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

Yanshul Sharma, Sanjay Moulik
Dept. of Computer Science and Engineering,
Indian Institute of Information Technology Guwahati,
(IIIT Guwahati), Guwahati 781015, India
yanshul.sharma@iiitg.ac.in, sanjay@iiitg.ac.in

Shounak Chakraborty
Dept. of Computer Science (IDI),
Norwegian University of Science and Technology
(NTNU), Trondheim 7491, Norway
shounak.chakraborty@ntnu.no

***Abstract*—In this work, we propose *RESTORE* that exploits the unique thermal feature of FinFET based multicore platforms, where processing speed increases with temperature, 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 that first derives a task-to-core allocation, and prepares a schedule. Next, it balances the performance and temperature on the fly by incorporating a prudential temperature cognizant voltage/frequency scaling while guaranteeing task deadlines. 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 Contributions

Modern FinFET based chip multiprocessors (CMPs) offer different thermal characteristics than conventional MOSFET based counterparts [1]–[5]. 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 follow either a global or partitioned approach [6]. By enabling task-migrations, global schedulers offer higher system throughput, but at the cost of higher migration overhead that proportionally scales with the number of cores, whereas, without allowing migrations, the partitioned schedulers attempt to propose judicious task-allocation to the cores making their implementation simple. 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 [7]–[9].

To offer high resource utilization with a limited number of migrations, in this paper, we propose *RESTORE*, a real-time semi-partitioned 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 execution of tasks in the system is divided into several intervals, whose boundaries act as pseudo-deadlines for tasks, meeting of which guarantees the final task deadlines. Our online thermal management periodically checks core temperatures and systematically sets the processing speed for each core to ensure 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 $80\,^{\circ}$C), hence the frequency (3.7 GHz on an average), that ensures legitimate time-critical performance for a range of workloads.

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

II. System Model

Our considered system is composed of a set of periodic tasks $\mathbb{T} = \{\mathbb{T}^1, \mathbb{T}^2, \ldots, \mathbb{T}^{|\mathbb{T}|}\}$ executing on a multicore platform $\mathbb{V} = \{\mathbb{V}^1, \mathbb{V}^2, \ldots, \mathbb{V}^{|\mathbb{V}|}\}$. Each core can run on a frequency chosen from its 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 the die temperature [3]. 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 [10] 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 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) [1]. The maximum temperature limit for our die is set as 80 °C.

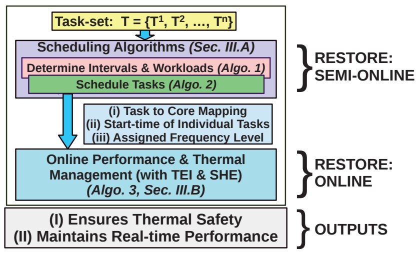

Task-set: T = {T1, T2, ..., Tn}
Scheduling Algorithms (Sec. III.A)
Determine Intervals & Workloads (Algo. 1)
Schedule Tasks (Algo. 2)
(i) Task to Core Mapping
(ii) Start-time of Individual Tasks
(iii) Assigned Frequency Level
Online Performance & Thermal Management (with TEI & SHE) (Algo. 3, Sec. III.B)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

Fig. 1: *RESTORE*: Process Overview.

III. *RESTORE*: Proposed Mechanism

The entire concept of *RESTORE* is depicted in Figure 1. Upon completion of the semi-online scheduling at the start of each interval, *RESTORE* produces: *(i)* task to core mapping, *(ii)* start-times of the individual tasks, and *(iii)* assigned frequency levels. The generated scheduling information is stored in a dispatch table and is 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 that attempts to maintain a stable frequency while considering the TEI of the underlying FinFET based cores.

A. *RESTORE: Semi-online Scheduling*

Determine Interval and Workload: Algorithm 1 maintains the overall execution progress of tasks in the system. The execution in the system is divided into multiple intervals belonging to the interval set $\mathbb{I}$ (line 2), where each interval is a group of consecutive time-slots addressing a span between two sequential deadlines corresponding to the set of ready tasks. 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: $sh^i = \lceil \frac{e^i \times |\mathbb{I}_k|}{p^i} \rceil$. The interval boundaries are considered as pseudo-deadlines for the tasks. Hence, the execution of the computed shares in individual intervals helps the algorithm maintain 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 prepares the schedule for $\mathbb{I}_k$.

Schedule Tasks: To prepare a basic schedule for the tasks on available cores for the interval $\mathbb{I}_k$, Algorithm 2 first computes the normalized operating frequency $\mathbb{F}_{\mathbb{I}_k}$, a ratio of the task workload to the system capacity in the interval, which can be represented as: $\mathbb{F}_{\mathbb{I}_k} = \lceil \frac{\sum_{i=1}^{|\mathbb{T}|} sh^i}{\mathbb{V} \times |\mathbb{I}_k|} \rceil$. $\mathbb{F}_{\mathbb{I}_k}$ is needed to handle the task set for the interval (line 1). If there is a significant variation in task requirements, there may be a few tasks that will not be able to complete their execution during $\mathbb{I}_k$ (at normalized

Algorithm 1: GET INTERVAL & 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 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 $sh^i$
6         $\wedge_1 \leftarrow \wedge_1 \cup \{\langle i, sh^i \rangle\}$
7     Call Algorithm 2
```

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 $\mathbb{I}_k$
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
```

frequency $\mathbb{F}_{\mathbb{I}_k}$), as no task can have parallel execution on multiple cores at any particular instant. Such tasks are added to the sorted list $\wedge_{high}$ to be 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 $\mathbb{F}_{opt}$, where $\mathbb{F}_{opt} = \lceil \frac{\sum_{i=1}^{|\wedge_{low}|} sh^i}{(\mathbb{V} \times |\mathbb{I}_k|) - \sum_{i=1}^{|\wedge_{high}|} sh^i} \rceil$. It uses McNaughtons's wrap-around rule [11] next to assign the prepared task sequence across available cores with a limited number of migrations.

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-span of a task $\mathbb{T}^i$ on a core into $\lceil sh^i / \Delta \rceil$ parts, which we call as *frames*. Here, Δ is the maximum allowed frame size given as an input to the algorithm. Towards periodic monitoring of the core-temperature, 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 $\mathbb{I}_k$. The higher and lower temperature thresholds are also given as input to the algorithm ($Temp^{Hi}_{thr}$, and $Temp^{Low}_{thr}$), along with the viable voltage levels for the cores, and the derived base frequencies for individual tasks. After initializing the required parameters, the scheduled tasks are fetched for execution. Next, the algorithm sets supply voltage (V_{in}) 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 initiated. At runtime, our algorithm will collect the core temperature at the end of each frame, the length of which is determined by $min\{\Delta, sh^i_{rem}\}$ (line 7). If the core temperature at the end of the last frame is more than $Temp^{Hi}_{thr}$, the algorithm sets V_{in} at the lowest possible level $V_{dd}[1]$ (line 8 to 9). It will reduce the core power consumption, but higher temperature maintains a suitable frequency, thanks to TEI. On the other hand, once the temperature is lower than $Temp^{Low}_{thr}$, V_{in} will be set to its highest possible value $V_{dd}[L]$ (line 10 to 11), where

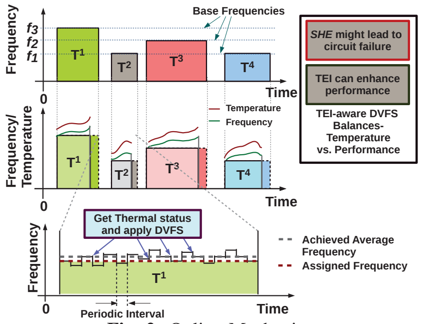
Algorithm 3: *RESTORE*:RUNTIME MANAGEMENT

```
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                 $V_{in} = V_{dd}[L]$

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                         $V_{in} = V_{dd}[m]$
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             $cycle\_cntr++$
```


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

L denotes 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 V_{in} is set, the core frequency is updated (line 18). During a frame, the core will execute a task normally, and the number of completed clock cycles is counted by employing a counter, $cycle\_cntr$ (line 22). Figure 2 illustrates the idea of our online technique 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 the temperature is monitored and exploited periodically to apply DVFS while guaranteeing the task deadlines.

C. Implementation of RESTORE

As our scheduling strategy avoids parallel execution of any task on multiple cores, 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. Moreover, based on the individual tasks' arrival and scheduled finish time-stamps, *RESTORE* adjusts the scheduling by procrastinating or suspending the task executions. On suspended execution, a new schedule

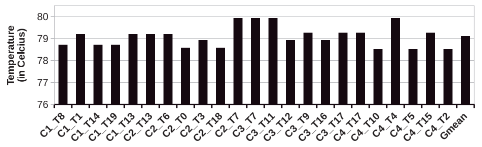


Fig. 3: Peak Temperature of the Cores.

is prepared, and on a task departure, the respective task entry is deleted from the task set to avoid considering it 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 [12], [13], and thermal footprints of such VRs can be mitigated by adopting techniques like ThermoGater [14]. To implement Algorithm 3, a monitor is employed that evaluates the core's thermal status and triggers voltage scaling on-demand.

IV. Evaluation, Summary, and Future Work

Simulation Setup: We simulated a homogeneous tiled CMP having 4 Alpha 21364 OoO cores, where each tile contains a core, data, instruction local/private L1 caches, and a shared last level L2 cache bank. For complete periodic performance-power-thermal analysis, we integrated gem5 [15], McPAT-monolithic [16], and HotSpot 6.0 [17] simulators, and the thermal properties of FinFET are adopted [1]. In our simulation, we assume a fixed temperature during the whole span of a frame, i.e., $\Delta = 1$M cycles like [1]. The default parameters used in our simulation are listed in Table II and the task details (consisted of PARSEC benchmarks with large input set [18]) are given in Table III. The system utilization varies from 0.8 to 1.0, and tasks-periods are generated randomly to meet the desired system utilization. The temperature threshold and voltage limits used in Algorithm 3 are: $Temp_{thr}^{Hi} = 80\,°\mathrm{C}$, $Temp_{thr}^{Low} = 75\,°\mathrm{C}$, $V[L] = 0.8v$ and $V[1] = 0.65v$ [1], [19].
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 Cj_Ti along the X-axis represents task T^i runs at the core V^j. Note that the peak temperature is collected by executing tasks for 3 intervals.
Frequency, Slacks, and Energy Savings: We compared performance of *RESTORE* with a prior work ENPASS [20]. *RESTORE* exploits TEI to maintain a higher core frequency than assigned ones, which enables it to execute workloads with system utilization of 0.9. ENPASS fails to execute tasks at the assigned frequencies for system utilization over 0.85, but can execute tasks with 0.8 system utilization with a base frequency of 3.06 GHz, as ENPASS maintains an average frequency of 3.21 GHz. The average runtime frequencies maintained by ENPASS and *RESTORE* for a system utilization of 0.9 is shown in Figure 4. The assigned frequency for this workload is 3.47 GHz, where ENPASS and *RESTORE* offer average runtime

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.

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

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)>.

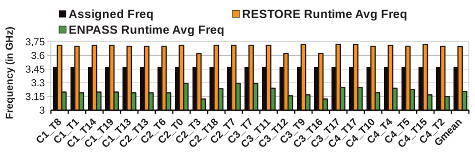


Fig. 4: Assigned vs. Actual Freq. (Sys. Util. = 0.9).

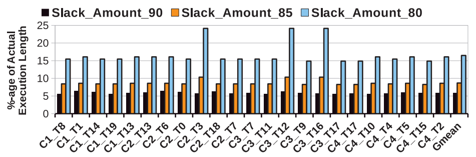


Fig. 5: Gained Slacks for different workloads.

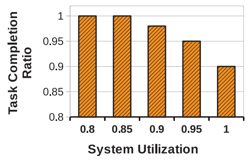


Fig. 6: Task Completion Ratio for various System Utilization.

frequencies of 3.21 and 3.69 GHz, respectively. By maintaining a higher processing speed, *RESTORE* not only meets the deadline but also generates slacks for each 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 a low system utilization of 0.8, 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 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.

Task Completion Ratio: We compared the performance of *RESTORE* with EDF-M, a semi-partitioned version of the Earliest Deadline First (EDF) scheduler. As we can observe from Figure 6, EDF-M is unable to schedule some tasks which require migration within an interval after the workload of 0.85. 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, it can schedule a lesser number of tasks, and the task completion ratio is less than 1.0 beyond 0.85 system utilization. As observed, the ratio reduces from 1.0 to 0.9 with variance in the system workload from 0.8 to 1.0.

Summary and Future Work: *RESTORE* exploits 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 online phase balances the performance and temperature 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), so 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. Developing a SHE and TEI cognizant optimal scheduling strategy for the FinFET based CMPs can be a promising research direction, which we intend to take as our future work.

References

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

[2] 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.

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

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

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

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

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

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

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

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

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

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

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

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

[15] N. Binkert *et al.*, “The gem5 simulator,” *ACM SIGARCH CAN*, 2011.

[16] 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.

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

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

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

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