrict the thermal overshoot beyond 82 °C. *RESTORE* also employs on-chip VR at the cores that has a switching speed of 20 mV/ns [20], and their respective power consumption are based on an earlier regulator-power model [19].

B. Results & Analysis

We compared the performance of the following two algorithms with the proposed scheduler *RESTORE*: EDF-M and ENPASS [15]. The original EDF scheduler [29] is optimal and one of the most popular schedulers for real-time systems but targeted towards single core platforms. Hence, *RESTORE*, proposed for multicore, is able to handle significantly higher workloads than the original EDF. To make a fair comparison, we modified EDF by applying deadline partitioning, DVFS, and temperature control mechanisms with EDF, like the proposed scheduler. ENPASS basically throttles individual cores, applies DVFS, and migrates activities (unlike *RESTORE*) between cores to maintain peak temperature. Simulations for ENPASS also consider the V/F pairs and system configurations mentioned in Table I and II. Each time a core temperature reaches a midpoint, the V/F is set accordingly so that thermal safety can be maintained while exploiting TEI to enhance performance. ENPASS reduces V/F to the lowest possible level (0.65v, 2.94 GHz) as soon as the temperature reaches the highest allowable value (i.e., 80 °C), and increases the V/F to the next possible level once it reaches a temperature level below 80 °C on a scale of 5.

Peak Temperature: *RESTORE* is able to maintain a safe peak temperature of 80 °C or less. For memory intensive tasks (e.g. T^4, T^{11}, etc.), the temperature reaches close to 80 °C for a small window of arithmetic operations. However, all of our tasks are able to maintain an average peak temperature around 79 °C, which assists in exploiting TEI while reducing SHE. The peak temperature for all the tasks running on 4 cores are shown in Figure 3 where each Ci_Tj along the X-axis represents task T^j runs at the core i. Note that the peak temperature is collected by executing tasks for 3 intervals.

Frequency, Slacks, and Energy Savings: The temporal changes in core temperature vary the core frequency for all cores due to the TEI effect. *RESTORE* prudentially exploits the TEI and is able to maintain a higher core frequency than assigned ones for all tasks. This higher runtime frequency, maintained by *RESTORE*, enables us to execute workloads having a system utilization of 0.9. ENPASS although fails to execute tasks at the assigned frequencies for higher system utilization ($\geq$ 0.85); but it is able to execute the tasks with 0.8 system utilization with an assigned base frequency of 3.06 GHz, as ENPASS is able to maintain an average frequency of 3.21 GHz. We plot the runtime average frequencies maintained by ENPASS and *RESTORE* for a system utilization of 0.9 in Figure 4. The assigned frequency for this workload is 3.47 GHz, where ENPASS and *RESTORE* offer average runtime frequencies of 3.21 and 3.69 GHz, respectively. By maintaining a higher processing speed than the assigned one, *RESTORE* not only meets the deadline, but also generates slacks for each individual task. Figure 5 shows that the percentage of allotted runtime for each task has been converted to slacks. The amount of slacks for the tasks are more in the case of lower system utilization of 0.8, which is trivially lower for a higher system workload of 0.9. For 0.8, 0.85, and 0.9 system utilization values (represented as $Slack\_Amount\_80$, $Slack\_Amount\_85$, and $Slack\_Amount\_90$, respectively) *RESTORE* respectively offers 16%, 9% and 6% slack windows on an average, which further enables us to shut down the cores during the slacks. Thus, for 0.8, 0.85, and 0.9 system utilization, *RESTORE* achieves average energy savings of around 17%, 11%, and 5%, respectively.

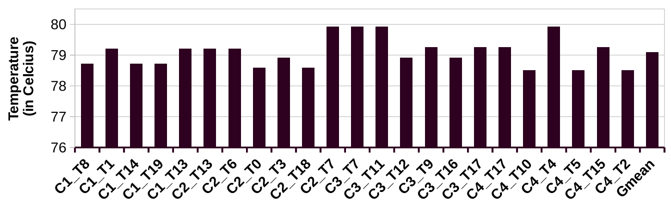


Fig. 3: Peak Temperature of the Cores.

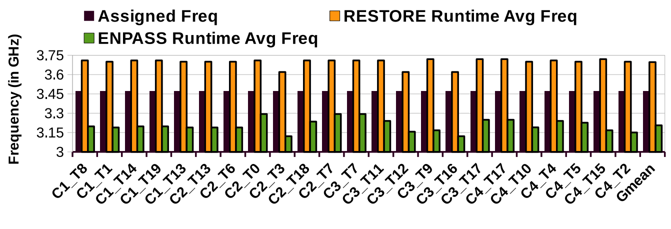


Fig. 4: Assigned vs. Actual Frequencies (Sys. Util. = 0.9).

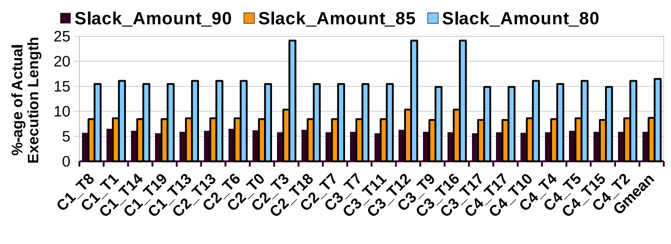


Fig. 5: Gained Slacks for different workloads.

Task Completion Ratio: In this experiment, the system utilization has been varied from 0.8 to 1.0. Since *RESTORE* and EDF-M are based on a semi-partitioned approach, they are able to handle all tasks till the system utilization of 0.85. Thereafter, EDF-M is unable to schedule some tasks which require migration within an interval. For EDF-M, migrations are only allowed at interval boundaries. We measured the ratio of tasks successfully scheduled by EDF-M to the tasks scheduled by *RESTORE* which is shown in Figure 6. The total number of tasks that required execution was 20, and *RESTORE* was able to handle all of them, even at the highest system utilization of 1.0. But due to the migration constraint of EDF-M within an interval, it is able to schedule less number of tasks than *RESTORE*, and the task completion ratio is less than 1.0 beyond 0.85 system utilization. As we observed, the ratio reduces from 1.0 to 0.9 with variance in system utilization from 0.8 to 1.0.

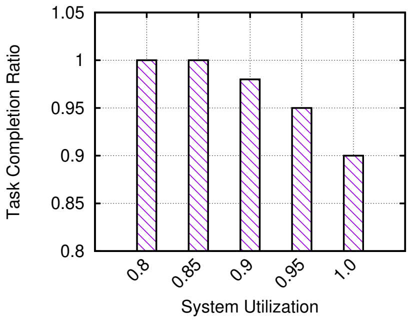


Fig. 6: Task Completion Ratio for various System Utilization.

V. Concluding Remarks

Rapid progress in VLSI technology drives real-time systems researchers to include energy/thermal aspects of the cores while scheduling time-critical tasks. Specifically, scheduling algorithms need to be reformed for the contemporary FinFET based systems by considering the TEI feature. *RESTORE*, consisting of three phases, exploits the TEI feature of FinFET based multicores for real-time paradigm while maintaining thermal as well as deadline constraints. The first two phases of *RESTORE* monitor the overall progress of individual tasks and allocate tasks to the cores by considering timing parameters. The last phase balances the performance and temperature online by incorporating a prudential thermal aware V/F scaling while guaranteeing the schedule. Simulation results claim, *RESTORE* is able to maintain a safe and stable thermal status (peak temperature below 80 °C), hence the frequency (3.7 GHz on an average) that ensures legitimate time-critical performance for a variety of workloads while surpassing state-of-the-arts. Our future work will include more interplay between SHE and TEI for the FinFET based real-time systems.

References

[1] A. Biondi *et al.*, "Schedulability analysis of hierarchical real-time systems under shared resources," *IEEE Trans. on Comp.*, 2016.

[2] S. Law *et al.*, "Industrial Application of a Partitioning Scheduler to Support Mixed Criticality Systems," in *ECRTS*, 2019.

[3] P. Arató *et al.*, "Algorithmic aspects of hardware/software partitioning," *ACM TODAES*, 2005.

[4] J. Anderson *et al.*, "An EDF-based scheduling algorithm for multiprocessor soft real-time systems," in *ECRTS*, 2005.

[5] C. Hobbs *et al.*, "Optimal soft real-time semi-partitioned scheduling made simple (and dynamic)," in *RTNS*, 2019.

[6] D. Casini *et al.*, "Task splitting and load balancing of dynamic real-time workloads for semi-partitioned EDF," *IEEE Trans. on Comp.*, 2020.

[7] S. Ahmed and J. H. Anderson, "A soft-real-time-optimal semi-clustered scheduler with a constant tardiness bound," in *RTCSA*, 2020.

[8] Z. Wang *et al.*, "Efficient task partitioning and scheduling for thermal management in multicore processors," in *ISQED*, 2015.

[9] Y. Lee *et al.*, "Thermal-aware scheduling for integrated CPUs–GPU platforms," *ACM TECS*, 2019.

[10] E. Cai and D. Marculescu, "Temperature effect inversion-aware power-performance optimization for FinFET-based multicore systems," *IEEE TCAD*, 2017.

[11] C. Lee and N. K. Jha, "CACTI-FinFET: An integrated delay and power modeling framework for FinFET-based caches under process variations," in *DAC*, 2011.

[12] W. Lee *et al.*, "Dynamic thermal management for FinFET-based circuits exploiting the temperature effect inversion phenomenon," in *ISLPED*, 2014.

[13] S. Kim *et al.*, "Temperature dependence of substrate and drain–currents in bulk FinFETs," *IEEE T-ED*, 2007.

[14] E. Cai and D. Marculescu, "TEI-Turbo: temperature effect inversion-aware turbo boost for FinFET-based multi-core systems," in *ICCAD*, 2015.

[15] K. Neshatpour *et al.*, "Enhancing power, performance, and energy efficiency in chip multiprocessors exploiting inverse thermal dependence," *IEEE TVLSI*, 2018.

[16] A. Mandke *et al.*, "Adaptive power optimization of on-chip SNUCA cache on tiled chip multicore architecture using remap policy," in *WAMCA*, 2011.

[17] S. Moulik *et al.*, "Energy aware frame based fair scheduling," *Sustainable Computing: Informatics and Systems*, 2018.

[18] J. Yang *et al.*, "A Discrete DP-Wrap Scheduling Algorithm for Multiprocessor Systems," in *IEEE Smart Cities*, 2015, pp. 958–962.

[19] W. Kim *et al.*, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *HPCA*, 2008.

[20] S. Eyerman and L. Eeckhout, "Fine-grained DVFS using on-chip regulators," *ACM TACO*, 2011.

[21] E. A. Burton *et al.*, "FIVR — fully integrated voltage regulators on 4th generation Intel® Core™ SoCs," in *APEC*, 2014.

[22] Y. Lee *et al.*, "An agile approach to building RISC-V microprocessors," *IEEE Micro*, 2016.

[23] S. Khatamifard *et al.*, "ThermoGater: thermally-aware on-chip voltage regulation," in *ISCA*, 2017.

[24] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH CAN*, 2011.

[25] C. Bienia *et al.*, "The PARSEC benchmark suite: Characterization and architectural implications," in *PACT*, 2008.

[26] A. Guler and N. K. Jha, "McPAT-Monolithic: An area/power/timing architecture modeling framework for 3-D hybrid monolithic multicore systems," *IEEE TVLSI*, 2020.

[27] R. Zhang *et al.*, "HotSpot 6.0: Validation, acceleration and extension." in *University of Virginia, Tech. Report CS-2015-04*, 2015.

[28] M. I. Khan *et al.*, "Self-heating and reliability issues in FinFET and 3D ICs," in *ICSICT*, 2014.

[29] Y. Zhu and F. Mueller, "Feedback EDF scheduling of real-time tasks exploiting dynamic voltage scaling," *Real-Time Systems*, vol. 31, no. 1-3, pp. 33–63, 2005.