Prepare: Power-Aware Approximate Real-time Task Scheduling for Energy-Adaptive QoS Maximization

SHOUNAK CHAKRABORTY*, Department of Computer Science,
Norwegian University of Science and Technology (NTNU), Norway
SANGEET SAHA*, Embedded and Intelligent Systems Laboratory, University of Essex, UK
MAGNUS SJÄLANDER, Department of Computer Science,
Norwegian University of Science and Technology (NTNU), Norway
KLAUS MCDONALD-MAIER, Embedded and Intelligent Systems Laboratory, University of Essex, UK

Achieving high result-accuracy in approximate computing (AC) based real-time applications without violating power constraints of the underlying hardware is a challenging problem. Execution of such AC real-time tasks can be divided into the execution of the *mandatory part* to obtain a result of acceptable quality, followed by a partial/complete execution of the *optional part* to improve accuracy of the initially obtained result within the given time-limit. However, enhancing result-accuracy at the cost of increased execution length might lead to deadline violations with higher energy usage.

We propose *Prepare*, a novel hybrid offline-online approximate real-time task-scheduling approach, that first schedules AC-based tasks and determines operational processing speeds for each individual task constrained by system-wide power limit, deadline, and task-dependency. At runtime, by employing fine-grained DVFS, the energy-adaptive processing speed governing mechanism of *Prepare* reduces processing speed during each last level cache miss induced stall and scales up the processing speed once the stall finishes to a higher value than the predetermined one. To ensure on-chip thermal safety, this higher processing speed is maintained only for a short time-span after each stall, however, this reduces execution times of the individual task and generates slacks. *Prepare* exploits the slacks either to enhance result-accuracy of the tasks, or to improve thermal and energy efficiency of the underlying hardware, or both. With a 70 − 80% workload, *Prepare* offers 75% result-accuracy with its constrained scheduling, which is enhanced by 5.3% for our benchmark based evaluation of the online energy-adaptive mechanism on a 4-core based homogeneous chip multi-processor, while meeting the deadline constraint. Overall, while maintaining runtime thermal safety, *Prepare* reduces peak temperature by up to 8.6 °C for our baseline system. Our empirical evaluation shows that constrained scheduling of *Prepare* outperforms a state-of-the-art scheduling policy, whereas our runtime energy-adaptive mechanism surpasses two current DVFS based thermal management techniques.

*Both authors contributed equally to this research.

Authors' addresses: Shounak Chakraborty, shounak.chakraborty@ntnu.no, Department of Computer Science,
Norwegian University of Science and Technology (NTNU), Sem Sælandsvei 9, Gløshaugen, Trondheim, Norway, 7491; Sangeet Saha, sangeet.saha@essex.ac.uk, Embedded and Intelligent Systems Laboratory, University of Essex, Colchester, UK; Magnus Själander, magnus.sjalander@ntnu.no, Department of Computer Science,
Norwegian University of Science and Technology (NTNU), Sem Sælandsvei 9, Gløshaugen, Trondheim, Norway, 7491; Klaus McDonald-Maier, kdm@essex.ac.uk, Embedded and Intelligent Systems Laboratory, University of Essex, Colchester, UK, CO4 3SQ.

This article appears as part of the ESWEEK-TECS special issue and was presented in the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2021.

1539-9087/2021/1-ART1 $15.00
https://doi.org/10.1145/3476993

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**; **Real-time systems**; **Architectures**; • **Hardware → Power and energy**.

1 INTRODUCTION

Contemporary real-time systems are constrained by deadlines and strict power limits of the underlying hardware. For such systems, approximated results obtained within the given deadline, while respecting power constraint are preferable to accurate results obtained after the deadline [31]. Such approximated results can be generated by real-time computing approaches that are inspired by approximate computing (AC), where each task is decomposed into a mandatory part, which generates results with minimum acceptable accuracy, followed by an optional part [8]. The result-accuracy increases with the number of execution cycles spent on the optional part. Specifically, to achieve a noticeable amount of increase in result-accuracy, a certain number of additional cycles need to be executed from the optional part. However, in order to maximize the result-accuracy, while meeting the power and deadline constraints, proper scheduling approaches have to explore both the architectural characteristics of the system and the approximation tolerance of the applications.

Scheduling approximated real-time tasks with the objective to maximize result-accuracy without violating the power limits of the hardware has become an active research topic in the recent past. Stavrinides and Karatza were among the first to propose real-time scheduling of approximated tasks [43]. In recent theoretical analyses [32, 33], approximated real-time task-scheduling techniques with energy and deadline constraints have been proposed. Mo et al. [33], in their theoretical analysis, improved system level result-accuracy through task to processor allocation, and task adjustment constrained by a preset energy budget. *However, restriction on energy usage does in itself not guarantee the maintenance of a safe peak temperature of the chip. This can be addressed by incorporating power constraint together with a runtime power management technique based on several architectural parameters. However, comprehensive studies that combine the theoretical aspects of energy-efficient real-time scheduling of approximated tasks along with runtime architectural characteristics have not previously been conducted.*

We devise an integer linear programming (ILP) based technique to schedule approximated real-time tasks on a chip multi-processor (CMP) platform, where scheduling is constrained by system-wide power consumption, task-dependency, and deadlines. Each of these approximated real-time tasks is equipped with multiple versions having a diverse set of result-accuracy based on the respective amount of the optional part that is executed. With an objective to maximize the result-accuracy while respecting the given constraints, our ILP based scheduling scheme provides the following information for task-execution: which versions of the individual task (*Version ID*) will be executed on which core (*Processor ID*) along with the task starting times (*Start-Time Instant*) and the processing speed (*voltage/frequency (V/F) setting*).

At runtime, our energy-adaptive power-performance management mechanism will employ fine-grained DVFS (FG-DVFS) to reduce the V/F settings at the individual cores during last level cache (LLC) miss induced stall cycles. Once the stall finishes, the V/F setting is scaled up to the next possible higher level over the assigned one (by the constrained scheduling) for a *short time-span*, so that the thermal safety is guaranteed. Actually, the gained energy savings by maintaining a lower V/F setting during memory stalls is traded against improved performance by applying a higher V/F setting, after the stall in an energy-adaptive manner. This energy-adaptive strategy ensures that accelerating processing speed for a *short time-span* should not violate the amount of energy saved by lowering the V/F during LLC-miss induced stall. Thus, it decides the length of the *short time-span*. However, as contemporary applications [2, 5, 40], that include approximations spend a significant amount of time accessing memory, increasing the frequency for a *short time-span* on the completion of each stall will reduce the total execution-time of the tasks and will generate slacks. *Prepare* attempts to exploit these slacks to enhance the result-accuracy by executing a better optional version of the task (based on its availability), or by power gating the core to enhance thermal and energy efficiency of the CMP, or both, were possible. The stall-aware energy-adaptive V/F management technique enhances result-accuracy while maintaining thermal safety, without impacting the predetermined schedule and slack-cognizant core-gating significantly reduces the temperature of the cores. The working mechanism of *Prepare* is summarized in Figure 1.

The contributions of our proposed technique can be summarized as follows:

- We schedule a set of dependent approximated real-time tasks on a power constrained chip multi-processor with an objective to maximize the result-accuracy (i.e., Quality of Service (QoS)) by employing an ILP based strategy (see Sec. 3.1).
- We apply an energy-adaptive strategy that prudentially reduces execution times of the individual task by exploiting LLC-miss induced stall cycles and generating slacks by accelerating the processing speed for a stipulated *short time-span*, while guaranteeing the thermal safety (see Sec. 3.2), which we have empirically validated and reported in Figure 13 and Figure 14.
- We further exploit the slacks in either or both of the following ways:

(1) to enhance the result-accuracy by executing higher version from the optional part of the tasks (based on availability),
(2) to improve thermal and energy efficiency of the CMP by power gating the cores.

We argue and empirically validate the significance of task-scheduling approach of *Prepare* in combination with the energy-adaptive runtime mechanisms (see Sec. 4). We achieve 75% result-accuracy (QoS) with our ILP based constrained scheduling for a set of tasks, for which a recent prior policy [33] achieves a QoS of 60%. Our evaluation (on our baseline homogeneous chip multiprocessor with four cores) reveals that the runtime energy-adaptive mechanism of *Prepare* further enhances the achieved QoS by 5.3%, while meeting the deadlines. *Prepare* applies power gating during slacks, which reduces the peak temperature compared to our baseline system by up to 8.6 °C, without compromising the performance and also outperforms two state-of-the-art DVFS based techniques, GDP [1] and *Integrate* [13]. Both of these prior policies can reduce peak temperature by 8 °C, however GDP looses 8% in performance and hence, may not be an apt choice for real-time paradigms. With an objective to maximize the slacks, *Integrate* schedules the tasks offline, which is maintained during the execution, hence, does not offer runtime performance benefits. To the best of our knowledge, *Prepare is the first scheduling mechanism that considers runtime architectural phenomena (LLC-miss induced stalls) to conduct energy-adaptive FG-DVFS for enhancing the result-accuracy of dependent approximated real-time task-sets without violating deadline constraints while maintaining thermal safety.*

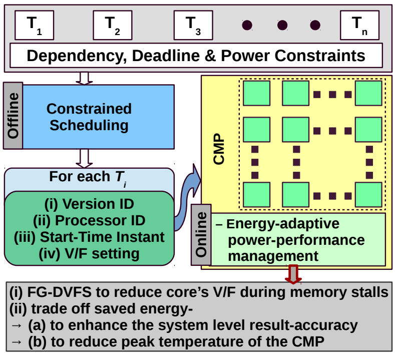
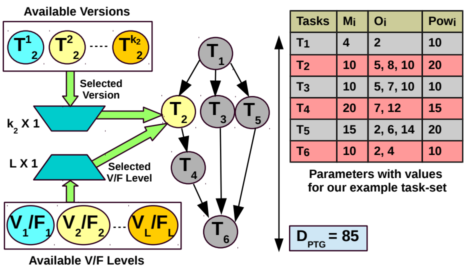

T_1
T_2
T_3
T_n
Dependency, Deadline & Power Constraints
Offline
Constrained Scheduling
For each T_i
(i) Version ID
(ii) Processor ID
(iii) Start-Time Instant
(iv) V/F setting
CMP
Online
– Energy-adaptive power-performance management
(i) FG-DVFS to reduce core's V/F during memory stalls
(ii) trade off saved energy-
→ (a) to enhance the system level result-accuracy
→ (b) to reduce peak temperature of the CMP

Fig. 1. Overview of *Prepare*.



Tasks	M_i	O_i	Pow_i
T_1	4	2	10
T_2	10	5, 8, 10	20
T_3	10	5, 7, 10	10
T_4	20	7, 12	15
T_5	15	2, 6, 14	20
T_6	10	2, 4	10

Parameters with values for our example task-set

$D_{PTG} = 85$

Fig. 2. Precedence task graph (PTG).

2 SYSTEM MODEL AND ASSUMPTIONS

We consider a CMP consisting of m homogeneous cores, denoted as $P = \{P_1, P_2, ..., P_m\}$. Each of these cores can support L distinct V/F settings denoted as $V = \{V_1, V_2, ..., V_L\}$ and $F = \{F_1, F_2, ..., F_L\}$, where $V_i < V_{i+1}$ and $F_i < F_{i+1}$. The considered task-model is represented as a task graph (see Figure 2), $G = (T, E)$, where T is a set of tasks ($T = \{T_i \mid 1 \leq i \leq n\}$) and E is a set of directed edges ($E = \{\langle T_i, T_j\rangle \mid 1 \leq i, j \leq n; i \neq j\}$), representing the precedence relations between distinct pair of tasks. An edge $\langle T_i, T_j\rangle$ refers to a precedence, that a task T_j can begin its execution only after the completion of T_i. The source and sink tasks have no predecessors and no successors, respectively. Being a real-time application, G must be executed within the given deadline, D_{PTG}, by executing all of its associated task (T_i). We assume that our task graph has a single source node and a single sink node.

For each task T_i ($1 \leq i \leq n$), the worst case execution length, len_i, is logically decomposed into M_i cycles for the mandatory part, and O_i maximum cycles for the optional part. We further assume that a task T_i may have k_i different versions, that is, $T_i = \{T_i^1, T_i^2, \ldots, T_i^{k_i}\}$, which are distinct by their given execution lengths in cycles (O_i), denoted as O_i^1, O_i^2, ..., $O_i^{k_i}$, where O_i^p achieves higher result-accuracy than O_i^q, if $p > q$. For each O_i^j (where $1 \leq j \leq k_i$), there exists a separate executable entity, which is executed after complete execution of the mandatory portion (M_i) of the respective task (T_i). The length (len_i^j) of the j^{th} version of task T_i (i.e., T_i^j where $1 \leq j \leq k_i$) can now be defined as:

$$len_i^j = M_i + O_i^j \tag{1}$$

Note that, length of T_i^j (i.e., len_i^j) includes the memory cycles required for accessing LLC, which has been obtained by executing individual task for a specific configuration (see Figure 4). The result-accuracy Acc_i^j of the T_i^j is defined by the executed optional part of the task, O_i^j (i.e., $Acc_i^j = O_i^j$). Thus, the overall system level result-accuracy (QoS) is defined as the sum of the executed cycles of O_i^j for all the tasks [8] and can be represented as:

$$QoS = \sum_{i=1}^{n} O_i^j \mid T_i = T_i^j \tag{2}$$

The temperature of η-th core (θ_η) is a function of its power consumption over time and can be expressed as:

$$\forall \eta : 1 \leq \eta \leq m \mid \theta_\eta = func(Pow, time) \tag{3}$$

and, the peak temperature (θ_p) of the system can be written as: $\theta_p = max(\theta_\eta)$. If a task T_i executes at frequency F_i, then its execution time ET_i can be denoted as $\frac{len_i}{F_i}$, and if $F_a > F_b$, then $\frac{len_i}{F_a} < \frac{len_i}{F_b}$. However, it has to be noted that increased frequency might significantly increase the effective temperature due to raised power consumption. The dynamic power consumption of a processor-core while executing a task, is a function of its V/F setting, while the static power is a function of supply voltage (V) and temperature (θ). The total power consumed by the task T_i, while executing at F_a, is denoted as $Pow_{i,a}$. We further assume an overall system-wide power limit (Pow_{BGT}), which includes both dynamic and static power, where the estimation for the static power in our theoretical model has been performed by considering a fixed temperature.

3 *PREPARE*

In this section, the working mechanism of *Prepare* is discussed. After discussing the constrained scheduling in Sec. 3.1, we illustrate the runtime power-thermal management of *Prepare* in Sec. 3.2. At the end of the constrained scheduling, the following information will be generated: *(i)* task to core mapping, *(ii)* start-times of individual task, *(iii)* assigned frequency, and *(iv)* respective tasks' versions. The generated scheduling information is stored in a dispatch table, that is used to execute the tasks. All individual tasks in the dispatch table are ordered according to their execution start-time, as obtained from the offline schedule. At runtime, *Prepare*-online traverses the dispatch table and eventually selects the tasks to run as per their start time along with the dynamic power-thermal management. Moreover, information regarding the individual task versions are also stored in this table.

To validate *Prepare* empirically, first we employ the tool CPLEX [7] for verification of the constrained scheduling, with an example task-set represented as a DAG, where tasks are created by PARSEC applications [5] (see Sec. 4). Next, the generated information for this task-set is used (through the dispatch table) in our online simulation framework consisting of gem5 [6] (a full system simulator for performance traces), McPAT [29] (power simulator) and HotSpot 6.0 [45] (for simulating on-chip thermal status). The evaluation of online mechanism considers a 4 out-of-order (OoO) core based tiled CMP architecture, which will be discussed further in Sec. 4 in addition with the detailed simulation setup. To enable FG-DVFS at the cores, *Prepare* incorporates on-chip voltage regulators (VRs) (per-core), power usage and thermal footprint of which are also accounted in our evaluation, whereas the V/F transition mechanisms used in FG-DVFS are discussed in Sec. 3.2.

3.1 Constrained Scheduling

Before presenting our *ILP* based scheduling strategy, first we define a binary decision variable $Z_{iklt\eta}$, where $i = 1, 2, ..., n$; $k = 1, 2, ..., k_i$; $l = 1, 2, ..., L$; $t = 0, 1, ..., D_{PTG}$, and $\eta = 1, 2, ...m$. Here, indices i, k, l, t and η, denote task ID, corresponding version ID, a particular V/F level, timestamp, and processor ID, respectively. The variable $Z_{iklt\eta}$ will be 1, if the ILP solver finds a solution that k^{th} version of T_i (T_i^k) can start its execution at t^{th} timestamp on processor η by selecting l^{th} V/F level. This will eventually enforce that $Z_{iklt\eta}$ for T_i will be zero, for all other possible combination, i.e. it cannot start on any other processors with other versions at any time stamp.

For each task T_i, its completion time should be no later than the given deadline (D_{PTG}). Now, the latest start time Te_i^l of T_i will be assigned by considering the version of T_i having minimum execution time ($min(len_i^k)$). Hence, the upper bound of a task start time can be restricted by the following equation: $Te_i^{la} = D_{PTG} - \underset{1 \leq k \leq k_i, 1 \leq l \leq L}{min(len_i^k / F_l)}$

Let us assume the start time of the task T_j is denoted by st_j and it can be defined as:

$$st_j = \sum_{\eta=1}^{m} \sum_{k=1}^{k_j} \sum_{t=0}^{Te_i^{la}} \sum_{l=1}^{L} t \cdot Z_{jklt\eta} \tag{4}$$

The execution length of T_i (say, el_i) can be calculated as:

$$el_i = \sum_{\eta=1}^{m} \sum_{k=1}^{k_i} \sum_{t=0}^{Te_i^{la}} \sum_{l=1}^{L} \lfloor \frac{len_i^k}{F_l} \rfloor \cdot Z_{iklt\eta} \tag{5}$$

The end time of task T_i can be denoted as et_i and it can be calculated by adding the execution length from its start time, i.e.

$$et_i = st_i + el_i \tag{6}$$

The required constraints on the decision variable to model our scheduling strategy are stated as follows.

(1) ***Unique Execution Start Time Constraint:*** Each task must start its execution with a specific V/F level on a particular processor, at a unique time stamp. That is,

$$\forall i : 1 \leq i \leq n \mid \sum_{k=1}^{k_i} \sum_{t=0}^{Te_i^{la}} \sum_{l=1}^{L} \sum_{\eta=1}^{m} Z_{ilkt\eta} = 1 \tag{7}$$

The above constraint enforces the following:

- For each task T_i, exactly one version will be selected for execution using a particular V/F level.
- T_i will start its execution on a processor at a unique time stamp within $[0, D_{PTG}]$.

(2) **Resource Constraint:** Resource bounds for processors must be satisfied at each time stamp. Any processor can execute at most one task at a given time without any preemption. In this regard, the following phenomenon needs to be hold true:

Lemma 1: *If a task T_i^k has still not finished execution at the t^{th} time stamp, it must have started at most within $(t - \lfloor \frac{len_i^k}{F_l} \rfloor + 1)$ previous time stamps. Hence, for this duration the proposed binary variable $(Z_{ilkt\eta})$ should exhibit* 1*:* $\sum_{t'=\psi}^{t} Z_{iklt'\eta} = 1$, where, $\psi = max(0, t - \lfloor \frac{len_i^k}{F_l} \rfloor + 1)$.

Hence, for all tasks and for all processors, the resource constraint can be defined as:

$$\forall t : 0 \leq t \leq D_{PTG} \ \& \ \forall \eta : 1 \leq \eta \leq m \mid \sum_{i=1}^{n} \sum_{k=1}^{k_i} \sum_{l=1}^{L} \sum_{t'=\psi}^{t} Z_{iklt'\eta} \leq 1 \tag{8}$$

Equation 8 ensures that at any time stamp t, a processor is busy due to ongoing execution of at most one task.

(3) ***Execution Dependency Constraint:*** Corresponding to each directed edge ($\langle T_i, T_j \rangle \in E$) in the PTG, the execution of task T_j must commence only after the completion of its predecessor, T_i. Hence, by using Equation 4 and 6, the dependency constraint between task T_i and T_j is symbolically represented as:

$$\forall \langle T_i, T_j \rangle \in E \mid st_j \geq et_i \tag{9}$$

(4) ***Deadline Constraint:*** In order to ensure that the task graph G meets its end-to-end absolute deadline D_{PTG}, the sink node T_n must complete execution by D_{PTG}. By using Equation 4 and 5, this constraint can be represented as:

$$st_n + el_n \leq D_{PTG} \tag{10}$$

(5) **Power Constraint:** Our system-wide power constraint is imposed through the following equation:

$$\forall t : 0 \leq t \leq D_{PTG} \mid \sum_{i=1}^{n} \sum_{k=1}^{k_i} \sum_{l=1}^{L} \sum_{\eta=1}^{m} Z_{ilkt\eta} \cdot Pow_{i,l}^{k} \leq Pow_{BGT} \tag{11}$$

Objective. The objective remains to maximize the overall QoS, and is stated as:

$$maximize \ QoS \tag{12}$$

subject to the constraints presented in Equation 7 to 11, where QoS can be expressed as,

$$QoS = \sum_{i=1}^{n} \sum_{\eta=1}^{m} \sum_{k=1}^{k_i} \sum_{l=1}^{L} \sum_{t=0}^{Te_i^{la}} O_i^k \cdot Z_{iklt\eta} \tag{13}$$

Complexity analysis: The complexity analysis for our ILP is presented in Table 1. The third column of the table illustrates the upper bound of the number of constraints for each equation. For example, in Equation 7, the unique start time should be determined for all n tasks, hence, for a given PTG, overall $O(n)$ constraints will be required. Similarly, the number of variables for this constraint can be represented as $O(K \cdot L \cdot m \cdot D_{PTG})$, where K denotes the maximum number of possible versions of a task. However, as the number of processors (m), and the number of frequency levels (L) are typically constants for a given system, thus the complexity may be considered as $O(K \cdot D_{PTG})$. Similarly, for deadline constraint (see Equation 10), this condition should be checked for a single sink node, and thus, only $O(1)$ constraints will be required. In this way, the total complexity of ILP (in terms of the number of constraints) can be represented as $O(n + D_{PTG})$. Therefore, the amortized complexity becomes $\frac{O(n)}{O(D_{PTG})}$.

Constraint Type	Equation	# Constraints	# Variables Per Constraints
Unique Execution Start Time	Equation 7	$O(n)$	$O(K \cdot D_{PTG})$
Resource	Equation 8	$O(D_{PTG})$	$O(n \cdot K \cdot D_{PTG})$
Execution Dependency	Equation 9	$O(n)$	$O(K \cdot D_{PTG})$
Deadline	Equation 10	$O(1)$	$O(K \cdot D_{PTG})$
Power	Equation 11	$O(D_{PTG})$	$O(K \cdot n)$

Table 1. Complexity of ILP

Example: For ease of understanding of our scheduling strategy, let us consider a representative example with the task-set given in Table 2, which is pictorially represented in Figure 2. These tasks need to be scheduled on two processors ($m = 2$), with a common deadline $D_{PTG} = 100$ time units. Our assumed power budget for both processors is set as $Pow_{BGT} = 25$. Each of these tasks can be executed in two V/F levels, where the frequencies are normalized as 1 and 0.5. The power consumption, as well as the execution lengths for each task (Pow_i) are based on the higher frequency. Note that, the power consumption at the higher frequency is assumed as double that of the lower frequency, whereas the respective execution times (ET_i's) of the tasks (T_i's) in higher frequency are half of the execution length in the lower. As per the proposed ILP formulation, CPLEX [7] generates the schedule illustrated in diagram ① in Figure 3 and in Table 3. Here, T_2 and T_5 are executed in lower V/F setting to satisfy the power constraint. Out of all tasks, T_4 was able to execute its reduced precision version in higher V/F setting. The total obtained QoS value is 38 (see Equation 2). Note that, we have slacks at both processors (P_1 and P_2). At P_1, we have slack at the end of T_4, and at P_2, slacks have been generated both at the beginning and at the end.

3.2 Dynamic Energy-Adaptive Mechanism

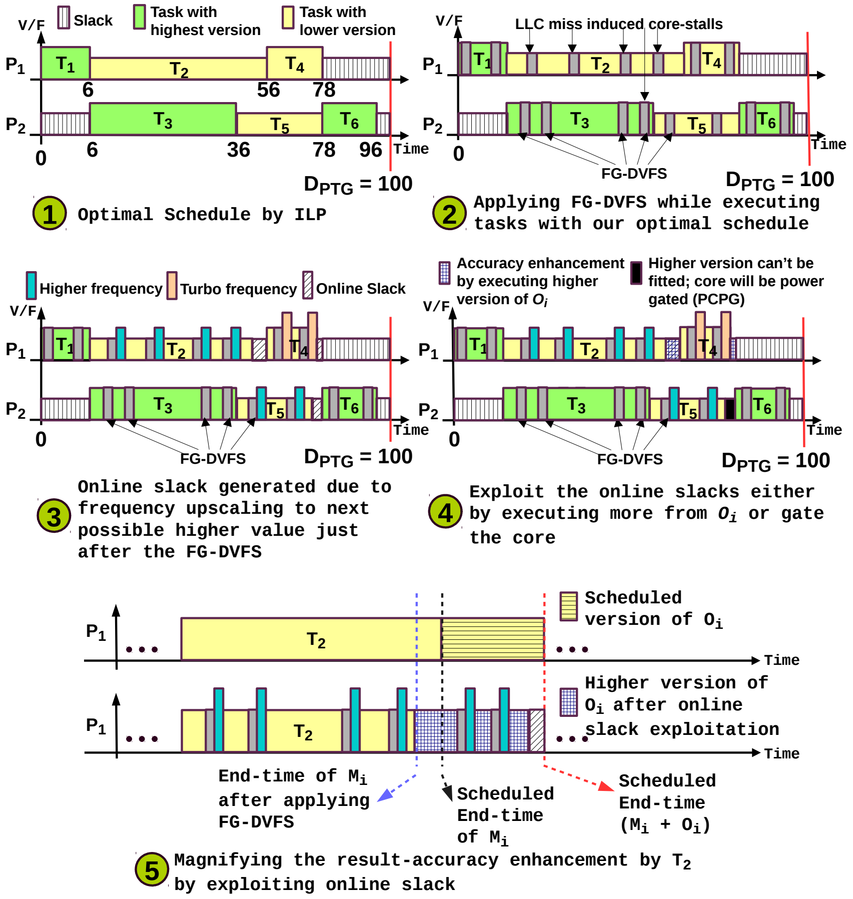
The entire concept of our online energy-adaptive mechanism is illustrated in Figure 3. As per the generated schedule (see diagram ① in Figure 3), each task should be executed at a certain processing speed (V/F setting) on the assigned core. To improve energy and thermal efficiency, this assigned V/F setting can be adapted dynamically, but that may lead to deadline failure, if not managed carefully. *Prepare* employs a fine-grained DVFS (FG-DVFS) based strategy to reduce the temperature and power consumption of the core, by lowering the V/F setting during LLC-miss induced memory stalls (see diagram ② in Figure 3). On completion of the LLC-miss, the V/F setting of the core will be increased to a higher value than previously assigned, which will execute instructions for a stipulated time-span. Basically, we introduce an energy-adaptive mechanism that trades off the power/thermal benefits achieved by applying FG-DVFS in association with LLC stalls and determines the stipulated time-span for which the higher V/F will be maintained (see diagram ③ in Figure 3) to ensure thermal safety. Note that, execution of the tasks at higher V/F

Task	M_i (#cycles)	O_i (#cycles)	Pow_i	Task	M_i (#cycles)	O_i (#cycles)	Pow_i
T_1^1	4	2	10	T_4^1	15	7	15
T_2^1	20	5	20	T_4^2	15	12	
T_2^2	20	7		T_5^1	15	2	20
T_2^3	20	10		T_5^2	15	6	
T_3^1	20	5	10	T_5^3	15	14	
T_3^2	20	8		T_6^1	10	2	10
T_3^3	20	10		T_6^2	10	8	

Table 2. Parameters with values for example task set

Tasks	Mapped Processor	Selected Version	Execute Start Time	O_i	Assigned V/F Level
T_1	P_1	1	0	2	1
T_2	P_1	1	6	5	0.5
T_3	P_2	3	6	10	1
T_4	P_1	1	56	7	1
T_5	P_2	2	36	6	0.5
T_6	P_2	2	78	8	1
Achieved QoS				38	

Table 3. Outputs of the constrained scheduling



V/F
Slack
Task with highest version
Task with lower version
P1
P2
T1
T2
T3
T4
T5
T6
0
6
36
56
78
96
Time
DPTG = 100
1 Optimal Schedule by ILP
LLC miss induced core-stalls
FG-DVFS
2 Applying FG-DVFS while executing tasks with our optimal schedule
Higher frequency
Turbo frequency
Online Slack
3 Online slack generated due to frequency upscaling to next possible higher value just after the FG-DVFS
Accuracy enhancement by executing higher version of Oi
Higher version can't be fitted; core will be power gated (PCPG)
4 Exploit the online slacks either by executing more from Oi or gate the core
Scheduled version of Oi
Higher version of Oi after online slack exploitation
End-time of Mi after applying FG-DVFS
Scheduled End-time of Mi
Scheduled End-time (Mi + Oi)
5 Magnifying the result-accuracy enhancement by T2 by exploiting online slack

Fig. 3. *Prepare*: working mechanism (not to scale).

for a stipulated short time-span is only shown for T_2, T_4 and T_5 in diagram (3) for better readability, however, similar frequency switching is also be applicable for the remaining tasks.

As modern applications spend a significant amount of their total execution time accessing memory [2, 5, 40], such increases in V/F for short time-spans just after each LLC-miss induced stalls will reduce execution time noticeably and will generate computational slacks. *Prepare* exploits slack intervals either to enhance overall result-accuracy by executing higher version of the optional part of the respective tasks, or to power gate the cores to improve power/thermal efficiency (see diagram ④ in Figure 3), or both. Additionally, any slacks generated by the constrained scheduling will also be exploited to enhance thermal and energy efficiency of the system by power gating the cores. The voltage switching and power gating overheads of the processors [3] are tackled by judicious verification of the available slack intervals and the underlying processor model.

In the following portions of this section, we will first discuss the background for applying FG-DVFS during memory stalls (see Sec. 3.2.1, and Figure 4) that motivated us to apply FG-DVFS during LLC-miss induced stalls to enhance energy/thermal efficiency. Subsequently, we will elaborate the viability of FG-DVFS (see Sec. 3.2.2) and plot an initial analysis (in Figure 6) regarding reduction in dynamic energy, offered by FG-DVFS. Finally, we will focus on how the gained benefits can be traded off further in an energy-adaptive manner (see Sec. 3.2.3) before discussing our online algorithm (see Sec. 3.2.4).

3.2.1 Analyzing Memory Accesses. Existing literature reveals that a significant portion of the execution time of modern applications is expended on accessing memory (both data and instruction blocks) [2, 5, 40], and each of such memory accesses is costly in terms of access-time. In the case of an instruction miss in an OoO core, the dispatch of the instructions will be stalled, once the front-end is depleted by instructions. On the other hand, when a load miss (i.e. miss for a data block) takes place, due to memory-level parallelism (MLP), only the very first miss, or an isolated miss, of potentially multiple in-flight memory accesses to the same memory location (i.e. cache block), will observe the full time of the memory access. To take advantage of FG-DVFS at the core, we need to identify which loads cause isolated misses. Such isolated load misses can be identified by looking at the miss status holding register (MSHR) of the respective requester cores. If a load (LLC-)miss does not have an allocated MSHR, and its requester core has no pending memory access at present, then, the miss can be declared as isolated miss. We executed eight PARSEC applications in gem5 [6] for 80M cycles (in Region of Interest (RoI)) on a single core OoO *Alpha 21364* processor, equipped with two levels of caches (64KB 4W L1 (D/I) and 1MB L2) and observed the amount of time (in percentage) the stalls take place, while accessing memory. We segregated the isolated misses and instruction misses for individual benchmark applications and illustrate the results in Figure 4. The result shows that, two memory intensive applications, *Ded* and *Stream*, spend up to 60% of their total execution time in the memory stalls. For all the applications, on an average 25% of the total execution time a core is stalled accessing memory. This is noticeably high and, can therefore be utilized for reducing the power and temperature of the core without impacting its performance.

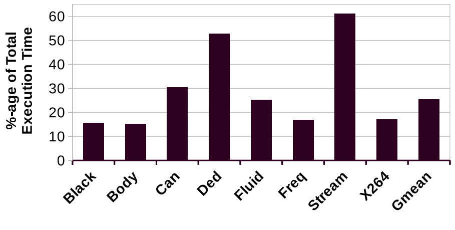


Fig. 4. Amount of total execution time spent on off-chip accesses. *A background analysis.*

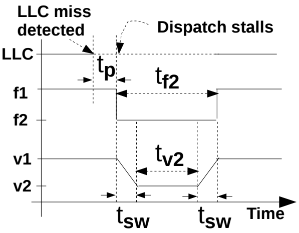


Fig. 5. Timing diagram for FG-DVFS.

3.2.2 Individual stall-span vs. FG-DVFS. Now, we will discuss the scenarios for an instruction miss and an isolated miss for which FG-DVFS can be applied at the requester core. Let us assume that each core dispatches D instructions per cycle. Once an LLC-miss occurs due to the absence of an instruction block, it will require L cycles before the dispatch stops, where L is the front-end pipeline depth of the core. Meanwhile, the off-chip memory is being accessed, and on completion instruction fetching will be resumed. However, the dispatching of instructions will be resumed only L cycles later. Hence, in case of an instruction miss, the stall-span for applying FG-DVFS is equal to the off-chip memory access latency. The FG-DVFS in our energy-adaptive mechanism scales down the V/F on detection of an (instruction) LLC-miss and scales the V/F up on completion of the memory access. The V/F scaling effectively changes the frequency abruptly during the process, which consequently elongates the depletion of the front-end pipeline. But this temporal overhead will not impact the performance as it will be hidden by the off-chip access.

An isolated (load) miss can be identified by exploiting the MSHR of its requester core, however, scaling down V/F on detection of an isolated miss might result into performance degradation. Basically, the dispatch stalls during a load miss, if one of the following situations takes place: *(A)* the Re-Order Buffer (ROB) is filled up, *(B)* the physical registers get exhausted, or *(C)* the issue queue is filled up with all instructions that depend on the currently missed block. As our considered system handles time-critical applications, we wait until the dispatch stalls. On detection of dispatch stall, the V/F will be scaled down, and once the data arrive from the memory, the dispatch resumes after scaling up the V/F. In fact, waiting up to dispatch stalls on occurrence of an LLC-miss, before scaling down V/F, can also tackle the overlapping misses of independent blocks. Waiting for dispatch stalls might reduce the benefits gained by FG-DVFS, but can safeguard our time-critical applications from deadline violation. Diagram ② in Figure 3 shows when FG-DVFS will be applied during the stall cycles. For better readability, in this figure we have indicated (using arrows) that FG-DVFS is applied only at P_2 during T_3 and T_5, however this is applicable for all tasks and for all the processor cores. Upon detecting an isolated or instruction miss, the FG-DVFS controller will scale down the V/F setting of the core (on dispatch stalls for the data misses) and will maintain this lower V/F during the stall-span. Once the stall finishes, the controller will increase the V/F to the (predetermined) assigned level.

Each of the individual stall-span depends upon several timing overheads, that can result into variation in the stall-span for individual LLC-misses. In this paper, we assume uniform memory access time, which is a simplification, as studying variation in off-chip memory access latency prediction is another topic of research [34], out of the scope of this paper. In *Prepare*, we assume a fixed DRAM access latency of 70ns [10], however our FG-DVFS mechanism can also be extended to tackle variable memory delay, without incurring significant changes.

Figure 5 illustrates the (individual) timing diagram, while applying FG-DVFS (see diagram ② in Figure 3). Conventional systems usually incur a time-gap between an LLC-miss detection and dispatch stalls at the requester core, which is denoted as t_p, in this figure. Next, the FG-DVFS controller will reduce the frequency without any delay to a lower level and will concurrently start regulating the voltage (from v1 to v2). Note that, the controller will start scaling up the voltage for a stipulated time-span before completion of the stall, so that, instructions can be executed at the assigned V/F once the stall finishes. Once the voltage is scaled up, the frequency will be set to its respective higher level that incurs no delay.

In our work, we considered the OoO core having a higher V/F setting of 1.12V/3.0GHz and a lower V/F setting of 0.76V/1.2GHz. The considered on-chip VR has a switching speed of 20mV/ns and consumes 0.106W and 0.168W, for 0.76V and 1.12V outputs, respectively [14]. During a stall-span of 70ns, the core (OoO *Alpha*21364 architecture) can consume 337nJ of energy at the V/F setting of 1.12V/3.0GHz, which is our baseline. Depending upon the LLC-miss (i.e. load or instruction miss), the energy usage by a core can be in between 107.1 to 115.4nJ which includes the power consumption of the core and the VR during switching[1]. Thus, the overall energy usage is minimized

[1] The energy usage is calculated at the nano-second precision level by discretizing.

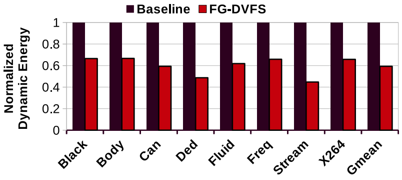


Fig. 6. Reduction in dynamic energy of the core by FG-DVFS. *A background analysis.*

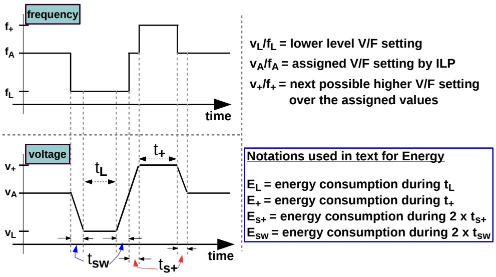

frequency
f+
fA
fL
time
vL/fL = lower level V/F setting
vA/fA = assigned V/F setting by ILP
v+/f+ = next possible higher V/F setting over the assigned values
voltage
v+
vA
vL
tL
t+
tsw
ts+
time
Notations used in text for Energy
EL = energy consumption during tL
E+ = energy consumption during t+
ES+ = energy consumption during 2 x tS+
ESW = energy consumption during 2 x tSW

Fig. 7. Enhancing performance by exploiting gains of FG-DVFS. An extension of Figure 5.

by up to 68%, which is significant. Note that, we analyzed the configuration of our considered core (see Sec. 4.2.1) and derived the maximum time taken before the dispatch stalls from the detection of an apt LLC-miss to be around 8ns (i.e. t_p in Figure 5). We further illustrate the overall reduction in dynamic energy of the core gained by FG-DVFS in Figure 6 by executing eight PARSEC applications for 80M cycles (in RoI) in gem5. The figure reports an average reduction of 40% (up to 54%) in dynamic energy of the core, which is significant. In case of the core, dynamic energy shares a significant amount of the total energy consumption [26], hence, this noticeable reduction in dynamic energy potentially motivates us to employ FG-DVFS towards maintaining a safe thermal status. We derived all of these values by simulating the architecture in our simulation setup, with the configuration details discussed in Sec. 4.2 (see Table 5).

3.2.3 Trading off Gains by FG-DVFS. Employing FG-DVFS leads to significant reduction in energy, which helps the core to execute below the assumed power budget, without impacting the performance. Hence, we further planned to trade this energy savings to improve result-accuracy and/or thermal/energy efficiency, while remaining in the power budget. Once the stall finishes, FG-DVFS scales up the V/F to the assigned level (see diagram (2) in Figure 3). We will extend this V/F scaling up process, by setting V/F to the next available higher level and the core will execute the instructions at this higher level for a particular time-span. However, execution at this higher V/F than the assigned one will be allowed for a stipulated time-span, which is set by considering the amount of energy saved during the memory stall. The saved energy will be traded off at the higher V/F level, so that the system will not violate its power budget. The stipulated time-span for this higher V/F will be sufficiently small ($< 70ns$), to reduce the chance of thermal overshoot [26]. However, such short time-spans for which tasks are executed at a higher speed after each stall will altogether result into early completion of the individual task, with respect to the assigned schedule generated by the constrained scheduling. Thus, a portion of spare time, i.e. slack, will be generated at the end of each task as shown in diagram (3) in Figure 3. Now, this slack interval can be exploited further to enhance result-accuracy or to improve thermal/energy benefits depicted in diagram (4) in Figure 3. We illustrate the change in result-accuracy of T_2 in diagram (5) of Figure 3. By applying proposed V/F scaling mechanism, M_i will finish early creating more room for enhancing result-accuracy, by executing higher version of O_i, based on its availability. However, as V/F scaling will be applied in the same fashion during execution of O_i as well, it might leave a slack before its scheduled end-time (as shown in Figure 3).

Now, we will illustrate the whole idea of exploitation of the gains of running the core at lower V/F during stalls with an example shown in Figure 7 (an extended version of Figure 5 towards exploitation of energy saved by FG-DVFS). Basically, Figure 7 represents the timing diagram during an individual miss and the frequency switching on miss-revolve (diagram (3) in Figure 3). We have assumed our baseline V/F setting as, v_A/f_A which has been assigned by our constrained scheduling. During a stall induced by LLC-miss, the V/F setting of the core will be stepped down to v_L/f_L. While scaling up the V/F on completion of a stall interval, the V/F will be set to v_+/f_+ (higher than v_A/f_A). The respective energy consumption while staying at A, L and + levels are E_A, E_L and E_+ (see Figure 7). Note that, E_A represents the energy consumption without any changes to the V/F, hence, is considered as baseline energy consumption for the duration of $2 \cdot t_{sw} + t_L$ time unit. The energy usages by the switching processes are E_{sw} and E_{s+} during switching between A and L levels and A and + levels, respectively (see Figure 7). As violating power budgets might result in severe thermal issues, the energy saved by FG-DVFS over the baseline (i.e., $E_{saved} = E_A - (2 \cdot E_{sw} + E_L)$) during $2 \cdot t_{sw} + t_L$ should be the upper limit, while running the core at the + level. Now, to be in power budget or to maintain on-chip thermal safety, $E_{saved} \geq E_+ + 2 \cdot E_{s+}$. As the power consumption at v_+/f_+ level is known, the time-span (t_+) at which the core will be operating at this higher frequency (f_+) can be determined on-the-fly. We will now use the values for the parameters for energy computation and will summarize the same in Table 4, to show the efficacy of our policy. According to the values mentioned in the table, FG-DVFS saves 68.2% during stall interval (i.e. $2 \cdot t_{sw} + t_L$). After reaching A from L, our system will now scale up its V/F to + level. By considering the switching energy (E_{s+}) together with the core energy usage, we derived that, for each such stall interval, the maximum improvement in performance will be as high as 17.3%. The overall performance improvement and generation of slacks along with its exploitation towards improving result-accuracy or power/thermal efficiency will be discussed in details in Sec. 4.

Parameters	Values	Parameters	Values	Parameters	Values	Parameters	Values
v_A/f_A	$1.12v/3.0GHz$	t_L	34ns	t_+	32ns	E_{sw}	53.8nJ
v_L/f_L	$0.76v/1.2GHz$	t_{sw}	18ns	E_L	30.2nJ	E_{s+}	31.3nJ
v_+/f_+	$1.17v/3.6GHz$	t_{s+}	3ns	E_+	200nJ	E_A	337nJ

Table 4. Values used and obtained in our example.

3.2.4 *Online Management.* Figure 3 (diagram ③ and ④) illustrates the entire online process for applying FG-DVFS at the LLC-miss induced stall cycles. The figure also depicts how the online algorithm attempts to stimulate either the result-accuracy/thermal/energy efficiency based on the available scope. On the availability of a higher version optional portion of the current task that can be executed without violating the real-time constraints, the algorithm will execute this higher version of O_i to improve the result-accuracy. On the other hand, while exploiting online slacks for enhancing thermal or energy efficiency, our algorithm power gates the cores during the slack. To meet the timing constraint, our algorithm further initiates the turning on process of the core sufficiently earlier (by considering the turn-on overhead of the core), so that it can start executing the task on its scheduled arrival.

We present the process of online energy adaptive result-accuracy and energy efficiency enhancement at the individual cores in Algorithm 1 and 2. Our online mechanism considers the assigned frequency of each task derived from our constrained scheduling and converts all the timing parameters in cycles, that enables us to keep track of the time by increasing a counter, *global_cycles*. *Power_gate_overhead*, *Min_Slack* and *Block_cycles* are the inputs to the Algorithm 1, that represent the time taken for the core to be turned on from its gated status, a minimum threshold value for the slack-span (also known as processor break-even time [17])[2], and number of cycles a core might be blocked at the beginning due to completion of the execution of the source task (for instance, a slack at the beginning at P_2 in Diagram ① in Figure 3), respectively. The length

[2] the value of which is based on the architecture of the underlying core

Algorithm 1: Per core run-time power/thermal management within a *FRAME*

```
   Input: $Power\_gate\_overhead, Min\_Slack, Block\_cycles, dispatch\_table, Slack\_End\_Cycles$
1  # Check the $dispatch\_table$ if $Block\_cycles$ exists for the current core at the beginning of the $FRAME$
2  # (due to execution of the source task at some other core)
3  if $Block\_cycles \geq (Min\_Slack + Power\_gate\_overhead)$ then
4      # Power gate the core for $gated\_cycles$
5      $gated\_cycles = Block\_cycles - Power\_gate\_overhead$
6      $global\_cycles$ + = Power-Gate($gated\_cycles$)
7  else
8      $global\_cycles = 0$
9  # Apply energy-adaptive results-accuracy enhancements for each task
10 for each task ($T_i$) assigned to this core do
11     Fetch $T_i$ and set its predetermined V/F setting
12     $global\_cycles$ + = Call Algorithm 2
13     # Check if slack exists by looking at the $dispatch\_table$, and get the end cycles of the slack
14     if $(Slack\_End\_Cycles - global\_cycles) \geq (Min\_Slack + Power\_gate\_overhead))$ then
15         # Power gate the core during the slack
16         $gated\_cycles = Slack\_End\_Cycles - global\_cycles - Power\_gate\_overhead$
17         $global\_cycles$ + = Power-Gate($gated\_cycles$)
18 Function Power-Gate($gated\_cycles$):
19     $update\_cycle = gated\_cycles + Power\_gate\_overhead$
20     Apply power gating at the core
21     while $gated\_cycles > 0$ do
22         $gated\_cycles$--
23     Turn on the core
24     return $update\_cycle$
```

of the time-span between 0 to D_{PTG} is now defined as *FRAME* in our online mechanism and is represented in *cycles*. For ease of understanding, the 100 time-units of Figure 3 can be assumed as 100 cycles. For the current core, our online algorithm also considers the scheduled start cycles for the individual tasks (T_i) through the dispatch table [28], an input to the algorithm ($dispatch\_table$), in which the tasks are ordered according to their execution start cycle instant, as obtained from the outputs of our constrained scheduling. The $dispatch\_table$ also contains the timing details (in cycles) of the slacks. For each slack entry in the $dispatch\_table$, our algorithm accesses the end time of the slack as input, denoted by $Slack\_End\_Cycles$.

While within a *FRAME*, the algorithm checks if any $Block\_cycles$ for the current core exists and has a value larger than the sum of Min_Slack and $Power\_gate\_overhead$ (see line 3). On detection of a such sufficiently large $Block\_cycles$, the core will be power-gated for a stipulated number of cycles by calling *Power-Gate(gated_cycles)* function (see line 4 to 6). However, the span of this gated mode is determined by considering the $Power\_gate\_overhead$ of the individual cores (see line 5). This implies that, the turn-on process within *Power-Gate(gated_cycles)* function (see line 6) for the core will be initiated $Power\_gate\_overhead$ cycles before the end of the slack or the $Block\_cycles$, so that timing constraint can be met. Based upon the presence of $Block\_cycles$, our cycle counter $global\_cycles$ will be updated accordingly (line 6 or 8).

At runtime, *Prepare*:online traverses the dispatch table and subsequently fetches each of the tasks to execute as per their start time (in cycle) (see line 10 to 12). On a task start cycle instant, the algorithm will set the predetermined V/F setting for the task and will start the execution. Once the task execution has been started, the algorithm seeks for the stalls induced by long latency LLC-misses, so that FG-DVFS can be applied. To apply FG-DVFS, Algorithm 2 will be called (see line 12), which will attempt to improve result-accuracy by the energy-adaptive mechanism, which will be detailed hereafter. However, with the availability of a sufficiently large slack, our online algorithm also enables power gating at the cores (see line 14 to 17). If the core is not gated, *Power-Gate()* function (see line 18 to 24) will power gate the respective core and will keep it gated for a stipulated

Algorithm 2: Energy-adaptive result-accuracy enhancement for a single task

```
   Input: $DVFS\_PERIOD$, $TURBO$, $DVFS\_Overall\_at\_Assigned$, $LowerVF$, $TurboVF$, $AssignedVF$, $Task\_Cycles$
   Output: $g\_cycles$
1  $scaled\_down$ = 0, $scaled\_up$ = 0, $DVFS\_count$ = 0, $g\_cycles$ = 0
2  while $T_i$ is being executed do
3      $g\_cycles++$
4      if ($dvfs\_enabled$ = 0) and (LLC-miss is detected for Address) then
5          if (dispatch stalls on a data miss) or (an instruction miss is detected) then
6              Apply_DVFS($LowerVF$) and stop increasing $g\_cycles$
7              $scale\_down$ = 1
8              $dvfs\_enabled$ = 1
9              # Set counter to the duration of the reduced V/F setting (i.e. $LowerVF$)
10             $counter$ = $DVFS\_PERIOD$
11         else
12             # Block is already being handled by an earlier request, so, execute as normal
13     if ($counter$ == 0) and ($scale\_down$ == 1) then
14         Apply_DVFS($TurboVF$) and stop increasing $g\_cycles$
15         $scaled\_down$ = 0
16         $scaled\_up$ = 1
17         # Set counter to the duration of the increased V/F setting (i.e. $TurboVF$)
18         $counter$ = $TURBO$
19     if (counter == 0) and (scaled_up == 1) then
20         Apply_DVFS($AssignedVF$)
21         $dvfs\_enabled$ = 0
22         $scaled\_up$ = 0
23         # Since DVFS (scaled down and up) is applied for predetermined lengths,
24         # a fixed number of cycles can be added to $g\_cycles$ that accounts for the frequency difference
25         $g\_cycles$ + = $DVFS\_Overall\_at\_Assigned$
26     # $counter$ is an unsigned saturating decrementer
27     $counter$--
28     # Upon finishing the mandatory part, check if sufficient number of cycles exist
29     # such that a higher precision can be calculated
30     if execution of mandatory part ($M_i$) of $T_i$ is finished and execution of optional part ($O_i$) is not yet started then
31         # Calculate the remaining cycles allocated for the task
32         $cycles\_left$ = $Task\_Cycles$ - $g\_cycles$
33         # Call function that returns optional part with highest possible precision that can run within the provided cycles
34         $optional\_part$ = $get\_optional\_part(T_i, cycles\_left)$
35         if $optional\_part$ then
36             Fetch the optional part of $T_i$
37 return $g\_cycles$
```

period (*gated_cycles*), that depends on the available slacks at the end of the executions of the individual tasks (see line 17). Note that, the core will be turned on and will be ready at the starting time instant (in cycle) of the next task. Basically, this function monitors *power gating* and *turn on* processes by considering the respective temporal parameters. After completion of the turn-on process, the value of the *global_cycles* is updated.

Algorithm 2 keeps track of the cycles (see line 3) while the tasks are executed and returns the cycle count as output (*g_cycles*). This counter is increased at every clock cycle during task execution and keeps track of the total number of cycles within a *FRAME*. However, dynamically scaling the frequency changes the cycle-span and that might violate the overall schedule. To address this issue, we introduced the following inputs to Algorithm 2: *DVFS_PERIOD*, *TURBO*, and *DVFS_Overall_at_Assigned*. The first two inputs are the respective number of cycles the core will execute during *DVFS with Lower* and *Turbo* V/F settings, respectively. The last one is a derived value, which is a sum of the number of cycles that can be completed at the assigned frequency during the same time-span of *DVFS_PERIOD* and the actual cycles executed during *TURBO* at the Turbo V/F setting[3]. Note that, for every task, we will have the following three levels of V/F

[3] Due to our assumption of a fixed memory delay, *DVFS_Overall_at_Assigned* is constant and can be computed offline.

settings which are also inputs to Algorithm 2: $LowerVF$, $TurboVF$, and $AssignedVF$. $Task\_cycles$ (for each T_i) is another input to the algorithm, which is the starting time-stamp (in cycles) of the next event, which can either: *(i)* be the starting of the execution of the next task, or *(ii)* be the end of the *FRAME*. This value is derived for the individual tasks by looking at the dispatch table entries.

On detection of an LLC-miss, the core will wait until dispatch stalls in case of a data miss[4], whereas an instruction LLC-miss is sufficient for applying FG-DVFS (see line 4 to 10). Once an apt miss is detected or the dispatch stalls, the $Apply\_DVFS()$ function[5] will be called with $LowerVF$ (line 6), else the execution proceeds as normal (see line 12). A stipulated duration for which core will maintain $LowerVF$ is managed with a $counter$ by initialized it to $DVFS\_PERIOD$ (see line 10). Once this $counter$ reaches zero for a DVFS enabled core (see line 15), it scales up the V/F setting (see line 14) to serve the outstanding instructions at the $TurboVF$ by calling $Apply\_DVFS()$ for the current task on completion of the off-chip memory access. The time-limit for maintaining $TurboVF$ is determined by initializing $counter$ to $TURBO$ (see line 18). Note that, our algorithm stops increasing the cycle counter (g_cycles) if the core is not being operated at the $AssignedVF$ (see line 6 and 14), and on completion of each such V/F scaling operations (including both $LowerVF$ and $TurboVF$) g_cycles will be updated with $DVFS\_Overall\_at\_Assigned$ (see line 25).

Algorithm 2 also monitors if the execution of M_i is over and the execution of O_i has not been started (see line 30). Due to online energy-adaptive mechanism, M_i will be completed early, which can now be exploited for enhancing the result-accuracy. At this point, Algorithm 2 computes the cycles left ($cycles\_left$) for a task T_i (see line 32). Next, to update the optional portion with higher precision, the algorithm will call $get\_optional\_part()$ with the following inputs: T_i, and $cycles\_left$ (see line 34). Based upon the already scheduled O_i, this function starts searching for the highest possible precision version of O_i which can be executed within $cycles\_left$. At the end of the searching process, the optional part is updated with the best possible version, and Algorithm 2 fetches the O_i for execution (see line 35 to 36).

3.2.5 Hardware Mechanism. Our energy-adaptive online strategy employs FG-DVFS of the cores during long memory stalls. Through selective detection of an instruction miss or an isolated load miss by inspecting MSHR entries, our FG-DVFS mechanism scales down the V/F settings of the individual cores at the commencement of the stall-span. Additionally, cores are power gated on detection of a sufficiently large slack. Towards implementation, the counters (used in the algorithms) have to be implemented to keep track of the time-spans, so that a core can start operating at its predetermined processing speed at the proper time, which guarantees that the real-time constraints are met. The monitoring logic for FG-DVFS has to be implemented at the respective cores, that will govern the associated VR to scale V/F setting of the core on LLC-miss induced stalls. As modern CMPs are usually equipped with such VRs and implementation of the above-mentioned counters are straightforward in practice, the online mechanism of *Prepare* incurs limited hardware implementation cost [14]. The presence of such VR on-chip might incur their own power and thermal overheads, which can be addressed by employing techniques like ThermoGater [25]. Additionally, overheads related to overclocking which is incorporated during $Turbo$ V/F can be tackled by adopting recently proposed mechanisms [23, 42], however, a detailed discussion of overclocking is out of scope of this paper.

[4] Determination of such misses will be complicated in case of multi-cores where cores have shared LLC blocks, which is out of scope of this paper. *Prepare* assumes all threads of a particular task/application are running on the same core.
[5] $Apply\_DVFS()$ is a system's function that handles the V/F scaling process at the individual cores.

4 EVALUATION

In this section, first we show the efficacy of our constrained scheduling approach (see Sec. 4.1) followed by the evaluation of runtime energy-adaptive strategy (see Sec. 4.2).

4.1 *Prepare*: Constrained Scheduling

For evaluation, we define **N**ormalized **A**chieved **Q**oS (NAQ), as the ratio between the actually achieved QoS (see Equation 2) for the PTG, and the maximum achievable QoS by executing the highest version of each task. Mathematically, NAQ can be formulated as:

$$NAQ = \frac{\sum_{i=1}^{n} Acc_i^j}{\sum_{i=1}^{n} Acc_i^{k_i}}, \tag{14}$$

where k_i denotes the highest version of task T_i. Now, we model a multi-core system and our task-set:

- *Processor System:* For the purpose of our experiment, a homogeneous multi-core platform equipped with 4 Alpha 21364 cores (i.e., $m = 4$) has been considered. The per core Pow_{BGT} (i.e. maximum) is set at 4.0W [29], which is obtained through power-profiling for individual task in McPAT [29].
- *Task Characteristic:* Each PTG consists of a set of subtasks (aka nodes) under precedence constraints and has a deadline D_{PTG}. Each subtask (T_i) is a multi-threaded task (see Table 6), where all threads of a task are executed on the same core (in a quasi-parallel manner), characterized by execution times, ET_i. We assumed that a subtask can consume between 4×10^7 and 6×10^8 clock cycles [33] and they can have a maximum of 5 versions, i.e. $k = 5$. The assumptions regarding execution lengths also include memory cycles for our individual task consisting of PARSEC benchmark applications [5] (see Figure 4). The total execution requirement of a PTG (C_{PTG}) is defined as the sum of the execution times of its subtasks, $C_{PTG} = \sum_{i=1}^{n} ET_i$. Thus, utilization U_i of a PTG can be denoted as $\frac{C_{PTG}}{D_{PTG}}$. The average utilization of a PTG has been taken from a normal distribution, by considering normalized frequency 0.6. Given the PTG's utilization, we can obtain the total utilization of the system (Sys_{uti}) by summing up the utilization of all the PTGs. Given the system utilization, the total system workload (Sys_{WL}) / system pressure can be derived by: $Sys_{WL} = \frac{Sys_{uti}}{m} \times 100\%$. For a given system utilization, all the PTGs have been generated by following the method proposed by Qamhieh and Midonnet [37]. Given a Sys_{WL}, we have created a set of DAGs. The number of DAGs (ρ) within a set can be measured as:

$$\rho = \frac{m \times Sys_{WL}}{U_i} \tag{15}$$

  In the generated PTGs, the minimum number of tasks (nodes) is equal to 5 and maximum number of nodes is set to 20. For each PTG in the set, the number of nodes have been randomly generated within the given limit. It may be noted that as the individual utilization (U_i) of a DAG is lower than the given system workload (Sys_{WL}), the number of DAGs (ρ) within the set will always be higher than m. All experiments are carried out using the CPLEX optimizer version 12.10.0, with a timeout of 5 hours.
- *Task Temporal Parameters:* For each task T_i, based on which portion of the len_i is considered as its mandatory portion (M_i), we consider the following cases: *(i) man_low* : $M_i \sim U(0.2, 0.4) \times len_i$ (low portion of a task T_i's length (len_i) is for the mandatory portion). *(ii) man_med* : $M_i \sim U(0.4, 0.6) \times len_i$ (medium portion of a task T_i's length (len_i) is for the mandatory portion). *(iii) man_high* : $M_i \sim U(0.6, 0.8) \times len_i$ (high portion of a task T_i's length (len_i) is for the mandatory portion).

- *Frequency Level:* We have chosen two distinct normalized frequency levels as: $f_{norm} = 0.6$ and 1 for task execution; the least possible normalized frequency is 0.3. The respective actual V/F settings for our considered core are given in Table 5.

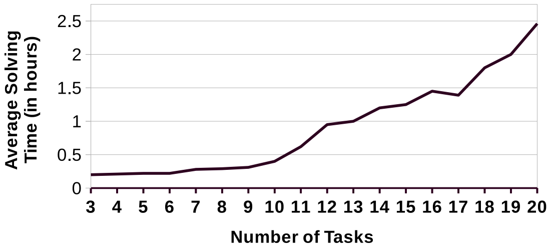
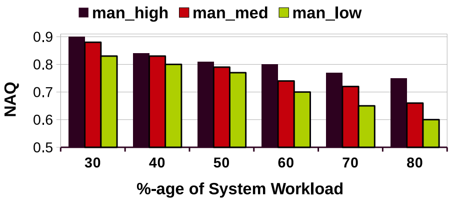


Fig. 8. Analysis of running time of ILP-formulation.

Scalability analysis of ILP. Figure 8 shows the average solving time per number of tasks (nodes) in each graph. It can be observed that, when the number of tasks in each graph is within 10, the average solving time remains comparable. However, when the number of tasks grows beyond that value, the average solving time also rises. This trend can be supported by the complexity analysis provided in Table 1. As discussed, the number of constraints mainly depends on n, i.e. the number of tasks. With $n = 20$, the ILP generates on average 5000 constraints and the solving time reaches up to 150 minutes.



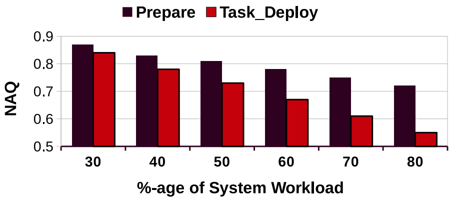


Fig. 9. Change in NAQ for various system workloads.

Fig. 10. Comparing NAQ: *Prepare* vs. *Task_Deploy*.

Effects of System Workload. Figure 9 depicts the NAQ achieved by *Prepare* for various values of Sys_{WL}. The NAQ is derived by running each DAG that belongs to the set. Then, we have taken the average over these obtained individual NAQ values. It can be observed that *Prepare* is able to achieve 90% QoS, when the system workload is low. However, the QoS is reduced by 15% on average, when the workload increases by 40%. Other two insightful observations can be derived from this figure. Firstly, as the system workload increases in order to maintain the number of DAGs (ρ) in the system, the individual utilization (U_i) also increases and this eventually contributes to low NAQ values. This happens due to the fact that increasing U_i results in higher execution length of individual task (nodes) and thus the possibility of obtaining sufficient free slots in the scheduling period reduces within the deadline. Insufficient free slots in turn reduces the probability of obtaining feasible schedules by selecting higher versions of the tasks.

Secondly, in case of man_{high}, it imposes a less adverse effect on the achieved NAQ with the increasing value of Sys_{WL}. This can be attributed to the fact that, when the mandatory portions of an individual task are high, the length of the optional portions will be low. As a result, the variance among the different versions of a task become less. However, due to fewer variations among the optional portions of a task, there will be less impact on the achieved accuracy. On the other hand, in case of man_{low}, we can observe the alternative trend, and man_{med} offers a performance between man_{high} and man_{low}. However, the *NAQ* sharply decreases while Sys_{WL} increases. We have also compared our strategy with a prior strategy ($Task\_Deploy$) [33] and the results are shown in Figure 10. Towards a fair comparison with $Task\_Deploy$, first we computed the overall energy limit based on the considered power budget (Pow_{BGT}) of the experimental framework of *Prepare*. We used this as energy limit, while experimenting with $Task\_Deploy$. Next, we consider our comparison in a case where M_i of tasks is chosen uniformly from 20% to 80% of len_i. It can be observed, as the execution demand of individual task nodes increases (due to increase in Sys_{WL}), *Prepare* maintains improved QoS by achieving higher NAQ than $Task\_Deploy$. *Prepare* is able to maintain 75% QoS at 70% workload where $Task\_Deploy$ achieves 60% QoS. This is because the considered overall energy limit in $Task\_Deploy$ would increase with the higher Sys_{WL}. Moreover, $Task\_Deploy$ also allows unlimited tasks migration, which incurs additional overhead.

4.2 *Prepare*: Runtime Energy-Adaptive Mechanism

After showing the efficacy of our constrained scheduling, we will now discuss the online energy-adaptive result-accuracy enhancement and power saving technique. We will first discuss our simulation framework, followed by the benchmark applications used in our simulation. The empirical results will show the changes in execution time by applying energy-adaptive FG-DVFS that resembles diagram ③ in Figure 3. Next, we will show the impacts of slack-exploitation towards enhancing result-accuracy, and thermal and energy efficiency (diagram ④ in Figure 3). We further studied the effects on peak temperature for an individual task during execution, as tasks are also executed with a V/F setting higher than the originally assigned. Finally, we will compare energy-adaptive strategy of *Prepare* with prior techniques.

4.2.1 Simulation Framework. We simulated a homogeneous tiled CMP having 4 tiles in the gem5 full system simulator [6]. Each tile contains an Alpha 21364 OoO core along with its private L1 data and instruction caches. The L2 cache is logically shared, yet physically distributed among the tiles, where each tile contains an L2-bank of the same size. We further use McPAT [29] and HotSpot [45] to generate power consumption and temperature from the performance traces generated by gem5 in the following manner. We collect periodic performance traces from gem5 and send them to McPAT to generate the power traces. Basically, the dynamic power consumption is derived for individual on-chip components by executing McPAT. As, the leakage power estimation in McPAT assumes uniform on-chip temperature, which is impractical, we compute component-wise leakage power by considering the temperatures of individual on-chip components at the end of the last period [20–22]. Finally, we derive total power consumption from dynamic and leakage power estimations, which are sent to HotSpot 6.0 towards generating temperature traces. Based on prior analysis [9, 10], the span of this periodic interval is set to 0.33 μs, during which the temperature across the CMP is assumed stable. The HotFloorPlan module of HotSpot 6.0 generates floorplan of the CMP once at the beginning by considering the component-wise area estimation from McPAT. We consider 32nm technology nodes to estimate power consumption and area details. The detailed system parameters used in our simulation are listed in Table 5.

Each core operates in two basic *Assigned* frequency levels (*Assigned*_1 and *Assigned*_2, where *Assigned*_1 > *Assigned*_2), and can also execute tasks in $Turbo$ mode. The respective values for V/F settings are given in Table 5. While reducing V/F for improving power efficiency during the LLC miss induced stalls, the core will always operate at *Lower* V/F setting. If the assigned frequency

Parameter	**Value**	**Parameter**	**Value**
ISA	Alpha 21364	L1-I	64KB, 4Way, 3CC
Execution Units	2 int/br., 1 mul, 1 fp, 1 ld/st	L1-D	64KB, 4Way, 3CC
Max. V/F ($Assigned\_1$)	1.12V, 3.0GHz	L2	1MB, 16Way, 12CC
Med. V/F ($Assigned\_2$)	1.02V, 1.8GHz	MSHRs	8
Min. V/F ($Lower$)	0.76V, 1.2GHz	Cache	LRU, 64B blocks
Turbo V/F ($Turbo$)	1.17V, 3.6GHz	#Cache-Levels	2
VR-Speed	20 mV/ns	Cache model	SNUCA
Power_gate_overhead	60 ns	DRAM latency	70 ns
ROB Size	200	Technology	32 nm
Dispatch/Issue width	8	Ambient Temp.	47 °C

Table 5. System parameters [CC: clock cycle]

of a task is $Assigned\_1$ ($Assigned\_2$), it will be executing tasks at $Turbo$ ($Assigned\_1$) speed for a while immediately after the stall-span, for which FG-DVFS (for $Lower$) was viable.

4.2.2 Descriptions of the Task-set. Our tasks are generated by using PARSEC benchmark suite [5], which can be fitted in an AC based paradigm [2, 41]. In their work, Sidiroglou et al. have shown how PARSEC benchmark can be fitted in the approximation paradigm through the loop perforation technique [41]. Based on these prior studies, we framed our task set by defining each task with a couple of PARSEC applications, where the former one is executed as M_i of the respective task and the latter is representing O_i. For creating multiple versions of O_i, the latter application will have different executable files, with different execution lengths. Note that, for each M_i and O_i, we have 2 copies of a particular PARSEC application, that run in parallel on 2 different cores of the CMP[6]. The details about the task-set is given in Table 6, where the execution lengths (Exec_Length) are given in million cycles in the RoI for the respective M_i's and O_i's. For example, while running T_2 with its first version of O_i (having a length of 100M cycles), 2 copies of $Stream$ will be executed for 400M cycles concurrently on 2 cores to complete M_i, and after that, to complete O_i, 2 copies of Can will be executed concurrently on the same set of cores. The table also details the V/F settings that have been assigned for the individual task, by the constrained scheduling along with the assigned versions [Exec_Lengths] of O_i's. To keep implementation of the isolated miss detection simple in Algorithm 2, we assumed that, all threads of a single PARSEC application will be executed on the same core, i.e. the cores do not have any shared cache blocks. Note that, execution length of each task in Table 6 is set by scaling the task lengths given in Table 2 and the outputs of the constrained scheduling (see Table 3) are also set accordingly.

Tasks	**Benchmarks (M_i, O_i)**	**Exec_Length ([M_i], [O_i])**	**V/F Setting**	**Version(Oi) [Exec_Length]**
T_1	*Black (2 copies), Body (2 copies)*	[80], [40]	$Assigned\_1$	1[40]
T_2	*Stream (2 copies), Can (2 copies)*	[400], [100, 140, 200]	$Assigned\_2$	1[100]
T_3	*Ded (2 copies), Fluid (2 copies)*	[400], [100, 160, 200]	$Assigned\_1$	3[200]
T_4	*Fluid (2 copies), Freq (2 copies)*	[300], [140, 240]	$Assigned\_1$	1[140]
T_5	*Body (2 copies), X264 (2 copies)*	[300], [40, 120, 280]	$Assigned\_2$	2[120]
T_6	*X264 (2 copies), Ded (2 copies)*	[200], [40, 160]	$Assigned\_1$	2[160]

Table 6. Tasks formation with PARSEC. (Acronyms: Blackscholes (*Black*), Bodytrack (*Body*), Canneal (*Can*), Dedup (*Ded*), Fluidanimate (*Fluid*), Freqmine (*Freq*), Streamcluster (*Stream*), and X264 (*X264*)). (The execution lengths are in million cycles.)

[6] Effectively, 2 cores of the CMP represent a single core, i.e., a P_i in Figure 3.

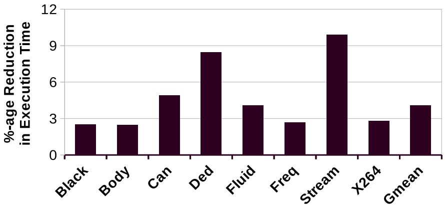


Fig. 11. Reduction in execution time for PARSEC.

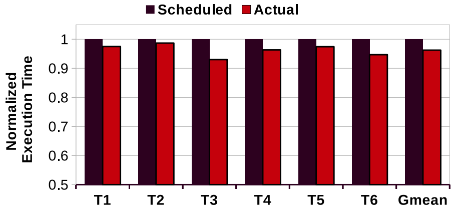


Fig. 12. Reduction in execution time for the tasks.

4.2.3 Effects on Execution Time. We experimented with individual PARSEC applications at first to see the effects on execution time, while applying Algorithm 1. The memory intensive applications, like *Ded* and *Stream*, offer more scope to apply FG-DVFS, due to comparatively higher number of LLC-misses. On the other hand, mixed applications, like *Can*, *Fluid*, etc., experience a lesser number of LLC-misses leading to comparatively lesser chances for applying FG-DVFS. The spike in V/F just after the LLC-miss induced stalls enhances the performance for almost all benchmark applications, reducing execution length by 2.5% to 10% with an average of 3.8%. The respective reduction in execution lengths for eight PARSEC applications are shown in Figure 11, generated by running applications for 200M cycles continuously within RoI. The respective reductions in tasks' execution-lengths are shown in Figure 12, where all the benchmarks of the individual task continuously executed in RoI for the specified number of cycles as per Table 6. The presence of memory intensive applications in case of T_2, T_3 and T_6 results in more reduction in execution lengths, and we were further able to execute higher version of T_2 towards improving result-accuracy (NAQ). By applying Algorithm 1 on the schedule generated by our constrained scheduling, we achieved 5.3% overall improvement in NAQ, whereas a significant amount of scheduled execution lengths (up to 7.0%) can be used as slacks at the end of each individual task. The details regarding scheduled and dynamically updated task versions for T_1 to T_6 are reported in Table 7, in addition with the respective slack details and overall improvement in achieved NAQ.

Tasks	Mapped Core (V/F)	Scheduled Version (ILP)	Updated Version (Online)	Amount of Slack
T_1	$P_1(Assigned\_1)$	1	1	2.5%
T_2	$P_2(Assigned\_2)$	1	2	1.3%
T_3	$P_2(Assigned\_1)$	3	3	7.0%
T_4	$P_1(Assigned\_1)$	1	1	3.7%
T_5	$P_2(Assigned\_2)$	2	2	2.6%
T_6	$P_2(Assigned\_1)$	2	2	5.3%
Improvement in Achieved NAQ			5.3%	

Table 7. Outputs of the constrained scheduling and online updates

4.2.4 Runtime Peak Temperature. The spike in V/F setting (while applying Algorithm 1) at the individual cores might result into temperature overshoot. To observe the effects on runtime core temperature, we simulated the entire mechanism in our simulation setup and report the runtime peak temperature in Figure 13 for our task-set. T_3 and T_5 experience lesser peak temperature for executing in *Assigned*_2 V/F, while the remainder are executed at *Assigned*_1. However, changes in peak temperature is hardly noticed in Figure 13 for all tasks, while applying Algorithm 1. As the spike in V/F setting is taken place for a significantly smaller time-duration and just after running the core in a lower V/F for a stipulated duration of a stall period, the change in peak temperature is not noticeable. Thus, *Prepare*:online claims thermal safety as stated in our contribution (see Sec. 1). However, the reduction in peak temperature is significant in case of *Prepare* while considering the thermal gains during slacks, shown next while comparing it with prior mechanisms.

4.2.5 Comparison with prior work. Our Algorithm 1 gates the cores during the slacks, which significantly reduces peak temperature. To show the efficacy of *Prepare*, we further implemented two prior DVFS based techniques, *GDP* [1] and *Integrate* [13], for our task-set, where in case of *GDP* a different threshold peak temperature is assumed for individual task towards applying DVFS. For T_1 and T_2, the threshold is set as 95 °C, which is 90 °C for T_4 and T_6. We set the threshold as 80 °C for T_3 and T_5. Note that, we allow *GDP* to execute tasks at the lowest possible frequency (Min. V/F in Table 5) and the respective task-wise viable frequency level is set by our scheduling mechanism. *Integrate* [13], on the other hand, decides to run individual task at different frequencies through constrained scheduling, however, the frequency will not be changed during execution of a particular task, and frequency can only be changed to run a new task, if needed. The constrained scheduling of *Integrate* does not consider any power constraint, and its power model does not include switching power of the VR unlike *Prepare*. *Integrate* further power gates the cores during the slacks to reduce temperature of the underlying CMP.

For all of our tasks, *Prepare*, *GDP* and *Integrate* show significant and almost identical reduction in peak temperature, which is caused due to slacks in case of *Prepare* and *Integrate*, and the respective thresholds in case of *GDP*. As *Integrate* targets to maximize slack, in some cases *Integrate* offers slightly better thermal gains. However, the reductions in peak temperatures for all tasks are significantly higher, with a range between 6 to 8.6 °C, shown in Figure 14. But, executing instructions in lower V/F during violation of the temperature threshold in case of *GDP* leads to noticeable aggravation in performance, which is shown in Figure 15. As *Integrate* maintains the assigned frequency during execution of a particular task, no noticeable changes in instructions per second (IPS) is observed. On an average, *GDP* degrades performance (IPS) by 8%, that leads to deadline failure for the considered task-set, whereas reduction in execution length in case of *Prepare* will assist in improving result-accuracy along with a scope for power gating during the generated slacks. Thus, while offering almost similar thermal gains, *Prepare* outperforms both GDP [1] and *Integrate* [13] in terms of performance in our considered time-critical environment that executes AC based task-set.

4.2.6 Discussion. Our constrained scheduling first generates the schedule for its task-set (represented by PTG), which is able to achieve a high NAQ of around 75%, while maintaining the power and deadline constraints with 70 − 80% of system workload. During execution, Algorithm 1 shows

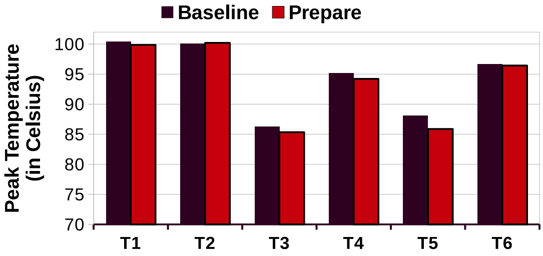


Fig. 13. Runtime peak temperature.

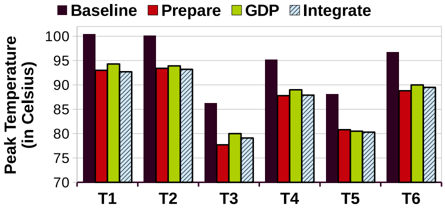


Fig. 14. Overall gain in peak temperature.

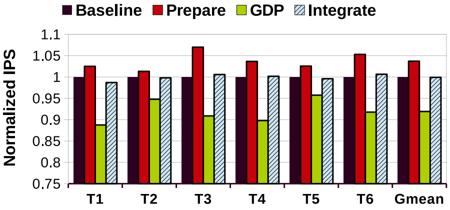


Fig. 15. Change in performance.

an improvement of 5.3% in result-accuracy for our test case discussed above. Basically, applying FG-DVFS to reduce the core V/F settings during LLC-miss induced stalls does not impact the execution time, but makes room for energy-adaptive scaling of V/F. Hence, by executing tasks at some higher V/F than the predetermined one for a short time-span just after the memory stall, significantly reduces the execution time. However, such spike in V/F might incur temperature overshoot during execution, which is controlled by our energy-adaptive mechanism that does not allow a core to stay at the higher V/F for a long time-span, thanks to energy-adaptive mechanism (see Sec. 3.2.3). By employing energy-adaptive V/F scaling, we observed higher reduction in execution times for the tasks that generates online slacks. These online slacks are further exploited by *Prepare* to enhance the result-accuracy as well as thermal efficiency of the underlying system, while maintaining the timing constraint.

5 PRIOR WORK

Energy minimization in recent CMP based real-time systems has become a topic of paramount importance [35, 36]. Energy efficient scheduling for the time-critical dependent tasks on CMP platform, imposes enormous research challenges [19]. In recent past, a few research attempts [4, 18, 24] were undertaken to devise energy-aware real-time scheduling for a set of real-time task-sets.

Recently, Cao et al. introduced the concept of AC towards meeting the energy budget of a large scale real-time system for the tasks without precedent constraints [8]. Other prior efforts further explored AC tasks scheduling for embedded real-time systems with energy minimization [8, 30, 46], for the set of independent tasks. Yu et al. first proposed the concept of a "Imprecise Computation (IC)" [44], where tasks can be decomposed into mandatory and optional parts. Authors further proposed a "dynamic-slack-reclamation" technique, that improves the system-wide QoS for more energy savings, but task-dependencies were not considered. To the best of our knowledge, the very first attempt to schedule IC/AC dependent tasks is found in [43], where authors compared the performance of conventional real-time scheduling approaches like Highest Level First (HLF) and Least Space Time First (LSTF) between a couple of task-sets, where one set has the AC tasks, but energy efficiency was not accounted. The energy cognizant scheduling of dependent AC tasks were further considered in [32, 33] that employed DVFS at the cores for energy savings.

The majority of the prior energy/thermal management policies [12, 26] control the dynamic power of the cores in CMPs either by employing DVFS [38, 39] or through task migration [11, 15, 16]. Recently, Roeder et al. [39] studied the effectiveness of DVFS, planned offline, for heterogeneous real-time systems with multi-version task-model. However, energy efficiency of such systems can be improved further by dynamic adjustment of the offline generated schedule based on the runtime tasks' as well as system's characteristics. Donald and Martonosi [12] classified the thermal management techniques into several groups based on their distinct implementation strategies. This paper has shown the efficacy of several DVFS in combination with task migration based policies towards controlling temperature, where distributed DVFS in addition with the task migration are claimed to be the best. But, underlying migration overheads at the cache hierarchy were not considered, which might raise scalability issues in large sized CMPs. Hanumaiah et al. [22] proposes a thermal efficient thread migration policy, that was integrated with DVFS to reduce temperature of the homogeneous CMPs [21]. Recently, Esmaili et al. has also integrated DPM, DVFS and task migration in their constrained scheduling, where the power budget of the system was not considered [13]. However, combining DVFS and DPM can boost up system throughput and thermal efficiency of the large CMPs [27].

Most of these prior techniques tackled power/thermal issues by employing DVFS that is operated by the off-chip voltage regulators, and hence, DVFS is applied in a coarse grained manner [26], except for the strategy proposed in [14]. In *Prepare*, we have studied the potential of FG-DVFS in improving thermal efficiency of a multi-core system. Next, *Prepare* further proposed a novel energy-adaptive technique (see Sec. 3.2) that dynamically enhances the gained result-accuracy in offline (see Sec. 3.1) for a set of approximated real-time tasks. We also empirically validate the claims with our simulation setup and illustrated that our energy-adaptive online mechanism assures thermal safety (see Sec. 4.2.4 and 4.2.5). Our results also shows that, both offline and online mechanisms of *Prepare* outperform state-of-the-art techniques. To the best of our knowledge, *Prepare is the first technique that considers LLC-miss induced stalls during execution of a scheduled task-set to conduct energy-adaptive fine-grained V/F scaling that enhances result-accuracy during execution of the AC real-time precedence constrained task-set without violating deadline constraint, while maintaining on-chip thermal safety.*

6 CONCLUSION

This paper has proposed *Prepare*, a novel approximate real-time tasks scheduling approach that schedules a set of dependent approximate tasks on a CMP, with an objective to enhance result-accuracy. Our ILP based scheduling mechanism is constrained by system-wide power and deadline. *Prepare* also includes a runtime energy-adaptive V/F management strategy that assists in enhancing results-accuracy of the task-set, as well as thermal efficiency of the underlying hardware. Basically, FG-DVFS reduces the core power consumption during LLC induced stalls and scales up V/F to a higher value than the predetermined ones just after the stall period. To maintain the peak temperature, this higher V/F is maintained (in an energy-adaptive manner) for a smaller time-span, which further reduces execution length of the individual task, and is exploited to enhance result-accuracy as well as thermal/energy efficiency.

With 70 − 80% workload, *Prepare* achieved 75% QoS with its constrained scheduling, which is enhanced by 5.3% for our benchmark based evaluation of the online energy-adaptive mechanism, while meeting the deadline constraints. Overall, *Prepare* achieves up to 8.6 °C reduction in peak temperature by applying power gating during the online slacks. Our empirical evaluation shows, *Prepare* outperforms a state-of-the-art prior scheduling policy and a couple of recently proposed DVFS based thermal management techniques.

ACKNOWLEDGEMENT

This work was funded by *Marie Curie Individual Fellowship (MSCA-IF)*, EU (Grant Number: 898296) and *Engineering and Physical Sciences Research Council (EPSRC)*, UK (Grant Numbers: EP/R02572X/1, EP/P017487/1 and EP/V000462/1). We are thankful to the anonymous reviewers for their constructive feedback and suggestions that helped us in improving the quality of the manuscript.

REFERENCES

[1] S. Dey A. Mirtar and A. Raghunathan. 2015. Joint Work and Voltage/Frequency Scaling for Quality-Optimized Dynamic Thermal Management. *IEEE TVLSI* (2015).

[2] S. Achour and M. C. Rinard. 2015. Approximate Computation with Outlier Detection in Topaz. *SIGPLAN Not.* (2015).

[3] K. Bhatti, C. Belleudy, and M. Auguin. 2010. Power Management in Real Time Embedded Systems through Online and Adaptive Interplay of DPM and DVFS Policies. In *EUC*.

[4] A. Bhuiyan, Z. Guo, A. Saifullah, N. Guan, and H. Xiong. 2018. Energy-efficient real-time scheduling of DAG tasks. *ACM TECS* (2018).

[5] C. Bienia, S. Kumar, J. P. Singh, and K. Li. 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *PACT*.

[6] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. 2011. The Gem5 Simulator. *SIGARCH CAN* (2011).

[7] C. Bliek, P. Bonami, and A. Lodi. 2014. Solving mixed-integer quadratic programming problems with IBM-CPLEX: a progress report. In *RAMP*.

[8] K. Cao, G. Xu, J. Zhou, T. Wei, M. Chen, and S. Hu. 2018. QoS-adaptive approximate real-time computation for mobility-aware IoT lifetime optimization. *IEEE TCAD* (2018).

[9] S. Chakraborty and H. K. Kapoor. 2018. Analysing the Role of Last Level Caches in Controlling Chip Temperature. *IEEE TSUSC* (2018).

[10] S. Chakraborty and H. K. Kapoor. 2019. Exploring the Role of Large Centralised Caches in Thermal Efficient Chip Design. *ACM TODAES* (2019).

[11] T. Chantem, R. P. Dick, and X. S. Hu. 2008. Temperature-Aware Scheduling and Assignment for Hard Real-Time Applications on MPSoCs. In *DATE*.

[12] J. Donald and M. Martonosi. 2006. Techniques for Multicore Thermal Management: Classification and New Exploration. In *ISCA*.

[13] A. Esmaili, M. Nazemi, and M. Pedram. 2019. Modeling Processor Idle Times in MPSoC Platforms to Enable Integrated DPM, DVFS, and Task Scheduling Subject to a Hard Deadline. In *ASPDAC*.

[14] S. Eyerman and L. Eeckhout. 2011. Fine-grained DVFS Using On-chip Regulators. *ACM TACO* (2011).

[15] Y. Ge, P. Malani, and Q. Qiu. 2010. Distributed task migration for thermal management in many-core systems. In *DAC*.

[16] Y. Ge, Q. Qiu, and Q. Wu. 2012. A Multi-Agent Framework for Thermal Aware Task Migration in Many-Core Systems. *IEEE TVLSI* (2012).

[17] M. E. T. Gerards and J. Kuper. 2013. Optimal DPM and DVFS for Frame-Based Real-Time Systems. *ACM TACO* (2013).

[18] Z. Guo, A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, and N. Guan. 2019. Energy-Efficient Real-Time Scheduling of DAGs on Clustered Multi-Core Platforms. In *RTAS*.

[19] Z. Guo, A. Bhuiyan, A. Saifullah, N. Guan, and H. Xiong. 2017. Energy-Efficient Multi-Core Scheduling for Real-Time DAG Tasks. In *ECRTS*.

[20] V. Hanumaiah and S. Vrudhula. 2014. Energy-Efficient Operation of Multicore Processors by DVFS, Task Migration, and Active Cooling. *IEEE TC* (2014).

[21] V. Hanumaiah, S. Vrudhula, and K. S. Chatha. 2009. Maximizing performance of thermally constrained multi-core processors by dynamic voltage and frequency control. In *ICCAD*.

[22] V. Hanumaiah, S. Vrudhula, and K. S. Chatha. 2011. Performance Optimal Online DVFS and Task Migration Techniques for Thermally Constrained Multi-Core Processors. *IEEE TCAD* (2011).

[23] H. B. Jang, J. Lee, J. Kong, T. Suh, and S. W. Chung. 2014. Leveraging Process Variation for Performance and Energy: In the Perspective of Overclocking. *IEEE Trans. on Comp.* (2014).

[24] K. Kanoun, N. Mastronarde, D. Atienza, and M. Van der Schaar. 2014. Online energy-efficient task-graph scheduling for multicore platforms. *IEEE TCAD* (2014).

[25] S. K. Khatamifard, L. Wang, W. Yu, S. Köse, and U. R. Karpuzcu. 2017. ThermoGater: Thermally-aware on-chip voltage regulation. In *ISCA*.

[26] J. Kong, S. W. Chung, and K. Skadron. 2012. Recent Thermal Management Techniques for Microprocessors. *ACM CSUR* (2012).

[27] J. Lee and N. S. Kim. 2012. Analyzing Potential Throughput Improvement of Power- and Thermal-Constrained Multicore Processors by Exploiting DVFS and PCPG. *IEEE TVLSI* (2012).

[28] Kyung-Jung Lee, Jae-Woo Kim, Hyuk-Jun Chang, and Hyun-Sik Ahn. 2018. Mixed harmonic runnable scheduling for Automotive Software on Multi-core processors. *International Journal of Automotive Technology* 19, 2 (2018), 323–330.

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

[30] I. Méndez-Díaz, J. Orozco, R. Santos, and P. Zabala. 2017. Energy-aware scheduling mandatory/optional tasks in multicore real-time systems. *International Transactions in Operational Research* (2017).

[31] S. Mittal. 2016. A survey of techniques for approximate computing. *ACM CSUR* (2016).

[32] L. Mo, A. Kritikakou, and O. Sentieys. 2018. Energy-quality-time optimized task mapping on DVFS-enabled multicores. *IEEE TCAD* (2018).

[33] L. Mo, A. Kritikakou, and O. Sentieys. 2019. Approximation-aware Task Deployment on Asymmetric Multicore Processors. In *DATE*.

[34] O. Mutlu and T. Moscibroda. 2007. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In *MICRO*.

[35] S. Narayana, P. Huang, G. Giannopoulou, L. Thiele, and R. V. Prasad. 2016. Exploring Energy Saving for Mixed-Criticality Systems on Multi-Cores. In *RTAS*.

[36] S. Pagani, J. Chen, and J. Henkel. 2015. Energy and peak power efficiency analysis for the single voltage approximation (SVA) scheme. *IEEE TCAD* (2015).

[37] M. Qamhieh and S. Midonnet. 2015. Simulation-based evaluations of DAG scheduling in hard real-time multiprocessor systems. *ACM SIGAPP Appl. Comput. Rev.* (2015).

[38] R. Rao, S. Vrudhula, C. Chakrabarti, and N. Chang. 2006. An Optimal Analytical Solution for Processor Speed Control with Thermal Constraints. In *ISLPED*.

[39] J. Roeder, B. Rouxel, S. Altmeyer, and C. Grelck. 2021. *Energy-Aware Scheduling of Multi-Version Tasks on Heterogeneous Real-Time Systems*.

[40] J. Shun and G. E. Blelloch. 2013. Ligra: A Lightweight Graph Processing Framework for Shared Memory. In *PPoPP*.

[41] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard. 2011. Managing Performance vs. Accuracy Trade-Offs with Loop Perforation. In *ACM SIGSOFT*.

[42] G. P. Srinivasa, D. Werner, M. Hempstead, and G. Challen. 2021. Thermal-Aware Overclocking for Smartphones. In *ISPASS*.

[43] G. L. Stavrinides and H. D. Karatza. 2010. Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations. *JSS* (2010).

[44] H. Yu, B. Veeravalli, and Y. Ha. 2008. Dynamic scheduling of imprecise-computation tasks in maximizing QoS under energy constraints for embedded systems. In *ASPDAC*.

[45] R. Zhang, M. R. Stan, and K. Skadron. 2015. *HotSpot 6.0: Validation, Acceleration and Extension.* Technical Report CS-2015-04. University of Virginia.

[46] J. Zhou, J. Yan, T. Wei, M. Chen, and X. S. Hu. 2017. Energy-adaptive scheduling of imprecise computation tasks for QoS optimization in real-time MPSoC systems. In *DATE*.