DELICIOUS: Deadline-Aware Approximate Computing in Cache-Conscious Multicore

DELICIOUS: Deadline-Aware Approximate Computing in Cache-Conscious Multicore

Sangeet Saha, Shounak Chakraborty, Sukarn Agarwal, Rahul Gangopadhyay, Magnus Själander, and Klaus McDonald-Maier

Abstract—Enhancing result-accuracy in approximate computing (AC) based real-time systems, without violating power constraints of the underlying hardware, is a challenging problem. Execution of such AC real-time applications can be split into two parts: (i) *the mandatory part*, execution of which provides a result of acceptable quality, followed by (ii) *the optional part*, that can be executed partially or fully to refine the initially obtained result in order to increase the result-accuracy, without violating the time-constraint. This paper introduces *DELICIOUS*, a novel hybrid offline-online *scheduling strategy* for AC real-time dependent tasks. By employing an efficient *heuristic algorithm*, *DELICIOUS* first generates a schedule for a task-set with an objective to maximize the results-accuracy, while respecting system-wide constraints. During execution, *DELICIOUS* then introduces a *prudential cache resizing* that reduces temperature of the adjacent cores, by generating thermal buffers at the turned off cache ways. *DELICIOUS* further trades off this thermal benefits by enhancing the processing speed of the cores for a stipulated duration, called *V/F Spiking*, without violating the power budget of the core, to shorten the execution length of the tasks. This reduced runtime is exploited either to enhance result-accuracy by dynamically adjusting the optional part, or to reduce temperature by enabling sleep mode at the cores. While surpassing the prior art, *DELICIOUS* offers 80% result-accuracy with its scheduling strategy, which is further enhanced by 8.3% in online, while reducing runtime peak temperature by 5.8 ℃ on average, as shown by benchmark based evaluation on a 4-core based multicore.

Index Terms—Real-time Systems, Approximate Computing, Thermal Management, Dead Block, Caches Resizing, TDP

1 Introduction

In real-time systems, the correctness not only depends on the result-accuracy, but also on the time at which these results are produced. For such time-critical scenarios, approximated results obtained on-time are preferable over accurate results produced after the deadline. In plenty of application domains, such as multimedia computing, tracking of mobile targets, real-time heuristic search, information gathering and control systems, an approximate result, obtained before the deadline is usually acceptable [5]. For example, in case of video streaming, frames having lower quality are better than completely missing frames. In target tracking, an approximated estimation of the target's location generated within deadline is better than an accurate location, obtained too late. In these domains, a task is logically decomposed into a mandatory subtask and an optional subtask [9], [35], [37]. The entire mandatory subtask must be completed before the deadline to generate the minimally acceptable QoS, followed by a partial/complete execution of the optional part, subject to availability of the resources, to improve accuracy of the initially obtained result within the deadline. The QoS increases with the number of execution cycles spent on the optional part.

Energy efficient scheduling of the AC real-time task-set that intends to improve result-accuracy without violating the underlying system constraints have become an active research avenue in recent past. Stavrinides and Karatza were among the first to propose scheduling of an AC real-time task-set [44]. A recent theoretical analysis [37] shows how to improve system level result-accuracy through task to processor allocation and task adjustment constrained by an energy budget. However, limiting the energy usage does not ensure thermal safety of the chip, which can be tackled by incorporating power constraint, like thermal design power (TDP), together with a runtime power management while considering several architectural parameters. In an energy efficient approach, *Prepare* [10], to improve system level result-accuracy, the authors considered the runtime architectural characteristics. However, the detailed runtime cache characteristics of the applications were not considered.

Researchers also employed integer linear programming (ILP) based scheduling strategies [10], [37] that might often become prohibitively expensive for large problem sizes, which can be overcome by designing a computationally feasible heuristic strategy. In *DELICIOUS*, we devise an efficient scheduling heuristic to schedule approximated real-time tasks on a chip multiprocessor (CMP) platform, where the scheduling is constrained by task-dependency and deadlines. The entire strategy of *DELICIOUS* is summarized in Figure 1. Our AC real-time application contains n number of dependent tasks (T_1 to T_n shown in the top of Figure 1) and the entire application has a deadline. Each task is equipped with multiple versions with diverse set of result-accuracy

- *S. Saha and K. McDonald-Maier are with School of Computer Science and Electronic Engineering, University of Essex, UK. E-mail: sangeet.saha@essex.ac.uk, kdm@essex.ac.uk.*
- *S. Chakraborty and M. Själander are with the Department of Computer Science, Norwegian University of Science and Technology, Trondheim, Norway 7491. E-mail: shounak.chakraborty@ntnu.no, magnus.sjalander@ntnu.no.*
- *S. Agarwal is with School of Informatics, University of Edinburgh, UK. E-mail: sagarwa2@ed.ac.uk.*
- *R. Gangopadhyay is with Faculty of Mathematics and Computer Science, St. Petersburg State University, Russia. E-mail: rahulincxtint@gmail.com.*

Manuscript received XXXXX xx, XXXX; revised XXXX XX, XXXX.

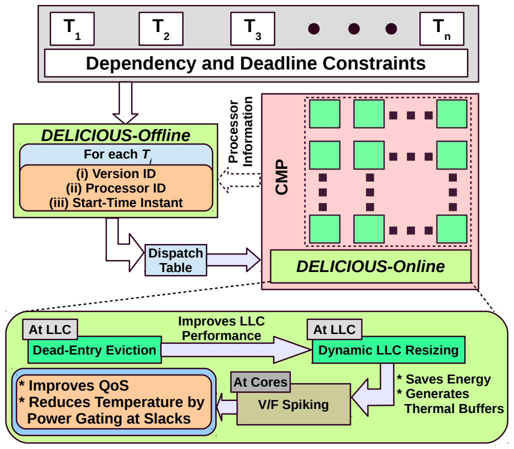

Fig. 1: *DELICIOUS*: Process Overview

based on the respective execution length of the optional part that is executed. In *DELICIOUS-Offline* (shown in the left of Figure 1), the scheduling information, which versions of a task (*Version ID*) will be executed on which core (*Processor ID*) in a CMP and its starting time (*Start-Time Instant*) for all the tasks will be generated with an objective to maximize the overall system-level result-accuracy. All tasks are assigned a base voltage/frequency (V/F) level, which is the highest possible V/F (other than turbo mode [2]) for the underlying processor core. The generated schedule is next stored in a dispatch table (shown just below the *DELICIOUS-Offline* part in Figure 1), from which task-executions are triggered.

With the objective to further enhancing the accuracy by exploiting runtime architectural characteristics (shown at the bottom of Figure 1), *DELICIOUS-Online* judiciously selects and evicts dead blocks[1] from the shared last level cache (LLC) and turns off spare LLC ways to reduce the temperature of the cores in its proximity. By considering the live thermal status, *DELICIOUS* attempts to execute tasks at a higher frequency than that originally assigned for a stipulated duration (so called *V/F Spiking*, based on fine-grained DVFS [17]). *V/F Spiking* increases throughput and enables more of the optional part of a task to be executed, and thus improves the QoS without impacting the pre-determined schedule. To improve power and thermal efficiency further, *DELICIOUS* shuts down cores during unused slacks generated by reducing execution times of the tasks.

The contributions of *DELICIOUS* are as follows:

1) Our intended problem has been clearly formulated as an optimization problem, discussed in Sec. 4, subject to a set of constraints.
2) We have presented a real-time scheduling policy, *DELICIOUS*, for AC real-time precedence constrained task graphs (PTGs) on homogeneous CMPs.
3) Design of a heuristic strategy for an AC real-time PTG on a CMP, where each task can have multiple versions with distinct degrees of accuracy (see Sec. 5). In addition to delivering satisfactory performance, the strategy exhibits reasonable time complexity with comparatively low, polynomial time scheduling overheads.
4) We apply a power/thermal restriction (i.e. TDP) aware *V/F Spiking* technique (see Sec. 6), induced by online *LLC-resizing*, to improve achieved QoS while keeping temperature in check, which we have empirically validated and reported in Figure 11 and 12.
5) By shortening the execution time for each task, *V/F Spiking* incurs *dynamic slacks*, which are either exploited *(i)* to execute a higher task-version subject to availability, or *(ii)* to put the core in sleep mode to reduce core temperature (see Sec. 6).

1. Dead blocks indicate the data that will never be accessed before being evicted from the cache (detailed later in Sec. 6).

We further argue and empirically validate the efficacy of the task-scheduling heuristic of *DELICIOUS* in combination with the runtime mechanisms (see Sec. 7). For a set of tasks, the scheduling heuristic of *DELICIOUS* achieves 80% QoS, which is close to a recent ILP based optimal policy, *Prepare* [10] that achieves a QoS of 83%, while running time of ILP based optimal scheduling of *Prepare* is significantly higher than the scheduling heuristic of *DELICIOUS* (see Figure 6). Our benchmark based evaluation with a 4-core based baseline CMP (equipped with 4MB 16-way associative shared L2 cache) in our simulation setup (consisted of gem5 [8], McPAT [30], and Hotspot [48]) shows that the dynamic LLC-resizing induced and TDP aware *V/F Spiking* of *DELICIOUS* further stimulates the achieved QoS by 8.3% and reduces core-temperature up to 9.2 °C, while meeting the deadlines. Our empirical analysis shows that, online mechanism of *DELICIOUS* outperforms *Prepare* [10] and *GDP* [33], in terms of online QoS enhancement, and peak temperature reduction. To the best of our knowledge, *DELICIOUS* is the first scheduling mechanism that introduces a dead block eviction based LLC-resizing induced TDP aware *V/F Spiking* technique for enhancing the QoS of dependent AC real-time task-set without violating the deadline and the thermal constraints.

Before formulating the problem in Sec. 4, we discuss the relevant prior work in Sec. 2, and brief our system model and assumptions in Sec. 3. The core offline and online mechanisms of *DELICIOUS* are detailed in Sec. 5 and Sec. 6, respectively. The evaluation of offline and online mechanisms of *DELICIOUS* are presented next in Sec. 7 before concluding the paper in Sec. 8. The acronyms used in our paper are abbreviated in Table 1.

TABLE 1: Acronyms and their Abbreviations

Acronyms	Abbreviations
AC	Approximate Computing
ILP	Integer Linear Programming
RoI	Region of Interest
PTG	Precedence-constrained Task Graph
QoS	Quality of Service
NAQ	Normalized Achieved QoS
CMP	Chip Multiprocessor
LLC	Last Level Cache
V/F	Voltage/Frequency
OoO	Out of Order
DVFS	Dynamic Voltage and Frequency Scaling
DPM	Dynamic Power Management

2 State-of-the-Art

Minimizing energy in recent CMP based real-time systems has become a topic of paramount importance [38], [39]. Scheduling time-critical dependent tasks on CMP platform while maintaining the energy/power constraint is gradually becoming challenging with technology scaling [22]. Researchers recently attempted to devise energy-aware scheduling for the real-time task-sets with various system-wide constraints [6], [23], [27]. In 2018, the concept of AC to meet the energy budget of a large scale real-time system was introduced for the tasks without precedent constraints [9]. Other prior arts also explored AC task scheduling for the embedded real-time systems while minimizing energy [9], [32], [49], for the set of independent tasks. Yu et al. proposed the concept of an "Imprecise Computation (IC)" [47], for the first time, where individual tasks are decomposed into mandatory and optional parts, and their "dynamic-slack-reclamation" technique improves the system-wide QoS for more energy savings, but task-dependencies were not considered. To the best of our knowledge, in the very first attempt to schedule IC/AC dependent tasks [44], authors measured the performance of conventional real-time scheduling techniques like Highest Level First (HLF) and Least Space Time First (LSTF) for a couple of task-sets, where one set contains the AC tasks, but energy efficiency was not considered. The energy aware scheduling of dependent AC tasks were considered in some prior works [36], [37] that employed DVFS at the cores.

Most of the prior energy/thermal management mechanisms [14], [17], [28] control the dynamic power of the cores in CMPs either by employing DVFS [41], [42] or by migrating tasks [13], [19], [20]. Recently, Roeder et al. [42] showed the effectiveness of DVFS, planned offline, for a heterogeneous real-time system with multi-version based task-model, but energy efficiency can be enhanced dynamically based on the runtime tasks' as well as system's characteristics. Donald and Martonosi [14] have shown the efficacy of different DVFS techniques along with task migration policies to control temperature, where distributed DVFS applied with task migration are claimed to be the best. However, underlying migration overheads at the caches were not accounted. Hanumaiah et al. [26] proposed a thermal efficient thread migration, that was integrated with DVFS to reduce temperature of the homogeneous CMPs [25]. Recently, Esmaili et al. also integrated DPM, DVFS, and task migration in constrained scheduling, but the power budget of the system was not included [16]. Another study shows how combining DVFS and DPM can significantly boost up system throughput and thermal efficiency of the large sized CMPs [29]. However, a couple of recent attempts have tried to combine DVFS with the cache based policies [11], [12], but their efficacy in improving QoS of the AC real-time systems have not been studied. Moreover, these studies did not focus on the block recency before evicting them from the LLC, which we have studied in *DELICIOUS*.

2.1 *DELICIOUS* Over Prior Arts

In *DELICIOUS*, we investigated the potential of LLC way-shutdown in improving thermal efficiency of a multicore system, and how this benefits can be traded off to improve core performance. Basically, *DELICIOUS* first proposes a novel heuristic based offline scheduling algorithm for a set of dependent AC real-time tasks, with an objective to improve the QoS (see Sec. 5). The QoS is further stimulated during execution by employing LLC resizing based mechanism that shuts down cache ways to reduce core-temperature in proximity, which assists the cores to maintain a higher V/F for a stipulated time-span (see Sec. 6). Our results also illustrates, both offline and online mechanisms of *DELICIOUS* surpass the recent techniques. To the best of our knowledge, *DELICIOUS* is the first technique that employs dynamic LLC-resizing for a scheduled AC real-time tasks to reduce core-temperature, that further offers room for *V/F Spiking* to enhance result-accuracy on-the-fly, while maintaining deadline and thermal safety.

3 System Model and Assumptions

The considered CMP consists of m homogeneous cores, denoted as $P = \{P_1, P_2, ..., P_m\}$. Each core supports L distinct V/Fs 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 offline schedule is generated by considering a single base core V/F ($\leq V_L/F_L$), at which core can execute tasks until completion without any potential thermal threats [1]. However, in online phase, during *V/F Spiking*, a core can execute tasks at higher V/F than the base level for a stipulated duration.

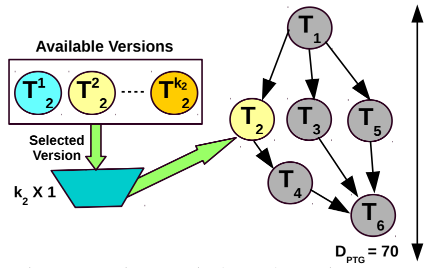

Fig. 2: Precedence task graph (PTG) with timing parameters

Our application is represented as a precedence task graph (PTG) (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 task-dependency or precedence relations between a distinct pair of tasks. An edge $\langle Ti, Tj\rangle$ implies a precedence, i.e. a task T_j can start its execution only after T_i is executed. Our single source and single sink tasks have no predecessors and no successors, respectively. Being a real-time application, G has to be executed within the given deadline, D_{PTG}, by executing all the associated tasks (T_i). Each T_i can have k_i different versions (signifying different degrees of accuracy), $T_i = \{T_i^1, T_i^2, \ldots, T_i^{k_i}\}$, those are distinct by their respective execution lengths (O_i), denoted as O_i^1, O_i^2, ..., $O_i^{k_i}$, where O_i^p offers higher result-accuracy than O_i^q, if $p > q$ [46]. *DELICIOUS* selects a particular version among the k_i versions of T_i, the selection procedure is detailed in the following section. For each optional part of a task (O_i), there exists a separate executable module, that is executed after the execution of the mandatory portion (M_i) of the respective task, T_i. The length of the j^{th} version of task T_i (len_i^j) can be defined as: $len_i^j = M_i + O_i^j$. Note that, len_i^j includes the cycles required for accessing LLC, which we obtain by executing an individual task for a particular configuration. We define result-accuracy Acc_i^j of T_i^j as 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 now defined as the sum of the executed cycles of O_i^j for all the tasks [9], which can be represented as: $QoS = \sum_{i=1}^{n} O_i^j \mid T_i = T_i^j$. Note that, in addition with execution of the M_i for each task, we also need to execute at least one version of O_i within deadline.

4 Problem Formulation

In order to present a formal model of the problem and its objective, we have formulated it as a constraint optimization problem. Let us consider a binary decision variable $Z_{ikt\eta}$, where $i = 1, 2, ..., n$; $k = 1, 2, ..., k_i$; $t = 0, 1, ..., D_{PTG}$, and $\eta = 1, 2, ...m$. Here, the indices i, k, t and η, denote task ID, corresponding version ID, timestamp, and processor ID, respectively. The variable $Z_{ikt\eta}$ is 1, if the k^{th} version of T_i (T_i^k) starts its execution at t^{th} timestamp on processor η. This will eventually enforce that $Z_{ikt\eta}$ for T_i will be zero, for all other possible combinations, i.e. it cannot start on any other processors with other versions at any time stamp. We now present the objective function with constraints on the binary variables to model the scheduling problem.

$$\textbf{Maximize} \quad \textbf{QoS} \tag{1a}$$

$$QoS = \sum_{i=1}^{n}\sum_{\eta=1}^{m}\sum_{k=1}^{k_i}\sum_{t=0}^{D_{PTG}} O_i^k \cdot Z_{ikt\eta} \tag{1b}$$

Subject to:

$$\sum_{k=1}^{k_i}\sum_{t=0}^{D_{PTG}}\sum_{\eta=1}^{m} Z_{ikt\eta} = 1 \qquad \forall i \in [1, n] \tag{1c}$$

$$\sum_{i=1}^{n}\sum_{k=1}^{k_i}\sum_{t'=\psi}^{t} Z_{ikt'\eta} \leq 1 \quad \forall t : 0 \leq t \leq D_{PTG} \;\&\; \forall \eta : 1 \leq \eta \leq m \tag{1d}$$

$$\psi = max(0, t - exe_i^k + 1)$$

$$st_j \geq et_i \qquad \langle T_i, T_j\rangle \in E \tag{1e}$$

$$st_n + el_n \leq D_{PTG} \tag{1f}$$

$$st_j = \sum_{\eta=1}^{m}\sum_{k=1}^{k_j}\sum_{t=0}^{D_{PTG}} t \cdot Z_{jkt\eta} \tag{1g}$$

$$el_i = \sum_{\eta=1}^{m}\sum_{k=1}^{k_i}\sum_{t=0}^{D_{PTG}} exe_i^k \cdot Z_{ikt\eta} \tag{1h}$$

$$et_i = st_i + el_i \tag{1i}$$

Equation 1b presents the objective function in the above formulation, whereas Equation 1c enforces the constraint that each task must start its execution on a particular processor at a unique timestamp with a unique version. In this scheduling problem, resource bounds for processors must be satisfied at each timestamp. Any processor can execute at most one task at a given time without any preemption (Equation 1d). Equation 1e and 1f enforce execution dependency and deadline satisfaction constraints, respectively, whereas start time (st_j), execution length (el_i) and end time (et_i) are defined in Equation 1g, 1h, and 1i, respectively.

Our scheduling problem stated above amicably lends itself towards its computation using a standard optimization tool, CPLEX. However, the presence of numerous decision variables and constraints makes this problem computationally highly complex. Therefore, solution techniques using standard optimizers, like CPLEX, are often computationally expensive in terms of time and space even for moderate problem sizes with respect to number of tasks, number of processors, nature of inter-task dependencies, etc. We reiterate here that the main motivation towards encoding of our problem as above is the clarity it lends in detailed understanding and appreciating the structure of the scheduling problem at hand. Such realization is immensely useful towards designing and analyzing an efficient lower overhead heuristic strategy for the problem. We next present *DELICIOUS-Offline*, an efficient heuristic algorithm for the problem discussed above.

Algorithm 1: *DELICIOUS-Offline*

```
Input:
i. Task graph $G(T, E)$
ii. $k_i$: Number of versions of each task $T_i$
iii. $l_i^j$ : Execution length of $j^{th}$ version of task $T_i$
iv. $D_{PTG}$: The deadline of the task graph.
v. $Acc_i^j$: accuracy achieved by executing the $j^{th}$ version of $T_i$
Output:
i. Task Schedule /* Selected Task versions ($\zeta_i$), Execution start
times ($st_i$), Mapped Processor id: ($P_i^j$ i.e. $i^{th}$ task on $j^{th}$
Processor, Obtained Accuracy) */
ii. Achieved system-level QoS.
1  $\forall T_i \in T$, Set $\zeta_i = k_i$ (highest version) /* Set the selected
     version to the highest version*/
2  while Algorithm 3 does not yield TRUE do
3      Store ALAP time in a priority list of non-decreasing order
         returned by Algorithm 3
4      $\forall T_i \in (T \mid k_i > 1)$, Compute the Penalty Factor
         $PF(T_i, \zeta_i)$, using Equation 2
5      Create a min-heap of tasks in $T$ with the $PF(T_i, k_i)$
         values as the key;
6      if Multiple $T_i$ have same $PF$ then
7          Select $T_i$ from the priority list (with highest ALAP
             value) ;
8      Extract the task $T_j$ at the root of the min-heap;
9      $\zeta_j = \zeta_j - 1$; /* Decrease the current version of $T_j$ by one;
         */
10     Compute the $PF(T_j, k_j)$ and reheapify;
11 Calculate $QoS(\mathcal{A})$ as: $QoS(\mathcal{A}) = \sum_{i=1}^{|T|} Acc_i^{\zeta_i}$;
12 Return $QoS(\mathcal{A})$ ;
```

5 *DELICIOUS-Offline* Phase

Typically, list scheduling-based heuristic techniques are employed to compute feasible schedules for PTGs executing on multi-cores. They attempt to construct a static-schedule for the given PTG, to minimize the overall schedule length, while satisfying resource and precedence constraints. On the contrary, our heuristic strategy tackles the problem of scheduling a PTG consisting of task nodes with multiple versions, to maximize overall system accuracy, while satisfying the deadline constraint. For this purpose, we devise our heuristic algorithm, *DELICIOUS-Offline*, to generate a schedule by setting all task nodes to their highest version. Since *DELICIOUS-Offline* attempts to maximize the overall system level accuracy, the resulting schedule length may however violate the given deadline. This situation can then be refrained by degrading the versions of tasks, while reducing impact on overall system accuracy.

5.1 *DELICIOUS-Offline* Algorithm

Our heuristic algorithm for *DELICIOUS-Offline* is represented in Algorithm 1, that first attempts to generate a feasible schedule with considering the highest version of all the tasks by calling Algorithm 3, *Sched-Gen* (line 1 to 2). *Sched-Gen* yields TRUE, if a feasible schedule is possible by satisfying the resource and deadline constraints for each task, and returns FALSE and ALAP[2] times (generated by Algorithm 2), otherwise. By considering all tasks with their respective highest versions may not be feasible due to their

2. It implies *As Late As Possible*.

Algorithm 2: *ALAP Time Calculation*

```
Input:
i.The task graph $G(T, E)$
ii. $\zeta_i$: selected version of each task $T_i$
iii. $len_i^{\zeta}$ : Execution length for the $\zeta^{th}$ version of $T_i$
iv. $D_{PTG}$: The deadline of the task graph.
Output:
i. $e_i^{la}$ : latest start time of each task $T_i$
1  for $T_i \in T$ do
2      if $T_i$ is a sink task in PTG then
3          $e_i^{la} = D_{PTG} - min\ (len_i^{\zeta})$
4      else
5          Calculate the minimum of the latest start times
             $min\ (e_j^{la})\ \forall\ T_j \in Succ(T_i)$ ;
           // Let task $T_{sc}$ has the minimum value of
              the latest start times among all
              successors of $T_i$
6          $e_i^{la} = e_{sc}^{la} - min\ (len_i^{\zeta})$ ;
```

high temporal requirements. If *Sched-Gen* yields FALSE, then *DELICIOUS-Offline* enters into a while loop until a feasible schedule for a chosen set of task versions is generated or all tasks have been reduced to their lowest versions (line 4 to 10). This while loop maintains the tasks in a priority queue organized as a min-heap with a parameter called $Penalty\ Factor(PF)$ as key (Equation 2). For a given task, T_i, with its current version ζ_i, $PF(T_i, \zeta_i)$ is defined by the reduction in achieved accuracy as T_i's version is lowered from ζ_i to $\zeta_i - 1$, and is calculated as:

$$PF(T_i, \zeta_i) = O_i^{\zeta_i} - O_i^{\zeta_i - 1} \quad (2)$$

If two tasks exhibit the same PF values, then the task with lower ALAP value will be selected from the ordered list (line 6 to 7). This is mainly due to the fact that in such a priority list based on task's ALAP times, the actual value of the ALAP time of a task provides an estimate of the remaining computational demand before completion of the sink task. For any given deadline bound, a relatively lower ALAP time for a task indicates a higher remaining processing requirement. Hence, *DELICIOUS-Offline* attempts to lower the version of a task having higher ALAP value, i.e. having lower processing requirement and less dependency.

In each iteration of the loop, *DELICIOUS-Offline* extracts the task (T_j) at the root of the min-heap (task with the minimum PF value), reduces its version by one, and check if the *Sched-Gen* returns TRUE or not (line 9 to 10). If *Sched-Gen* yields TRUE, then it indicates that a feasible schedule is obtained. *DELICIOUS-Offline* will then calculate and return the obtained system level QoS as output (line 11 to 12).

5.2 Schedule Generation (Sched-Gen)

DELICIOUS-Offline calls *Sched-Gen* (Algorithm 3) to determine a valid schedule for a stipulated set of task versions chosen by *Sched-Gen*.

Initialization and Task Prioritization (line 1 to 8): Algorithm 3 begins its execution by creating an array denoted as FP, which implies the number of free processors available. *Sched-Gen* uses a relative priority order amongst all tasks based on the tasks' ALAP start time, considering each task T_i at its currently selected versions ζ_i. This priority list based on task's ALAP times ensures that inter-task precedence relationships are always satisfied (ALAP time of a predecessor task is always less than the ALAP times of all its successors).

Algorithm 3: *Schedule Generation (Sched-Gen)*

```
Input:
i. $l_i^{\zeta_i}$ : Execution length of the selected $\zeta_i^{th}$ version of $T_i$
ii. $D_{PTG}$: Deadline of the task graph
Output: TRUE/FALSE: Feasible or infeasible schedule
1  /*.......................... INITIALIZATION.....................................*/ /*
     Let FP denote the set of processors currently available for
     execution; */
2  Initialize $FP = P$;
3  $\forall\ P_i \in FP$, Set $PL_i$= FALSE; /* Initialization, $PL_i$ : A flag
     which is set to FALSE if the Processor is available for
     execution; TRUE, otherwise; initially all processors are free
     */
4  /* Let FT denote the task-set currently finished their
     execution; */
5  $FT = NULL$;
6  /*.....................TASK PRIORITIZATION......................*/
7  Calculate ALAP start time ($e_i^{la}$) for each task $T_i$
     using Algorithm 2 at its currently selected version $\zeta_i$
8  $\hat{\tau}$ = $T$ /* Copy tasks into $\hat{\tau}$ */
9  /*...........TASK MAPPING & EXECUTION.............*/
10 for $t = 0$; $t \leq D_{PTG}\ AND\ T\ \neq NULL$; $t++$ do
11     for each processor in parallel do;
12     if There exists tasks ($T_j \in T$) | All predecessors of $T_j$ have
         finished their execution AND $FP \neq NULL$ then
13         Select processor $P_i$ with $PL_i$ == FALSE ;
14         Set $PL_i$ = TRUE /* Set $P_i$ to busy; */
15         Map $T_j$ in processor $P_i$;
16         $st_j$ = $t$ /* Set current time $t$ as the execution start time
             of $T_j$*/
17         $PBP_i$= $l_j^{\zeta_j}$ ; /* start execution of $T_j$; $PBP_i$: an integer
             variable denoting Procsseor Busy Period which
             holds the remaining time required to finish the
             current task in $p_i$ */
18         $FP$ = $FP \setminus P_i$; /* Remove $P_i$ from set FP */
19     else
20         $PBP_i$ = $PBP_i - 1$; /* Decrement remaining time */
21         if ($PBP_i$ == 0) then FP = FP $\cup$ {$P_i$}; /* Add $P_i$ to
             the set of free (available) processors */
22             $PL_i$ = FALSE; * Set $P_i$ back to free; */
23         FT = FT $\cup$ $T_j$ /* Add $T_j$ to set $FT$ of finished tasks */
24         $T = T \setminus T_j$; /* Delete $T_j$ from set $T$ */
25 if $|FT| \neq |\hat{\tau}|$ then
26     # Store the ALAP order;
27     Return FALSE;
28 else
29     Return TRUE;
```

Task Mapping and Execution (line 9 to 29): *Sched-Gen* assigns the task with no predecessors to a separate processor. Then it continues to consider tasks only when all its predecessor task(s) finish(es) their executions. Such task to processor assignments eventually enable that the beginning of the task will be the latest finishing time of its predecessors. In case, if a task has a single predecessor, then *DELICIOUS* can start to consider the task right after the finishing time of its predecessor. When a task has multiple predecessors, *DELICIOUS* considers the predecessor which has the latest finishing time. The successor task may be assigned to the same processor assigned to its predecessor with the latest finishing time. All tasks executing at a given time, run in parallel in the available processors. A task (say, T_j) mapped to a processor (say, p_i) will continue its execution until the execution requirement of the task is finished. The variable PBP_i denotes the "Processor Busy Period", which in turn provides the remaining execution requirement of T_j in p_i and thus, PBP_i becomes zero when T_j finishes its execution (line 20 to 21). After a task finishes its execution, it will be added to the set FT and will be removed from T (line 22 to 23). The set FT is finally stored in the dispatch table. The above processes of task mapping and execution continue iteratively either until all tasks in T complete their executions, or the deadline D_{PTG} is encountered. In line 24 to 29, *DELICIOUS* will check whether the number of finished tasks (FT) is equal to the number of tasks given in the input set T. Any mismatch will infer an incomplete schedule, otherwise, it will denote a successful one and *DELICIOUS-Offline* will return TRUE.

Our heuristic algorithm is associated with a few carefully selected, restricted design choices, that assist in controlling the complexity. It can be observed, that distinct schedules can be generated with each task (T_i), assigned to any of the available processors (P), with T_i being actually scheduled in any of the designated processors. Hence, the number of schedules depends on the number of tasks and processors. The schedule with enhanced accuracy could be any one of the subsets of the schedules that satisfy precedence, resource and timing constraints. However, to limit the complexity of this compute-intensive problem, our heuristic uses a phase-based approach. At first, it generates an accuracy maximized schedule, by restricting all tasks to their respective highest versions. Given the order and task-to-processor assignments as provided by the first phase, the task-versions are adjusted in the second phase, whilst meeting the deadline.

TABLE 2: Parameters and their values, for example task-set

Tasks	M_i (#cycles)	O_i (#cycles)	Tasks	M_i (#cycles)	O_i (#cycles)
T_1^1	4	2	T_4^1	20	7
T_2^1	10	5	T_4^2	20	12
T_2^2	10	8	T_5^1	19	2
T_2^3	10	10	T_5^2	19	6
T_3^1	10	5	T_5^3	19	14
T_3^2	10	7	T_6^1	10	2
T_3^3	10	10	T_6^2	10	4

Example *DELICIOUS-Offline*: Let us consider a representative example with the task-set given in Table 2, which is pictorially represented in Figure 2. These tasks have to be scheduled on two processors ($m = 2$), with a deadline $D_{PTG} = 70$ time units. In Figure 3[A], we have shown that, if the tasks are scheduled only with their respective highest versions, this will lead to deadline failure. Hence, by choosing different versions of the tasks, our algorithm generates the feasible schedule, which is depicted in Figure 3[B]. Here, T_2 and T_6 are executed with lower versions to satisfy the deadline. Our total obtained QoS value is 48.

Theorem 1. *The amortized complexity of DELICIOUS-Offline (Algorithm 1 to 3) is $\mathcal{O}(\frac{n}{D_{PTG}})$ per time-slot.*

Proof. Algorithm 1 is the heart of *DELICIOUS-Offline*. A step-wise analysis of computational overhead of Algorithm 1 due to the called functions/algorithms is as follows:

1) **ALAP Time Calculation:** Complexity of Algorithm 2 for calculating ALAP resembles the complexity of topological sorting of a DAG. Hence, complexity of Algorithm 2 can be written as $\mathcal{O}(n + |E|)$, where $n = |T|$.
2) **Schedule Generation:** We determine the complexity of Algorithm 3 by deducing the overheads of the individual steps: *(1)* line 2, 3, and 5 can be performed in constant time, and complexity of ALAP time is considered

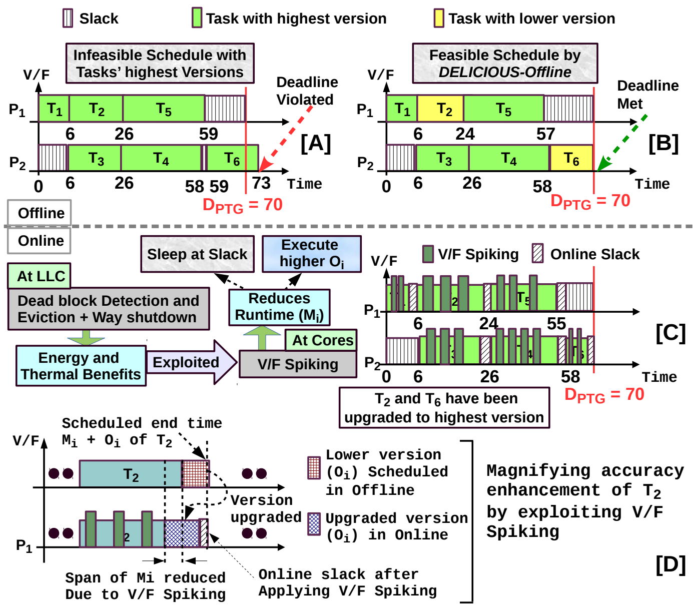
Slack
Task with highest version
Task with lower version
Infeasible Schedule with Tasks' highest Versions
Deadline Violated
V/F
P1
P2
T1 T2 T5
T3 T4 T6
6 26 59
0 6 26 58 59 73 Time
DPTG = 70
[A]
Feasible Schedule by DELICIOUS-Offline
Deadline Met
6 24 57
0 6 26 58 Time
DPTG = 70
[B]
Offline
Online
At LLC
Dead block Detection and Eviction + Way shutdown
Energy and Thermal Benefits
Exploited
V/F Spiking
At Cores
Sleep at Slack
Execute higher Oi
Reduces Runtime (Mi)
V/F Spiking
Online Slack
6 24 55
0 6 26 58 Time
DPTG = 70
[C]
T2 and T6 have been upgraded to highest version
Scheduled end time Mi + Oi of T2
Version upgraded
Span of Mi reduced Due to V/F Spiking
Online slack after Applying V/F Spiking
Lower version (Oi) Scheduled in Offline
Upgraded version (Oi) in Online
Magnifying accuracy enhancement of T2 by exploiting V/F Spiking
[D]

Fig. 3: Generated Schedule and online LLC induced *V/F spiking* (not to scale).

as $\mathcal{O}(n + |E|)$. Similarly, 8 can also be done in constant time. *(2)* The *for* loop (line 9 to 24) executes for each time step up to D_{PTG}. Inside this *for* loop, all individual operations consume an overhead of $\mathcal{O}(1)$. Thus, Algorithm 3 has a complexity of $\mathcal{O}(n+|E|)+\mathcal{O}(D_{PTG})$, which can be written as $\mathcal{O}(n+D_{PTG})$ for any standard graph.
3) *DELICIOUS-Offline*: Operations at line 1 of Algorithm 1 can be done in constant time. Each iteration of the *while* loop (line 2 to 10) calls Algorithm 3 to check the feasibility of the schedule that could be generated. Let us assume, K is the maximum number of possible versions for given tasks. Hence, it may be concluded that the *while* loop iterates at most K times. All other steps within the *while* loop take constant time. Thus, the complexity of this algorithm is dominated by the overhead of Algorithm 3. Finally, the overall complexity of *DELICIOUS-Offline* becomes $\mathcal{O}(K \cdot (n + D_{PTG}))$.
4) The amortized complexity of *DELICIOUS-Offline* is $\mathcal{O}(\frac{n}{D_{PTG}})$, where K typically consumes a small value.

□

6 *DELICIOUS-Online* Phase

To improve the accuracy or energy/thermal efficiency of the generated schedule, the selected V/F setting can be changed dynamically, but that might cause deadline failures, if not managed carefully. *DELICIOUS-Online* attempts to reduce core-temperatures by employing a dynamic LLC resizing that generates on-chip thermal buffers by shutting down cache ways with close vicinity to the cores (see Figure 3[C]). Such gained thermal benefits are traded off by a TDP cognizant V/F scaling of the cores, named here as *V/F Spiking*, that reduces the execution length of the tasks. *DELICIOUS-Online* uses this performance increase either to improve task accuracy while the core-temperature is kept in check, or to enhance energy and thermal efficiency by power gating the core (sleep mode) during generated slack. The possible task level changes by *DELICIOUS-Online* is illustrated in Figure 5, however, we magnified the *V/F Spiking* induced version upgrade for a task (T_2) in Figure 3[D]. Our LLC resizing selectively evicts dead blocks by periodic runtime analysis and trims LLC to improve the energy/thermal efficiencies without any noticeable performance impact.

6.1 Detecting Dead Blocks and Thermal Management at LLC

It is a well known fact that much of the data stored in the LLC is dead, i.e., the data will never be accessed before being evicted. In fact, a substantial amount (more than 80%) of all cache blocks at any particular time are dead as well as dead on arrival (DOA) [18], [31]. Hence, proactive eviction of dead blocks can offer a significant amount of spare cache space to the current application, which can be either used for more live blocks to enhance performance, or turned off to save energy. However, as the LLC is the final defense before approaching off-chip accesses, dead block detection and eviction should be done prudentially to maintain performance.

Detecting dead blocks at the block level granularity requires individual counters for each LLC block, where the size of individual counters can incur implementation overheads. To simplify our implementation and by considering time-criticality, we decided to detect only DOA blocks and to eventually evict them. We employ a single bit, called the *Dead_bit*, to track if a block is DOA. When a block is brought into the cache, the bit is set and is cleared if it is further accessed. We periodically check the *Dead_bit* and evict the block if the bit is still set. The check is performed one block at a time, iterating through all blocks within the predetermined period. Note that, for checking of the dead-bits and eviction of the dead blocks, a small time-slice is reserved at the end of each period, called back up period ($BackPer$). For our baseline 16-way set-associative and 4MB LLC, the storage overhead for implementing the *Dead_bit* is negligible at around 0.2%.

After detecting the dead blocks, *DELICIOUS* proactively evicts them from the LLC and turns off LLC ways to generate on-chip thermal buffers and to reduce core-temperature in its vicinity [11], [12]. Basically, the temperature of any on-chip component is guided by the basic *superposition and reciprocity* principle of heat transfer, which is driven by three factors: *(1)* the component's own power consumption, *(2)* heat abduction by ambient, and *(3)* conductive heat transfer with its peers [45]. Hence, prudential selection of these LLC-ways for shutting down on-the-fly can potentially reduce the chip temperature [12], by *(a)* curtailing its own power consumption and *(b)* incorporating heat transfer with the peers at the generated on-chip thermal buffers, while maintaining performance. As a significant number of LLC entries are DOA, which, if evicted, generates a large LLC portion as spare. But, such proactively generated empty locations might be scattered throughout the LLC, which has to be compacted to enable power gating of a complete cache way. This will generate continuous large thermal buffers, which will help in reducing temperature of the adjacent cores. Hence, we incorporate a simple but effective block swapping mechanism, discussed later, that prioritizes invalidation over write-back, and eventually empties an LLC way at the edge of the LLC bank before turning it off. By periodically monitoring the DOA blocks, and availability of the spare cache space after eviction, *DELICIOUS-Online* dynamically decides the number of LLC ways that can be power gated.

6.2 V/F Spiking: Effects and Amelioration

Increasing the V/F for a short duration, so called *V/F Spiking*, can enhance results accuracy if the core temperature can be kept in check by addressing the following issues:

- When should *V/F Spiking* be triggered?
- How long can the core maintain the increased V/F?

To answer these questions, one should consider the dynamic and leakage power consumption of the cores at different V/F settings and temperatures, along with the TDP of the cores. During task execution, *DELICIOUS* evenly divides the entire execution span into multiple periods, where at the end of each period, a decision on *V/F Spiking* will be taken. At the end of a period, if the core temperature is detected to be sufficiently below the critical temperature, then the power consumption of the core is evaluated to determine if an increased V/F that can be maintained without violating the power constraint. The dynamic power consumption (Dyn_{pow}) at the *target increased V/F* is derived by employing the following equation: $Dyn_{pow} = \alpha \cdot C \cdot {V_{dd}}^2 \cdot f$, where α and C are circuit related constants, and V_{dd} and f represent the supply voltage and core-frequency, respectively. By considering the current temperature (T) and the *target increased voltage*, the leakage consumption ($Leak_{pow}$) of the core can be derived at the end of the period through the following equation: $Leak_{pow} = A_1 \cdot T^2 \cdot e^{A_2 \cdot V_{dd} + A_3} + A_4 \cdot e^{A_5 \cdot V_{dd} + A_6}$, where, A_1 to A_6 are technology dependent constants. *DELICIOUS* inspects the available V/F levels and selects the maximum possible V/F setting for the upcoming period so that TDP is not violated during the next period. The span of a period can be determined empirically or from processor characteristics, during which the core temperature can be assumed to remain unchanged.

Maintaining a higher V/F setting for a period of time increases the core temperature, resulting in an increase in leakage power, which in turn generates heat in a self-reinforced cycle and can potentially affect the functional correctness of the chip. Employing an analytical formulation that estimated the generated heat from the power values can be a solution to determine the duration of the increased V/F residency [48]. But, the dynamic LLC resizing of *DELICIOUS-Online*, which significantly impacts the core thermal status, needs to be accounted for to correctly estimate the temperature, where the LLC resizing depends on the application's cache access behavior. In fact, our TDP based mechanism safeguards the core from thermal overshoot, but analytically determining the duration of the increased V/F residency might be unable to exploit the thermal benefits offered by LLC resizing. Hence, *DELICIOUS-Online* monitors the core temperature periodically using thermal sensors. Once the core temperature reaches the maximum threshold ($Temp_{Max}$), the V/F is reduced to the level at which the task is scheduled, and thus the duration of *V/F Spiking* is determined dynamically.

6.3 Proposed Online Technique

DELICIOUS-Online consists of two modules, the *LLC Resizing* module, which is implemented at the LLC controller for each LLC bank (discussed in Sec. 6.3.1), and the *V/F Spiking* module, which is implemented at the controller of the cores (discussed in Sec. 6.3.2). We illustrate the technique of *DELICIOUS-Online* in Algorithm 4. A complete schedule of the task-set is generated offline, called a $Frame$, details of which are kept in the dispatch table, where timing parameters of the tasks are converted into cycles prior to insertion. As long as all tasks are not selected from the Dispatch Table, each task (T_i) within a $Frame$ is fetched as per the schedule and the execution is initiated (line 1-4). For each LLC bank, Algorithm 5 is executed simultaneously with respect to each other at the LLC controller, to create on-chip thermal buffers by prudentially managing dead blocks (line 5-6). This gained thermal benefits are traded off to improve accuracy by employing *V/F Spiking* at each core during task execution (line 8-10). Note that, Algorithm 6 is executed at the respective core controllers, and is transparent to Algorithm 5.

Algorithm 4: *DELICIOUS-Online* Mechanism

```
1  for each Frame do
2      for all $T_i$ in Dispatch Table do
3          Get schedule details of $T_i$ from the Dispatch Table;
4          Fetch $T_i$ and start execution;
5          for each LLC bank do
6              Call Algorithm 5;
7              # Execute simultaneously at each bank;
8          for each Core do
9              Call Algorithm 6;
10             # Execute simultaneously at each core;
```

6.3.1 LLC Resizing Technique

DELICIOUS-Online is primarily built on the *LLC Resizing* mechanism that stimulates thermal efficiency of the cores adjacent to the power gated LLC portions. Figure 4 depicts the effects of power gated ways by illustrating the heat transfer from its adjacent cores. Before gating the ways, *DELICIOUS-Online* proactively evicts the dead blocks from the LLC by prudentially selecting them. After eviction of these dead blocks, a number of cache ways will be made empty by employing a swapping based compaction technique within each individual set. Once the selected way(s) is(are) empty, it is power gated.

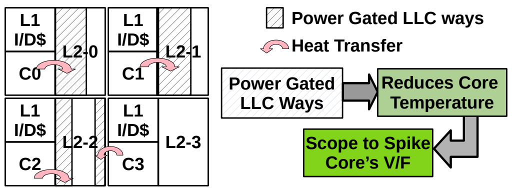

Fig. 4: Power Gated LLC ways offer scopes for V/F Spiking.

Algorithm 5: *LLC Resizing*

```
Input: $POWER\_DOWN$, $POWER\_UP$, $Limit$, $BackPer$, $Floorplan$
1  while A task ($T_i$) is being executed do
2      if $Curr\_Interval - BackPer$ is over then
3          for each LLC bank $B$ do
4              $ratio[B] = \frac{\#misses(B)}{\#accesses(B)}$;
5              if $ratio[B] < POWER\_DOWN$) and ($\#Off\_ways[B] < Limit$) then
6                  #Select a way $W$ as victim, which will be turned off and is in proximity to a core;
7                  for each set $S$ do
8                      for each block $blk$ having $\#Dead\_bit[blk] == 1$ do
9                          if $blk$ is clean then
10                             #Invalidate the block;
11                         else
12                             #Write it back off-chip;
13                     if $W$ at $S$ is not empty then
14                         if $S$ has at least an empty location then
15                             #Select an empty location, and move block from $W$;
16                         else
17                             if $S$ contains a CN block then
18                                 #Invalidate the block;
19                             else
20                                 if $W$ has NMRU block then
21                                     #Write it back;
22                                 else
23                                     #Select an NMRU block in $S$, and write it back;
24                             #Move block from $W$;
25                 #Power-gate $W$, and $\#Off\_ways[B]++$;
26             else
27                 if ($ratio[B] > POWER\_UP$) and ($\#Off\_ways[B] \geq 1$) then
28                     #Turn on an LLC way, and $Off\_ways[B]--$;
29         #Execute the task normally upto end of $Curr\_Interval$;
30     else
31         #Execute the task normally;
```

The entire *LLC Resizing* mechanism is illustrated in Algorithm 5. The whole task execution span is evenly divided into multiple time-intervals ($Curr\_Interval$), and a small time-span, $BackPer$ (back up period), is taken from the end of each $Curr\_Interval$ during which all the resizing related operations are performed. On completion of each $Curr\_Interval - BackPer$, the current performance of the bank (B) is determined by its miss ratio ($ratio[B]$ line 4). If the miss ratio is less than a preset threshold ($POWER\_DOWN$) and the number of turned off LLC ways ($\#Off\_ways[B]$) is within a preset limit ($Limit$), then a way (W) adjacent to a core is selected as the victim (line 5 to 6). The location details of the LLC ways and their adjacency to the cores are determined from the $Floorplan$ of the CMP, which is an input to our algorithm [11], [12]. For each set (S) the presence of dead blocks (blk) is determined by inspecting if their respective $Dead\_bit[blk]$ is set (line 8). If a dead block is clean, it is invalidated, else it is written back to the main memory (line 9 to 12).

On completion of the dead block eviction process, a set might not have an empty location at the victim way W (line 13). Hence, set S will then be checked if there is any empty location, and once an empty location is found, the block will be moved to there from W (line 15). However, if S does not have any empty location at the moment, then search for a clean NMRU (CN) block in S is performed, and will be invalidated on its presence. Otherwise, an NMRU block is selected from W, if available, or from any other random location of S and will be written back subsequently. Next, the block from W will be moved to this empty location (line 17 to 24). Once W is empty for all sets, it will be gated with updating $\#Off\_ways[B]$ (line 25). If at the end of a $Curr\_Interval$, $ratio[B]$ is higher than a preset threshold ($POWER\_UP$), and B has at least one way turned off, a way will then be turned on (line 27 to 28). No LLC reconfiguration is permitted within $Curr\_Interval - BackPer$ and on completion of resizing process (line 29 and line 31).

The block swapping needs to be performed by accessing the peripheral circuitry of the bank, performance of which is hence limited by the number of ports available per bank. However, the power and performance overheads incurred by this swapping mechanism are negligible [12]. Additionally, our LLC resizing technique can serve the outstanding cache requests during $BackPer$, unlike prior art [11]. The only difference is that, on an eviction caused by a cache miss, the selected way to be evicted cannot be the victim way. However, the performance impact of LLC resizing is also included in our simulation.

6.3.2 *Proposed V/F Spiking*

LLC resizing technique can potentially reduce temperature (hence the leakage power) of the cores adjacent to the gated LLC ways. Reduced core temperature therefore offers enough room for maintaining the increased V/F through *V/F Spiking* for a certain amount of time while keeping the core temperature below the critical value. Our proposed Algorithm 6 shows how *DELICIOUS-Online* exploits the thermal benefits of Algorithm 5 to enhance core V/F without violating the thermal constraint.

Algorithm 6 takes $Temp_{Max}$, TDP, and T_{Lim} as inputs, where $Temp_{Max}$ is the maximum allowable temperature for a core. We set $Temp_{Max}$ to 2 °C lower than the critical temperature of the core, to ensure that the core temperature will never reach at the critical value. During task execution, at the end of each $Interval$, each core temperature ($Temperature[C]$) will be observed (line 2 to 5). If the $Temperature[C]$ is lower by T_{Lim} than the $Temp_{Max}$, leakage power of the core ($Leak_{pow}[C]$) will be computed by considering $Temperature[C]$ and supply voltage (line 6). Next, the highest possible viable V/F level (V_H/F_H) is determined, so that total (calculated) power consumption ($Dyn^{H}_{pow}[C]$ + $Leak_{pow}[C]$) is not violating the TDP (line 7). Our algorithm also considers the power of on-chip voltage regulator (VR_{Pow}). On availability of such V_H/F_H, the core's V/F is set at V_H/F_H, and the task execution will be resumed (line 8 to 9). Executing tasks at higher V/F leads to early completion, that results into change in the generated schedule. Basically, higher V/F can potentially execute more number of cycles for a certain time-span than the execution at the V_{sched}/F_{sched}. Hence, we employ a counter (Cyc_Ctr) to keep track of the cycles executed at the higher V/F (line 10). Note that, the input T_{Lim} safeguards the core from any potential chattering effects in V/F by allowing *V/F Spiking* only when the core temperature is sufficiently below the $Temp_{Max}$.

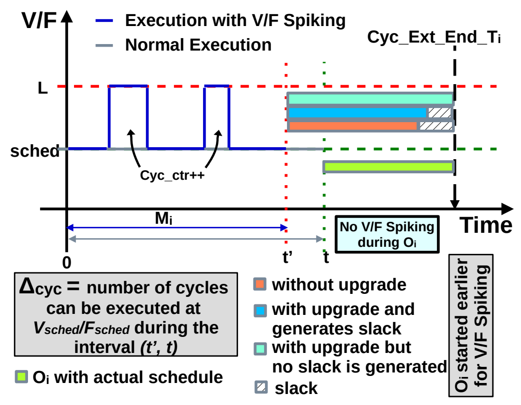
V/F
Execution with V/F Spiking
Normal Execution
Cyc_Ext_End_Tᵢ
L
sched
Cyc_ctr++
Mᵢ
No V/F Spiking during Oᵢ
Time
0
t'
t
Δcyc = number of cycles can be executed at Vsched/Fsched during the interval (t', t)
Oᵢ with actual schedule
without upgrade
with upgrade and generates slack
with upgrade but no slack is generated
slack
Oᵢ started earlier for V/F Spiking

Fig. 5: *V/F Spiking* and version upgrade.

During task execution, the core temperature will be monitored continuously, and once the $Temperature[C]$ reaches at $Temp_{Max}$, the V/F will be lowered to $V_{sched}/F_{sched}[C]$ (line 15). To keep track of the extra cycles completed at the higher frequency, Cyc_Ctr is exploited at the end of each V/F spike. By computing the elapsed time along with considering $V_{sched}/F_{sched}[C]$, the amount of extra cycles is derived (line 14 to 17). This cycle surplus during executing M_i is stored at Δ_{cyc}, which will be used next for O_i execution. We illustrate *V/F Spiking* process in Figure 5 at the task level granularity, that depicts when Cyc_Ctr is updated and how *V/F Spiking* helps in finishing the task early.

As per our example in Figure 5, M_i completes at t' with *V/F Spiking*, where its scheduled completion time was at t ($t' < t$). Hence, to execute O_i, the time left is the summation of Δ_{cyc} (which can be executed during interval (t', t) at $V_{sched}/F_{sched}[C]$) and the cycles left before execution of the next task, which we termed as extended end time of T_i ($Cyc\_Ext\_End\_T_i$) (line 18). Note that, for the sink task, $Cyc\_Ext\_End\_T_i$ will be set at the end of the current $Frame$. However, if the highest version of T_i is not scheduled earlier, a checking is performed if O_i can be upgraded (line 19 to 23). After selecting the best possible O_i, the execution will be started with $V_{sched}/F_{sched}[C]$ (line 26). Upgrading O_i may generate $slack$ before completion of $Cyc\_Ext\_End\_T_i$ (line 24 to 25), which can be utilized to power gate the core for improving energy/thermal efficiency. All the possible cases regarding upgrading O_i are depicted in Figure 5. By employing a counter and considering the processor's $Break\_Even\_Time$ (given as an input), the span of power-gate is traced, and the core will be turned on (line 31) before the starting time of the next task/frame.

6.4 Hardware Mechanism

Both Algorithm 5 and 6 can be implemented separately at the respective controllers. The way-shutdown logic at the LLC controller adopts power gating [40] at the way-level granularity of the LLC. Power gating is a conventional circuit based technique integrated with caches as well as cores in modern CMPs [4], [34]. By exploiting conventional control bits (e.g., valid bit, dirty bits, etc.) and the existing performance monitoring counters at the LLC [21], the $ratio$ and $Dead\_bit$ can be periodically monitored for LLC resizing. Moreover, implementing $Dead\_bit$ will not incur

Algorithm 6: *V/F Spiking*

```
Input: $Temp_{Max}$, $TDP$, $T_{Lim}$, $Break\_Even\_Time$
1  $\Delta_{Cyc} = 0$ ;
2  while $M_i$ is being executed do
3      if $Curr\_Interval$ is over then
4          for each core C in parallel do
5              if $Temperature[C] < Temp_{Max} - T_{Lim}$ then
6                  #Compute $Leak_{pow}[C]$;
7                  #Get highest V/F, $V_H/F_H$, so that
                     $Dyn^H_{pow}[C] + Leak_{pow}[C] + VR_{Pow} < TDP[C]$;
8                  if Such $V_H/F_H$ exists then
9                      $V/F[C]$ = $V_H/F_H$;
10                     #Start execution and start increasing
                         $Cyc\_Ctr$;
11     else
12         #Execute the task normally;
13         if $Temperature[C] == Temp_{Max}$ and $V/F[C] > V_{sched}/F_{sched}[C]$ then
14             $Time\_elaps = \frac{Cyc\_Ctr}{F[C]}$ ;
15             #Set $V/F[C] = V_{sched}/F_{sched}[C]$;
16             $\Delta_{Cyc} += Cyc\_Ctr$ - ($Time\_elaps \times F_{sched}[C]$) ;
17             $Cyc\_Ctr = 0$ and stop incrementing $Cyc\_Ctr$ ;
18 $Cyc\_rem_{O_i} = \Delta_{Cyc} + (Cyc\_Ext\_End\_T_i - Cyc\_End\_M_i)$ ;
19 if Highest $O_i$ is not scheduled then
20     # Call the function that returns optional part with highest
         possible accuracy which can run within $Cyc\_rem$ ;
21     $O_i$ = $get\_O_i(T_i, Cyc\_rem)$ ;
22     if $O\_i$ then
23         #Fetch the $O_i$ ;
24 if $Cyc\_rem_{O_i} > Cyc\_O_i$ then
25     $Slack$ = $Cyc\_rem\_O_i - Cyc\_O_i$ ;
26 #Start Execution of $O_i$ at $V_{sched}/F_{sched}[C]$ ;
27 if $O_i$ is finished and $Slack > Break\_Even\_Time$ then
28     #Power gate the core ;
29     while $Slack > 0$ do
30         $Slack$--;
31     #Turn on the core ;
```

any noticeable overheads, as discussed earlier. To efficiently scale V/F at the cores, on-chip voltage regulators [17] can be attached, which are also common in contemporary CMPs. Note that, on-chip thermal sensors will be used to observe the core temperature on-the-fly.

7 Evaluation

In this section, first we show the efficacy of *DELICIOUS-Offline* approach (Sec. 5) followed by the benchmark based evaluation of the *DELICIOUS-Online* (Sec. 6).

7.1 *DELICIOUS-Offline*

First, we define **N**ormalized **A**chieved **Q**oS (NAQ), which is the ratio between the actually achieved QoS for the PTG, and the maximum achievable QoS by executing the highest versions of all tasks. We formulate NAQ as: $NAQ = \frac{\sum_{i=1}^{n} Acc_i^j}{\sum_{i=1}^{n} Acc_i^{k_i}}$, where k_i represents the highest version of task T_i. Next, we model a multicore along with the task-set:

- *Processor System:* A homogeneous multicore platform equipped with 4 Intel $x86$ cores (i.e., $m = 4$) has been considered. The TDP of the each core is scaled and set as 10.5W, by considering the Intel Xeon's datasheet [1] and the runtime core power is obtained through McPAT [30].
- *Task-set:* The task characteristics have been taken from a prior technique, *Prepare* [10], that framed tasks by using PARSEC benchmark applications. The total execution requirement of a PTG (C_{PTG}) is 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 presented as $\frac{C_{PTG}}{D_{PTG}}$. The average utilization of a PTG is taken from a normal distribution, by considering a normalized frequency of 0.6. Given the PTG's utilization, we further obtain the total utilization of the system (Sys_{uti}) by summing up the utilization of all PTGs. Given the Sys_{uti}, the total system workload (Sys_{WL}) / system pressure can be derived by: $Sys_{WL} = \frac{Sys_{uti}}{m}$. For a given Sys_{uti}, all of our PTGs have been generated by following the method proposed in *Prepare* [10]. Given a Sys_{WL}, a set of DAGs have been created. The number of DAGs (ρ) within a set can be calculated as: $\rho = \frac{m \times Sys_{WL}}{U_i}$. In our generated PTGs, the minimum number of tasks is equal to 5 and the maximum number of tasks is set to 20. For each PTG in the set, the number of tasks have been generated randomly within a preset limit. Note that, as the individual U_i of a DAG is lower than the given Sys_{WL}, the number of DAGs (ρ) within the set will always be higher than m.
- *Task Temporal Parameters:* For each T_i, based on which portion of the len_i is considered as the mandatory portion (M_i), we consider the following cases [15]: *(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).

Scalability analysis of *DELICIOUS-Offline*. Figure 6 depicts the mean solving time per number of tasks in each PTG while applying the scheduling heuristic of *DELICIOUS*, and the ILP based scheduling of *Prepare* [10]. This result shows that, our proposed heuristic has better scalability with the number of tasks than the ILP based algorithm. With significantly lower running time, this heuristic generates nearly optimal schedule like ILP. In fact, with 20 tasks, the ILP based scheduling has almost 4× higher execution time than our scheduling heuristic.

Effects of System Workload. Figure 7 depicts the NAQ achieved by *DELICIOUS-Offline* for different values of Sys_{WL}. The NAQ is derived by running each of the DAGs that belongs to the set. Then, we have taken the average over the obtained individual NAQ values. We observed that, *DELICIOUS* is able to achieve 80% QoS, when the system workload is low. However, the QoS is reduced by 20% on average, when the workload is scaled up by 40%. Other two insightful observations can also be derived from this figure. Firstly, as the system workload is increased in order to maintain the number of DAGs (ρ) in the system, the individual U_i also increases and this eventually contributes to low *NAQ* values. This happens as increasing U_i results in higher execution length of each task and thus the possibility of obtaining sufficient free slots in the scheduling period

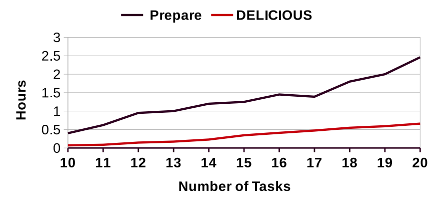

Fig. 6: Running time: *Prepare* (ILP) vs. *DELICIOUS-Offline*.

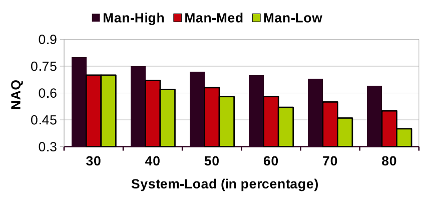

Fig. 7: Change in QoS (NAQ) for different system loads.

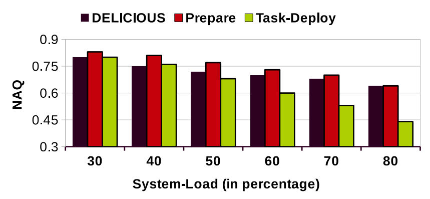

Fig. 8: Change in QoS (NAQ): Comparison with Prior Arts.

reduces within the deadline. Insufficient free slots in turn reduces the probability of obtaining feasible schedules by selecting higher tasks' versions.

Secondly, in case of man_{high}, the reduction in achieved *NAQ* is reduced comparatively lower than the man_{med} and man_{low}, while increasing the value of Sys_{WL}. This can be attributed to the fact that, when mandatory portions of the individual tasks are high, the length of the optional portions will be low. This results into the variance among the different versions of a task become less. Due to fewer variations among the optional parts of a task, there will be less impact on the achieved accuracy. On the other hand, in case of man_{low}, we observe that, the reduction in *NAQ* is higher than the other two, and man_{med} offers a performance between man_{high} and man_{low}. However, the *NAQ* sharply decreases while Sys_{WL} goes up. We also compared our strategy with prior arts, $Task\_Deploy$ [37] and *Prepare* [10] and the results are shown in Figure 8. Towards a fair comparison with $Task\_Deploy$, we computed the overall energy constraint based on the considered TDP of the experimental framework of *DELICIOUS*. This power limit is also used in case of *Prepare*. Next, we consider our comparison by uniformly choosing M_i of the tasks between 20% to 80% of len_i. As execution demand of individual tasks goes up (due to increase in Sys_{WL}), *DELICIOUS* maintains improved QoS by achieving higher NAQ than $Task\_Deploy$. *DELICIOUS* is able to maintain 70% QoS at 70% workload where $Task\_Deploy$ achieves 60% QoS. This is because the considered overall energy limit in $Task\_Deploy$ would scale up with the higher Sys_{WL}. Moreover, $Task\_Deploy$ also allows unlimited tasks migration, that incurs additional overhead. However, for all workloads, *Prepare* shows better NAQ among all policies due to employment of ILP based optimal scheduling, but, heuristic based strategy of *DELICIOUS-Offline* also offers a performance close to this optimal values, with a remarkably low computational time.

7.2 *DELICIOUS-Online*

7.2.1 Simulation Setup

In this work, a homogeneous tiled CMP having 4 tiles is simulated in the gem5 full system simulator [8]. Each tile has an $Intel\ x86$ Xeon 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. After collecting the periodic performance traces from gem5, it is sent to McPAT [30] to generate the power traces. Basically, we derive dynamic power consumption for individual on-chip components by executing McPAT. As McPAT assumes uniform on-chip temperature for estimating leakage power,

TABLE 3: V/F settings and Dynamic Power values for Intel $x86$ OoO core (at 22nm node)

V/F setting (V/GHz)	0.6/2.4	1.0/3.0	1.2/3.4	1.5/3.9
Dynamic Power (W)	3.759	4.498	5.214	5.942

TABLE 4: Temperature vs Leakage Power for Intel $x86$ OoO core (at 22 nm node)

Temperature (°C)	67	77	87	97	107	117
Leakage Power (W)	0.364	0.516	1.021	1.956	3.106	5.235

which is impractical, we compute the component-wise leakage power by considering the temperatures of individual on-chip components at the end of the last period [24], [25], [26]. Eventually, we derive the total power consumption from dynamic and leakage power estimations, the power values are sent to HotSpot 6.0 [48] towards generating temperature traces. Based on prior analyses [11], [12], the span of this periodic interval is set to 0.33 μs (i.e. 1.0M cycles at 3.0GHz frequency), during which we assume the temperature across the CMP is stable. We set $BackPer$ as last 5% time-span of the interval. 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. Our detailed system parameters used in the simulations by considering 22nm technology nodes are listed in Table 5.

Table 3 lists the V/F values for $Intel\ x86$ Xeon cores, for which power values are obtained from McPAT. The changes in leakage power for different temperatures are also obtained from McPAT and are shown in Table 4, where the leakage increases at higher rate at the higher temperatures. To simplify our online computation in Algorithm 4, we adopt piecewise linear approximation for each range of $10\,°C$ to compute leakage consumption at any temperature [11], [12]. In our simulation framework, each core runs at the Base V/F level with the effective frequency (f_{eff}) of 3.0GHz. For our experiments, we also consider another V/F magnitude (Med) between $Turbo$ and $Base$. Note that, a core can execute tasks in all of these V/F, however, core can maintain $Base$ V/F without any potential thermal threats, but the remaining two values are suggested to be maintained for particular time-spans, provided by the vendor. We set T_{Lim} (of Algorithm 6) as $4\,°C$.

To set $Curr\_Interval$, we evaluated nine PARSEC applications for DOA blocks on our baseline system with 0.5M $-$ 2.0M in 0.5M increments, by executing each application for 100M cycles within RoI, and the results are shown in Figure 9. The results show that, the cache access patterns for DOA blocks converge at 1.0M for most of the applications, which is hence considered here as $Curr\_Interval$, which is also in line with prior research [12]. For a 1.0M period-

TABLE 5: System parameters [CC: clock cycle]

Parameter	Value	Parameter	Value
ISA	Intel x86	L1-I	64KB, 4Way, 3CC
#Cores (type)	4 ($Xeon$)	L1-D	64KB, 4Way, 3CC
Base V/F ($Base$)	1.0V, 3.0GHz	L2	1MB, 16Way, 12CC
Med. V/F (Med)	1.2V, 3.4GHz	Cache	LRU, 64B blocks
Turbo V/F ($Turbo$)	1.5V, 3.9GHz	#Cache-Levels	2
VR-Speed	20 mV/ns	Cache model	SNUCA
Power_gate_overhead	60 ns	DRAM latency	70 ns
ROB Size	200	Technology	22 nm
Dispatch/Issue width	8	Ambient Temp.	47 °C

length, our evaluation shows that $89-93\%$ of the LLC blocks are DOA, on average. Such salient presence of DOA blocks further justifies the sufficiency of using *Dead_bit* to detect the dead entries in Algorithm 5.

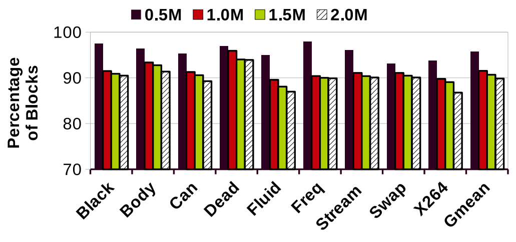

Fig. 9: Amount of DOA Blocks for different $Curr\_Interval$.

7.2.2 Task-set

Our tasks are generated by using PARSEC benchmark suite [7], which can be fitted in an AC based paradigm through the loop perforation technique [3], [43]. 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 and the latter one is representing O_i. For creating multiple versions of O_i, the latter application will have different executable files, with various execution lengths. We have constructed each M_i and O_i by using two copies of two different PARSEC applications, for example, for a task, T_1, M_1 is framed by two copies of $Black$, whereas the O_1 is constructed by two copies of $Body$. The task-set is detailed in Table 6, where the execution lengths (Exec_Length) are given in million cycles in the region of interest (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 200M cycles concurrently in our considered CMP 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. Note that, the execution length of each task in Table 6 is set by scaling the task lengths given in Table 2. The versions of O_i selected by *DELICIOUS-Offline* (*Sel. Oi [EL]*) are also given in Table 6. We have used a 4 core based CMP, where each task's M_i and O_i run on 2 cores. Two cores of this CMP implies a single processor-core, P_i in Figure 3.

7.2.3 LLC Resizing, Peak Temperature, and Performance Improvements

DELICIOUS-Offline schedules the task-set where T_2 and T_6 are scheduled with lower O_i. Both of these tasks' M_i's consist of memory intensive PARSEC applications (*stream* and *x264*). Presence of dead blocks at the LLC for *stream* and *x264* enables Algorithm 5 to turn off a number of cache ways, that assists Algorithm 6 to maintain $Turbo$ V/F for a

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 (ELs) are in million cycles. *Black (2)* implies 2 copies of *Black*, which is the same for others.

Tasks	Benchmarks (M_i, O_i)	EL ([M_i], [O_i])	Sel. Oi [EL]
T_1	*Black (2), Body (2)*	[80], [40]	#1 [40]
T_2	*Stream (2), Can (2)*	[200], [100, 160, 200]	#2 [160]
T_3	*Ded (2), Fluid (2)*	[200], [100, 140, 200]	#3 [200]
T_4	*Fluid (2), Freq (2)*	[400], [140, 240]	#2 [240]
T_5	*Body (2), X264 (2)*	[380], [40, 120, 280]	#2 [280]
T_6	*X264 (2), Ded (2)*	[200], [40, 80]	#1 [40]

longer time. We also experimented with a Med V/F level, higher than $Base$ V/F but lower than $Turbo$, by running the core at this level during *V/F Spiking*. The cores can execute tasks at Med for longer time, as the rate of temperature change at this level is slower than $Turbo$. Our simulation results in Figure 10 show the reduction in execution lengths of each task for Med and $Turbo$, where the offered thermal benefits at Med is however compensated by the performance benefits of the $Turbo$. Both Med and $Turbo$ offer almost similar performance benefits by reducing execution length 8.5% and 8.2%, respectively, without violating the temperature threshold. However, the execution length for $Turbo$ is slightly higher for T_4, a memory intensive task, that is able to maintain $Turbo$ residency for a longer time at some initial execution phases, which results into higher temperature, and thus it lacks some chances of *V/F Spiking* later. In *DELICIOUS*, we have chosen $Turbo$ for executing tasks during *V/F Spiking*, however, one can also choose Med as a promising alternative.

Figure 11 shows the average and minimum LLC sizes maintained for each task, and the respective reductions in core temperature are also depicted. Algorithm 5 is able to reduce peak temperature by 5.8 °C on an average by leveraging the generated thermal buffers through gated LLC ways, that elongates the vendor defined span (of 10ms) remarkably by 7% on an average (Figure 10), at $Turbo$. Overall, *DELICIOUS-Online* improves QoS by executing all tasks at their highest version, and the reduction in execution span also generates slacks at the end of each task. The generated amount of online slacks are significant, which are in the range of $6.2 - 10.1\%$ of their actual execution span (generated offline) across the tasks. The updated versions and the amount of generated slacks are listed in Table 7. However, by employing LLC resizing induced *V/F Spiking*, *DELICIOUS-Online* noticeably improves achieved QoS (by *DELICIOUS-Offline*) of the task-set by 8.3%.

7.2.4 Comparison with Prior Works

We compared *DELICIOUS* with two recent prior works, *Prepare* [10], that refines the schedule (generated offline) by employing an LLC miss induced DVFS technique, and, *GDP* [33], that employs a threshold temperature based technique to apply DVFS. Figure 12 depicts how *DELICIOUS* outperforms the prior policies in terms of the maximum reduction in peak temperature of the cores during the slacks. The longer slack intervals in *DELICIOUS* offer a maximum

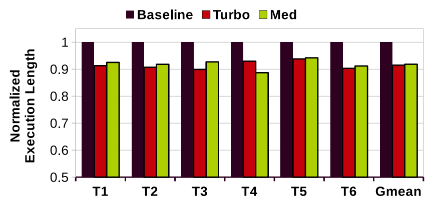

Fig. 10: Comparing Execution Length: Med and $Turbo$ during *V/F Spiking*.

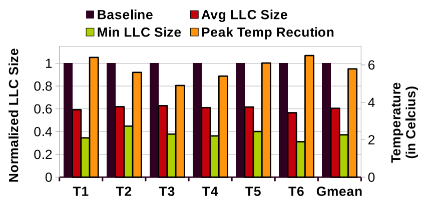

Fig. 11: Reduction in LLC size and Peak Temperature.

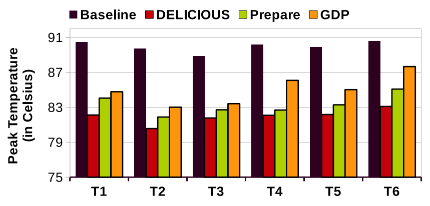

Fig. 12: Maximum Reduction in Peak Temperature at Slacks.

TABLE 7: Outputs of *DELICIOUS-Offline* and *Online*

Tasks	Mapped Core	Scheduled Version (Offline)	Updated Version (Online)	Amount of Slack
T_1	P_1	1	1	8.7%
T_2	P_1	2	3	9.3%
T_3	P_2	3	3	10.1%
T_4	P_2	2	2	7.05%
T_5	P_1	2	2	6.2%
T_6	P_2	1	2	9.7%
Improvement in Achieved QoS		8.3%		

TABLE 8: Comparison with Prior Works

Techniques	DELICIOUS	Prepare [10]	GDP [33]
Online QoS Scaled up	8.3%	5.3%	Not Applicable
Average Runtime Peak Temperature Reduction	5.8 °C	5.1 °C	4.9 °C

reduction of up to 9.2 °C, which is up to 7.8 and 6.7 °C for *Prepare* and *GDP*, respectively. Table 8 shows, *DELICIOUS* surpasses the prior techniques in terms of online QoS improvement, as eviction of dead blocks also plays a significant role in boosting up the performance along with the *V/F Spiking*. *Prepare* offers an online QoS improvement by 5.3%, which is 8.3% in case of *DELICIOUS-Online* (not applicable for *GDP*). In fact, our LLC resizing is also able to reduce core peak temperature by 5.8 °C, which is 5.1 and 4.9 °C for *Prepare* and *GDP*, respectively. The threshold temperature based DVFS in *GDP* scales down the core's V/F that does not allow thermal overshoot, whereas our *V/F Spiking* mechanism considers both TDP and critical temperature to prevent temperature overshoot with elongated time-span for $Turbo$ frequency. *Prepare*, on the other hand, controls peak temperature by introducing energy-adaptive DVFS at the cores.

8 Conclusion

Improving result-accuracy in AC based real-time paradigms without violating power constraints of the underlying hardware has recently become an active research avenue. Execution of the AC real-time applications is split into two parts: (i) *the mandatory part*, execution of which provides a result of acceptable quality, followed by (ii) *the optional part*, which can be executed partially or fully to refine the initially obtained result towards improving the result-accuracy without deadline violation. In this paper, we introduce *DELICIOUS*, a novel hybrid offline-online *scheduling strategy* for AC real-time dependent tasks. By employing an efficient heuristic algorithm, *DELICIOUS* first generates a schedule for a dependent AC task-set at a base processing frequency with an objective to maximize the results-accuracy, while respecting the system-wide constraints. At runtime, *DELICIOUS* next employs a prudential way on-off based LLC resizing induced thermal management to enhance the processing speed at the cores for a stipulated time-span without violating power budget, called as *V/F Spiking*, to reduce the tasks' execution lengths. The generated slack by the reduced execution length can be exploited either to enhance QoS further by dynamically adjusting the optional part or to reduce temperature by enabling sleep at the cores. In addition with surpassing the prior art, *DELICIOUS* offers 80% result-accuracy with our scheduling strategy, which is enhanced by 8.3% in online, while reducing runtime peak temperature by 5.8 °C on average within deadline, as shown by a benchmark based evaluation on a 4-core based CMP.

Acknowledgments

This work is supported by the UK Engineering and Physical Sciences Research Council (EPSRC) through grants $EP/X015955/1$, $EP/V000462/1$, and is also funded by *Marie Curie Individual Fellowship (MSCA-IF), EU (Grant Number* 898296*)*.

References

[1] "12th generation intel® core™ processor family," https://www.intel.com/content/www/us/en/products/docs/processors/core/core-technical-resources.html, accessed: 2022-03-28.
[2] "Overview information for intel turbo boost technology," https://www.intel.com/content/www/us/en/support/articles/000007359/processors/intel-core-processors.html, accessed: 2022-08-31.
[3] S. Achour and M. C. Rinard, "Approximate computation with outlier detection in topaz," *SIGPLAN Not.*, 2015.
[4] M. Arora *et al.*, "Understanding idle behavior and power gating mechanisms in the context of modern benchmarks on cpu-gpu integrated systems," in *HPCA*, 2015.
[5] H. Aydin *et al.*, "Optimal reward-based scheduling for periodic real-time tasks," *IEEE TC*, 2001.
[6] A. Bhuiyan *et al.*, "Energy-efficient real-time scheduling of DAG tasks," *ACM TECS*, 2018.
[7] C. Bienia *et al.*, "The PARSEC benchmark suite: Characterization and architectural implications," in *PACT*, 2008.
[8] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH CAN*, 2011.
[9] K. Cao *et al.*, "QoS-adaptive approximate real-time computation for mobility-aware IoT lifetime optimization," *IEEE TCAD*, 2019.
[10] S. Chakraborty *et al.*, "Prepare: Power-Aware Approximate Real-Time Task Scheduling for Energy-Adaptive QoS Maximization," *ACM TECS*, 2021.
[11] S. Chakraborty and H. K. Kapoor, "Exploring the role of large centralised caches in thermal efficient chip design," *ACM TODAES*, 2019.
[12] S. Chakraborty and M. Själander, "WaFFLe: Gated cache-ways with per-core fine-grained DVFS for reduced on-chip temperature and leakage consumption," *ACM TACO*, 2021.
[13] T. Chantem *et al.*, "Temperature-aware scheduling and assignment for hard real-time applications on mpsocs," in *DATE*, 2008.

[14] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *ISCA*, 2006.

[15] A. Esmaili *et al.*, "Energy-aware scheduling of task graphs with imprecise computations and end-to-end deadlines," *ACM TODAES*, 2019.

[16] ——, "Modeling processor idle times in MPSoC platforms to enable integrated DPM, DVFS, and task scheduling subject to a hard deadline," in *ASPDAC*, 2019.

[17] S. Eyerman and L. Eeckhout, "Fine-grained DVFS using on-chip regulators," *ACM TACO*, vol. 8, no. 1, 2011.

[18] P. Faldu and B. Grot, "Leeway: Addressing variability in dead-block prediction for last-level caches," in *PACT*, 2017.

[19] Y. Ge *et al.*, "Distributed task migration for thermal management in many-core systems," in *DAC*, 2010.

[20] ——, "A multi-agent framework for thermal aware task migration in many-core systems," *IEEE TVLSI*, 2012.

[21] B. Goel *et al.*, "Chapter two - techniques to measure, model, and manage power," ser. Advances in Computers (Elsevier), 2012.

[22] Z. Guo *et al.*, "Energy-Efficient Multi-Core Scheduling for Real-Time DAG Tasks," in *ECRTS*, 2017.

[23] ——, "Energy-efficient real-time scheduling of dags on clustered multi-core platforms," in *RTAS*, 2019.

[24] V. H. and S. V., "Energy-efficient operation of multicore processors by DVFS, task migration, and active cooling," *IEEE TC*, 2014.

[25] V. Hanumaiah *et al.*, "Maximizing performance of thermally constrained multi-core processors by dynamic voltage and frequency control," in *ICCAD*, 2009.

[26] V. Hanumaiah *et al.*, "Performance optimal online DVFS and task migration techniques for thermally constrained multi-core processors," *IEEE TCAD*, 2011.

[27] K. Kanoun *et al.*, "Online energy-efficient task-graph scheduling for multicore platforms," *IEEE TCAD*, 2014.

[28] J. Kong *et al.*, "Recent thermal management techniques for microprocessors," *ACM CSUR*, 2012.

[29] J. Lee and N. S. Kim, "Analyzing potential throughput improvement of power- and thermal-constrained multicore processors by exploiting DVFS and PCPG," *IEEE TVLSI*, 2012.

[30] S. Li *et al.*, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, 2009.

[31] C. Mazumdar *et al.*, "Dead page and dead block predictors: Cleaning TLBs and caches together," in *HPCA*, 2021.

[32] I. Méndez-Díaz *et al.*, "Energy-aware scheduling mandatory/optional tasks in multicore real-time systems," *International Transactions in Operational Research*, 2017.

[33] A. Mirtar *et al.*, "Joint work and voltage/frequency scaling for quality-optimized dynamic thermal management," *IEEE TVLSI*, 2015.

[34] S. Mittal, "A survey of architectural techniques for improving cache power efficiency," *Sustainable Computing: Informatics and Systems*, 2014.

[35] ——, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, 2016.

[36] L. Mo *et al.*, "Energy-quality-time optimized task mapping on DVFS-enabled multicores," *IEEE TCAD*, 2018.

[37] ——, "Approximation-aware task deployment on asymmetric multicore processors," in *DATE*, 2019.

[38] S. Narayana *et al.*, "Exploring energy saving for mixed-criticality systems on multi-cores," in *RTAS*, 2016.

[39] S. Pagani *et al.*, "Energy and peak power efficiency analysis for the single voltage approximation (SVA) scheme," *IEEE TCAD*, 2015.

[40] M. Powell *et al.*, "Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories," in *ISLPED*, 2000.

[41] R. Rao *et al.*, "An optimal analytical solution for processor speed control with thermal constraints," in *ISLPED*, 2006.

[42] J. Roeder *et al.*, "Energy-aware scheduling of multi-version tasks on heterogeneous real-time systems," in *ACM SAC*, 2021.

[43] S. Sidiroglou-Douskos *et al.*, "Managing performance vs. accuracy trade-offs with loop perforation," in *ACM SIGSOFT*, 2011.

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

[45] K. Stavrou and P. Trancoso, "TSIC: thermal scheduling simulator for chip multiprocessors," in *Proceedings of the 10th Panhellenic Conference on Advances in Informatics*, 2005, pp. 589–599.

[46] C. Tan *et al.*, "Approximation-aware scheduling on heterogeneous multi-core architectures," in *ASP-DAC*, 2015.

[47] H. Yu *et al.*, "Dynamic scheduling of imprecise-computation tasks in maximizing QoS under energy constraints for embedded systems," in *ASPDAC*, 2008.

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

[49] J. Zhou *et al.*, "Energy-adaptive scheduling of imprecise computation tasks for QoS optimization in real-time MPSoC systems," in *DATE*, 2017.

Sangeet Saha is currently associated with Department of Computer Science, University of Huddersfield, as a Lecturer and also a visiting fellow University of Essex (UK). His current research interests include real-time scheduling, scheduling for reconfigurable computers, real-time and fault-tolerant embedded systems, and cloud computing. He published many of his research contributions in conferences like CODES+ISSS, ISCAS, NASA AHS, etc. and in journals like ACM TECS, IEEE TCAD, etc.

Shounak Chakraborty is associated with Department of Computer Science, NTNU, Norway as a Post-Doc researcher. His research interests include High Performance Computer Architectures, Emerging Memory Technologies, Thermal Aware Architectures, etc. He published several of his research contributions in conferences like DATE, ASAP, CODES+ISSS, GLS-VLSI etc. and in journals like ACM TACO, ACM TECS, IEEE TCAD, etc.

Sukarn Agarwal is a Research Associate at University of Edinburgh (UK). His research interests include emerging memory technologies, memory system design and network-on-chip design. He published many of his research contributions in conferences like ASAP, VLSI-SoC, GLS-VLSI, ISVLSI, etc. and also published several of his research outcomes in journals like IEEE TVLSI, ACM TECS, IEEE TC, etc.

Rahul Gangopadhyay is associated as a researcher at St. Petersburg State University, Russia. His broad research domain is in Graph Theory, and specifically his research interests include hypergraph, rectilinear crossing, etc. He has published many of his research outcomes in journals like Computational Geometry, Graphs and Combinatorics, etc.

Magnus Själander is working as a Professor at the Norwegian University of Science and Technology (NTNU) and a Visiting Senior Lecturer at Uppsala University. Before joining NTNU in 2016 he has been a researcher at Chalmers, Florida State University, and Uppsala University. Själander's research interests include hardware/software co-design (compiler, architecture, and hardware implementation) for high-efficiency computing.

Klaus McDonald-Maier is currently the Head of the EIS Laboratory, University of Essex, UK. He is also the founder of UltraSoC Technologies Ltd., the CEO of Metrarc Ltd., and a Visiting Professor with the University of Kent. His current research interests include embedded systems and SoC design, security, development support and technology, parallel and energy-efficient architectures, computer vision, data analytics, and the application of soft computing and image processing techniques for real-world problems. He is a member of VDE and a Fellow of the BCS and IET.