ACCURATE: Accuracy Maximization for Real-Time Multi-core systems with Energy Efficient Way-sharing Caches

Sangeet Saha§, Shounak Chakraborty§, Xiaojun Zhai, Shoaih Ehsan, and Klaus McDonald-Maier

Abstract—Improving result-accuracy in approximate computing (AC) based real-time applications without violating deadline has recently become an active research domain. Execution-time of AC real-time tasks can individually be separated into: execution of the mandatory part to obtain a result of acceptable quality, followed by a partial/complete execution of the optional part to improve result-accuracy of the initial result within a given deadline. However, obtaining higher result-accuracy at the cost of enhanced execution time may lead to deadline violation, along with higher energy usage. We present ACCURATE, a novel hybrid offline-online approximate real-time scheduling approach that first schedules AC-based tasks on multi-core with an objective to maximize result-accuracy and determines operational processing speeds for each task constrained by system-wide power limit, deadline, and task-dependency. At runtime, by employing a way-sharing technique (WH LLC) at the last level cache, ACCURATE improves performance, which is further leveraged, to enhance result-accuracy by executing more from the optional part, and to improve energy efficiency of the cache by turning off a controlled number of cache-ways. ACCURATE also exploits the slacks either to improve result-accuracy of the tasks, or to enhance energy efficiency of the underlying system, or both. ACCURATE achieves 85% QoS with 36% average reduction in cache leakage consumption with a 24% average gain in energy delay product for a 4-core based chip-multiprocessor with 6.4% average improvement in performance.

Index Terms—Real-time scheduling, Approximated Computing, Multi-cores, Energy Efficiency, Dynamic Cache Way-Sharing, Dynamic Associativity Management

I. INTRODUCTION

In real-time computing, the correctness not only depends on the precision of the results, but also on the time at which they are produced. For such critical systems, approximated results obtained on-time are preferable over accurate results generated after the deadline has passed. For example, in a real-time video application, initially an inaccurate, but acceptable quality image is generated from the received data. Then, based on the available resources, the obtained image may further be refined [1]. Thus, Approximate Computation (AC) approaches [2] can minimize the possibility of tasks missing their deadlines due to strict resource requirements. In AC approaches, a task is decomposed into a mandatory part, followed by an optional part [3]. The mandatory part must be executed entirely in order to produce an acceptable result, while the result-accuracy increases with the execution cycles spent on the optional part. Specifically, to obtain a substantial amount of increase in result-accuracy, a certain number of additional cycles need to be executed from the optional part.

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.

Energy efficient scheduling of the approximated real-time tasks that target to maximize result-accuracy without violating the underlying system constraints have become a research topic of paramount importance in recent past. Stavrinides and Karatza were among the first to propose real-time scheduling of an AC based task-set [4]. In recent theoretical analysis [3], authors improved system level result-accuracy through task to processor allocation, and task adjustment constrained by a preset energy budget. But, restricting the energy usage does not guarantee thermal safety of the chip, which can be addressed by incorporating power constraint together with a runtime power management technique by considering several architectural parameters. However, comprehensive studies that combine the theoretical aspects of energy-efficient processing of approximated applications in real-time paradigm along with due consideration to the runtime architectural characteristics (e.g. cache performance, IPC, etc.) have not conducted so far.

A homogeneous chip-multiprocessor (CMP) platform along with a set of AC real-time tasks can be represented by precedence-constrained task graphs (PTG), equipped with multiple distinct implementable versions having various result-accuracy levels based on the respective amount of the optional part that is executed. By exploiting start time and the versions of the individual task nodes, our work, ACCURATE presented here, first determines task-to-processor allocation with an appropriate version of the individual task, the operating voltage/frequency (V/F) level, as well as their order of execution, such that the system level result-accuracy (i.e. QoS) is maximized, while meeting both the deadline, precedence, and power constraints. After the offline phase, task-executions are triggered as per the pre-computed schedule and each task will be executed with its associated V/F level assigned. During the execution, the cache based dynamic accuracy enhancement and energy minimization techniques of ACCURATE first attempt to improve the performance by adopting a way sharing
mechanism at the last level cache (LLC). This LLC-based runtime strategy ensures that improving performance through way-shared LLC (WH_LLC) can potentially finish the task early, which will be traded against either (i) to enhance result-accuracy by executing higher version of the task selected on-the-fly, or (ii) to improve energy efficiency by dynamically resizing the LLC.

As contemporary applications [5], [6], [7], that include approximations spend a significant amount of time accessing memory, employing way-shared LLC can reduce the total execution-time of the tasks and can generate slacks. ACCURATE attempts to exploit such slacks to enhance the result-accuracy by executing a higher optional version of the task (subject to availability), or by dynamically resizing the LLC to enhance energy efficiency while maintaining performance. Additionally, ACCURATE exploits slacks to enhance energy-efficiency of the system by enabling sleep/power-gated mode at the cores and LLC. However, our performance-cognizant online approach enhances result-accuracy for the tasks, and improves energy efficiency, without effecting the predetermined schedule. Figure 1 depicts the working mechanism of ACCURATE.

The major contributions of the ACCURATE are thus summarized as follows:

1) We propose an ILP based scheduling scheme, ACCURATE:Offline, for the AC real-time PTGs on a power-constrained CMP with an objective to maximize the result-accuracy, where the tasks are executed with a selected version (see Sec. IV-A).
2) We further propose a dynamic accuracy enhancement along with an online energy minimization techniques, i.e. ACCURATE:Online (see Sec. IV-B), which improves performance of the individual tasks, where improved performance is traded off either (i) to enhance result-accuracy by executing higher task-version selected on-the-fly, or (ii) to improve energy efficiency by dynamic LLC resizing. Additionally, in presence of any sufficiently large slacks, the system will be put into sleep/power-gated mode for more energy saving.

We argue and empirically validate the significance of our task scheduling approach in combination with our online cache based strategy (see Sec. V). The benchmark application based evaluation with a 4-core based baseline CMP (equipped with 2MB 8-way associative shared L2 cache) in our simulation setup (consisted of gem5 [8] and McPAT [9]) exhibits that through ILP-based task scheduling ACCURATE achieves 85% QoS, and the cache based online strategy reduces LLC leakage consumption by 36% on an average with 24% average gain in energy delay product (EDP) combined with 6.4% average performance improvement. The scheduling strategy of ACCURATE outperforms a prior Task Deploy [3] scheduling mechanism that offers a QoS of 55% for our considered task-set with 70% system workload, while ACCURATE achieves a QoS of 70%. We further empirically justify the exploitation of way-shared LLC (having a performance improvement of 10%) over another prior technique, Zcache [10] (having an average performance improvement of less than 6%) in ACCURATE. To

the best of our knowledge, ACCURATE is the first scheduling mechanism that trades off the performance gained by employing a way-sharing technique at LLC to improve both runtime energy efficiency and result-accuracy of the AC real-time task-set. After discussing the relevant related work in Sec. II we show how ACCURATE is different from the state-of-the-art.

Article Organization. After presenting the relevant related work in Sec. II, we will model the system in Sec. III where our processor and task models will be discussed along with the scheduling criteria. After modelling the system, the detailed mechanisms of ACCURATE will be illustrated in Sec. IV, in which Sec. IV-A and IV-B discuss ILP-based scheduling mechanism, and dynamic LLC based performance improvement and energy-efficient techniques, respectively. The efficacy of the ACCURATE is demonstrated in Sec. V along with detailing the description of our simulation setup. The paper is concluded in Sec. VI. The acronyms used in our paper are abbreviated in Table 1.

![Fig. 1: Overview of ACCURATE.](image)

<table>
<thead>
<tr>
<th>TABLE I: Acronyms and their Abbreviations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acronyms</td>
</tr>
<tr>
<td>AC</td>
</tr>
<tr>
<td>IC</td>
</tr>
<tr>
<td>QoS</td>
</tr>
<tr>
<td>CMP</td>
</tr>
<tr>
<td>NAQ</td>
</tr>
<tr>
<td>LLC</td>
</tr>
<tr>
<td>VIF</td>
</tr>
</tbody>
</table>

II. RELATED WORK

Now-a-days, energy minimization in contemporary multiprocessor embedded systems has become a topic of paramount importance [11], [12]. Energy efficient scheduling for the time-critical tasks, with precedence constraints on multiprocessor platform, imposes significant research challenges [13], [14]. Over the last few years, several research attempts [15], [16], [17], [18] were undertaken to devise energy and fault-aware real-time scheduling for a set of time-critical task-sets. Recently, in [19], Cao et al. introduce the concept of AC to meet the energy budget of a large scale real-time system that executes tasks without precedence constraints. Other prior efforts also explored energy-efficient AC tasks scheduling [19], [20], [21], without considering the precedence.
relations among the tasks. Yu et al. coined the concept of an “Imprecise Computation (IC)" tasks [22], where tasks also have a mandatory and an optional portions. Authors further proposed a “dynamic-slack-reclamation" technique to improve the system QoS to incorporate more energy efficiency, but task-dependencies were not considered. To the best of our knowledge, the first attempt to schedule IC/AC dependent tasks can be found in [4], where the authors compared the performance of conventional real-time scheduling approaches like Highest Level First (HLF) and Least Space Time First (LSTF) between two task-sets, where one set contains the AC tasks. However, this work did not include the energy efficiency.

The energy aware scheduling of dependent AC tasks are considered in [23], [3] that employ DVFS at the cores to improve energy efficiency. However, as DVFS curtails the supply voltage and frequency to save power, the transient faults of the system can significantly raise up the reliability issues [24]. Hence, in ACCURATE, we first propose an offline task allocation technique that schedules AC real-time tasks with respective frequency levels by considering precedence-power-temporal constraints. In addition, during execution, a way-sharing LLC strategy is employed to enhance the performance which will be further traded off towards stimulating result-accuracy as well as improving energy efficiency by dynamic cache resizing.

Authors in [25], [26] surveyed a number of performance cognizant low power on-chip cache design techniques along with their pros and cons. By employing Gated-VDD [24] at the circuit level to power gate the cache lines, a prediction based energy-efficient cache was proposed in [28] for TCMP static non-uniform cache access (SNUCA) based architecture, that incurs a remapping technique for the gated cache lines. To reduce cache leakage power significantly, a bank shutdown policy based on run-time bank usages was proposed in [29]. In [30], [31], the authors kept selected cache lines into low power drowsy/sleep mode, for minimizing cache leakage power where sleep mode consumes less power but retains stored data. In addition with effective reduction in overall energy consumption of a CMP, dynamic cache resizing can also assist in reducing chip temperature significantly [32], [33].

Towards uniformly distributing the cache loads across the cache sets, dynamic associativity management (DAM) techniques have been developed where heavily used sets are benefited by utilizing the idle ways of the underused ones. Several DAM based approaches [34], [10], [35] have already been proposed with variable implementation overheads. Out of these, FS-DAM [35] has been adopted in our work for its lesser implementation complexities along with the privilege of dynamic restructuring of the groups.

**ACCURATE over State-of-the-Art.** The majority of the prior scheduling approaches attempted to minimize the makespan time, however, in case of AC based precedence-constrained tasks, the objective becomes to maximize the overall result-accuracy, rather than makespan minimization. Moreover, most of these prior energy efficient scheduling mechanism employed DVFS at the cores, but have not considered on-chip LLCs that significantly contributes to the total on-chip power consumption [25]. As the majority of LLC power comes from their leakage consumption and a large portion of these LLCs remain underutilized during execution, prudential LLC resizing can be a viable knob to achieve energy efficiency [33], [32]. To exercise such energy-efficient mechanisms in real-time systems, promising techniques like DAM can be employed at the LLCs to safeguard the performance. In ACCURATE, after generating the schedule of the tasks through an ILP-based strategy (see Sec. IV-A), we have studied the potential of a DAM based way-sharing technique at the LLC in performance improvement for AC real-time task-set. During execution, ACCURATE further trades off this gained performance (see Sec. IV-B), either

- to save runtime energy by selective shutdown of LLC ways, where ways will be turned on if performance degrades, or
- to improve result-accuracy by executing higher version of the optional parts of the tasks, subject to availability.

**ACCURATE** also exploits the sufficiently large slacks to save more energy by enabling power-gated/sleep mode at the cores and LLCs. Our results also show, ACCURATE surpasses state-of-the-art techniques. To the best of our knowledge, ACCURATE is the first technique that considers LLC based online mechanism to enhance both result-accuracy and energy efficiency without violating the deadline constraint.

### III. System Model and Assumptions

We consider a CMP consisting of m homogeneous cores, denoted as $P = \{P_1, P_2, ..., P_m\}$. Each core supports 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 $F_i < F_{i+1}$. A real-time AC application ($A$) is modelled, as a PTG, $G = (T, E)$, where

- $T$ is a set of tasks ($T = \{T_i \mid 1 \leq i \leq n\}$) and
- $E$ is a set of directed edges ($E = \{(T_i, T_j) \mid 1 \leq i, j \leq n; \ i \neq j\}$),

representing the precedence relations between a distinct pair of tasks. An edge $(T_i, T_j)$ refers to the fact that a task $T_j$ may begin its execution after the completion of $T_i$. The source and sink tasks have no predecessors and no successors, respectively. Being a real-time application, $A$ must be executed within the given deadline, $D_{PTG}$, by executing all of its associated task nodes within the interval.

The worst-case execution length, $len_i$, for each task $T_i$ ($1 \leq i \leq n$) is logarithmically decomposed into $M_i$ cycles for the mandatory part, and $O_i$, the maximum cycles for the optional part. We further assume that a task $T_i$ may have $k_i$ different versions, that is, $T_i = \{T_i^1, T_i^2, ..., T_i^{k_i}\}$, which are distinct by their given execution lengths of their respective optional parts ($O_i$), denoted as $O_i^1, O_i^2, ..., O_i^{k_i}$, where $O_i^j$ achieves a higher result-accuracy than $O_i^q$, if $p > q$. The length ($\text{len}_i^j$) 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:

$$
\text{len}_i^j = M_i + O_i^j
$$

Note that, length of $T_i^j$ (i.e., $\text{len}_i^j$) includes the memory cycles needed to access LLC, which has been obtained by executing individual tasks for a particular configuration (see Figure 4). The result-accuracy $\text{Acc}_i^j$ of the $T_i^j$ is defined by the executed optional part of the task, $O_i^j$ (i.e., $\text{Acc}_i^j = O_i^j$).
Thus, the overall system level result-accuracy, which we also use to define the QoS of the system, is defined as the sum of the executed cycles of $O_i^j$ for all the tasks [19] and can be represented as:

$$QoS(A) = \sum_{i=1}^{n} O_i^j \mid T_i = T_i^j$$  \hspace{1cm} (2)$$

If a task $T_i$ executes at frequency $F_i$ then its execution time $ET_i$ can be denoted as $\lfloor len_i/F_i \rfloor$, which is a bound on the task execution time. We used this execution time for the offline phase. If $F_a > F_b$, then $\lfloor len_i/F_a \rfloor < \lfloor len_i/F_b \rfloor$. To enhance the result-accuracy of an individual task, while maintaining its deadline, a higher version of the task needs to be executed at a higher clock frequency of the core. However, increasing the clock frequency increases the power consumption ($Pow$), which might increase the core’s temperature. Hence, 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$^4$ Note that, $Pow_{BGT}$ includes power consumption of both cores and caches, where dynamic power consumption at the cores is higher than the static counterpart and caches are accounted for their static power consumption [25], [32]. However, towards maintaining accuracy in estimating the power consumption, both dynamic and static power have to be considered. Hence, our runtime power consumption is modeled by employing McPAT [9] tool, that estimates power consumption values (both dynamic and static power) for both cores and caches for our specific system configuration detailed in Sec. V-B1.

IV. ACCURATE

In this section, the working mechanism of ACCURATE is illustrated. After elaborating the ILP based scheduling in Sec. IV-A we will discuss the runtime LLC based power minimization and accuracy enhancement mechanism of ACCURATE in Sec. IV-B. Firstly, ACCURATE generates the schedule and provides the following information: (i) task to core mapping, (ii) start- and end-times of the individual tasks, (iii) assigned frequency, and (iv) respective tasks’ versions. A dispatch table stores the generated scheduling information by arranging the tasks as per their execution-order, which will be used to execute the tasks at runtime. During execution, ACCURATE traverses the dispatch table, selects and fetches individual tasks to execute according to their start time-stamps. Basically, while running the task-set, ACCURATE: Online allows the measurements of release and completion times for each task. These measures of time correspond to the generated schedule which is presented afterwards in Table IV and the respective pictorial timing diagram is shown in Figure 3. Note that, the dispatch table is stored and maintained in a repository residing in memory.

To empirically validate ACCURATE, at first we employ the tool CPLEX [36] to verify the constrained scheduling, with an example task-set represented as a DAG, where we created task with PARSEC application$^5$ (see Sec. V). After that, by accessing dispatch table, the generated information for this task-set will be used in our online simulation framework consisting of gem5 [8] (a full system simulator for performance traces) and McPAT [9] (power simulator). Our evaluation framework for the online mechanism considers a 4 out-of-order (OoO) core based tiled CMP architecture (TCMP) [37] (discussed further in Sec. V with the detailed simulation setup). To enable way-gating at the cores, ACCURATE incorporates power-gating mechanism [27] at the way-level granularity of each LLC bank, having negligible implementation overhead.

A. ACCURATE: Offline (ILP based Scheduling)

We present a scheduling strategy based on integer linear programming (ILP). For this purpose, we define a binary decision variable, $Z_{ikl\theta}$, where, $i = 1, 2, ..., n; k = 1, 2, ..., k_i; l = 1, 2, ..., L; \theta = 1, 2, ..., m$; Here indices, $i, k, l$, and $\theta$ denote task ID, corresponding version ID, particular V/F level, and the processor ID, respectively. $Z_{ikl\theta} = 1$, if the $k$-th version of $T_i$ (i.e., $T_i^k$) executes on processor $\theta$ at $l$-th V/F level, otherwise 0. We define another binary variable $Y_{ij}$, where $Y_{ij} = 1$, if task $T_i$ starts before $T_j$, else 0. Let $t_{start}(T_i)$ and $t_{finish}(T_i)$ denote the start time and finish time of the task $T_i$, respectively. Then we have

$$t_{finish}(T_i) = t_{start}(T_i) + \sum_{k=1}^{L} \sum_{l=1}^{m} \sum_{\theta=1}^{\theta_m} \left\lfloor \frac{len_k}{F_l} \right\rfloor Z_{ikl\theta} \hspace{1cm} (3)$$

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

1) Each task $T_i$ is assigned to exactly one processor with a particular version and executed at one frequency level-

$$\sum_{k=1}^{L} \sum_{l=1}^{m} \sum_{\theta=1}^{\theta_m} Z_{ikl\theta} = 1 \hspace{1cm} (4)$$

2) The application $A$ meets its end-to-end absolute deadline $D_{PTG}$. Hence, the sink node $T_n$ must be finished by $D_{PTG}$. This constraint can be represented as:

$$t_{finish}(T_n) \leq D_{PTG} \hspace{1cm} (5)$$

3) The peak power consumption of the system should not exceed the given power budget. Let $Pow_{peak}$ represents the peak power consumption of the system-

$$Pow_{peak} = \max\{Pow_{sys}\} \hspace{1cm} (6)$$

where,

$$Pow_{peak} \leq Pow_{BGT} \hspace{1cm} (7)$$

$Pow_{sys}$ is the power (both dynamic and static) consumption of all the busy cores, and can be obtained by summing up power consumption of all the tasks executing at that instant.

$^1$Our assumed fixed temperature is 350K, which is a reasonable average temperature of our considered processing platform while executing PARSEC benchmarks [33], [32].

$^2$We have also collected both CPU and memory cycles and power usages for each task by executing them on our simulation setup.
4) Execution dependency between tasks should be satisfied. The execution of $T_j$ must commence only after the completion of its predecessor $T_i$.

$$\forall (T_i, T_j) \in E \mid t_{start}(T_j) \geq t_{finish}(T_i)$$

(8)

5) To ensure, the tasks have no overlapping executions in the same processors, the following inequalities need to be satisfied: $\forall (T_i, T_j) \in A$, where $i \neq j$,

$$Y_{ij} + Y_{ji} > 0$$

(9)

$$Y_{ij} + Y_{ji} \leq 1$$

(10)

\[
t_{finish}(T_i) \leq t_{start}(T_j) + (1 - Y_{ij}) \times M
\]

(11)

Equation 11 prevents time-wise overlap of two tasks on the same processor, i.e. $T_j$ must start after completion of $T_i$, if $T_i$ starts before $T_j$. If tasks are executed in opposite order, we use big-M nullification to deactivate the constraint. $M$ has been considered as: $M = \max\{\{\frac{len_i}{PTG}\}_i, \forall l\}$.

Objective. The objective of the formulation is to choose the feasible solution, which maximizes QoS of the application. Hence, the objective can be written as follows:

Maximize $QoS(A)$

(12)

Here, in the context of this ILP formulation, $QoS(A)$ can be found as:

$$QoS(A) = \sum_{\theta = 1}^{n} \sum_{i=1}^{n} \sum_{k=1}^{L} Z_{\theta ik} \times O_i^k$$

(13)

subject to the constraints presented in Equation 4 to 11.

**TABLE II: Complexity of ILP**

<table>
<thead>
<tr>
<th>Equation</th>
<th># Constraints</th>
<th># Variables Per Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equation 4</td>
<td>$O(n)$</td>
<td>$O(K)$</td>
</tr>
<tr>
<td>Equation 5</td>
<td>$O(1)$</td>
<td>$O(K)$</td>
</tr>
<tr>
<td>Equation 6</td>
<td>$O(n)$</td>
<td>$O(K)$</td>
</tr>
<tr>
<td>Equation 7</td>
<td>$O(n)$</td>
<td>$O(K)$</td>
</tr>
<tr>
<td>Equation 8</td>
<td>$O(n^2)$</td>
<td>$O(K)$</td>
</tr>
</tbody>
</table>

Complexity Analysis: We present the complexity analysis for our ILP in Table II. The second column of this table lists the upper bound of the number of constraints for each equation. The unique resource constraint in Equation 7 should be determined for all $n$ tasks, hence, for a given PTG, overall $n$ constraints will be required. Similarly, the number of variables for this constraint can be represented as $O(K \cdot L \cdot m)$, where $K$ denotes the maximum number of possible constraints 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 $O(K)$. For deadline constraint in Equation 5, 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 $O(n^2)$. It may be noted that the complexity of ILP is independent of the number of processing elements in a platform and deadline of a PTG.

![Fig. 2: Task graph](image)

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

<table>
<thead>
<tr>
<th>Task</th>
<th>$M_i$ (#cycles)</th>
<th>$O_i$ (#cycles)</th>
<th>$Pow_i$</th>
<th>Task</th>
<th>$M_i$ (#cycles)</th>
<th>$O_i$ (#cycles)</th>
<th>$Pow_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>10</td>
<td>6</td>
<td>20</td>
<td>$T_4$</td>
<td>20</td>
<td>6</td>
<td>30</td>
</tr>
<tr>
<td>$T_2$</td>
<td>20</td>
<td>5</td>
<td>30</td>
<td>$T_5$</td>
<td>20</td>
<td>12</td>
<td>40</td>
</tr>
<tr>
<td>$T_3$</td>
<td>20</td>
<td>7</td>
<td>30</td>
<td>$T_6$</td>
<td>8</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>$T_4$</td>
<td>20</td>
<td>4</td>
<td>30</td>
<td>$T_7$</td>
<td>8</td>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>$T_5$</td>
<td>20</td>
<td>8</td>
<td>20</td>
<td>$T_8$</td>
<td>20</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>$T_6$</td>
<td>20</td>
<td>10</td>
<td>20</td>
<td>$T_9$</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
</tbody>
</table>

Example: Constrained scheduling at work: Let us consider the real-time task graph according to Table III and Figure 2. This PTG application needs to be scheduled on two processors ($m = 2$), with a deadline $D_{PTG} = 100$ time units. Our assumed power budget for both processors is set as $Pow_{BGT} = 50$. As per the constrained scheduling strategy, CPLEX [36], the ILP solver generates the scheduling output shown in Figure 3. The results are also represented in tabular form in Table IV. From Figure 3, it can be found that tasks $T_1$, $T_3$ and $T_5$ were executed with their highest versions on processor $P_1$. Out of these three tasks, $T_3$ executes in lower V/F level (i.e. 0.5) for satisfying the power constraint. On the other hand, task $T_2$ is able to execute with its highest version (of the available three versions) on the processor $P_2$ to maximize the overall QoS of the system. However, $T_4$ and $T_6$ executed on $P_2$ with their respective lowest versions, in order to maintain the temporal constraint. It is evident that the entire PTG is able to finish by 100 time units and thus, $D_{PTG} = 100$ has been fulfilled. The total obtained QoS value is 45.

**TABLE IV: Outputs of the constrained scheduling**

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Mapped Processor</th>
<th>Selected Version</th>
<th>Execute Start Time</th>
<th>$O_i$</th>
<th>Assigned V/F Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>$P_1$</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>$T_2$</td>
<td>$P_2$</td>
<td>3</td>
<td>16</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>$T_3$</td>
<td>$P_1$</td>
<td>3</td>
<td>16</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>$T_4$</td>
<td>$P_2$</td>
<td>1</td>
<td>46</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>$T_5$</td>
<td>$P_1$</td>
<td>3</td>
<td>46</td>
<td>5</td>
<td>0.5</td>
</tr>
<tr>
<td>$T_6$</td>
<td>$P_2$</td>
<td>1</td>
<td>72</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

Achieved QoS = 45

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
B. ACCURATE:Online (Dynamic Accuracy Enhancement and Power Minimization)

Once the tasks are scheduled, the execution will be triggered and our runtime mechanism will first boost up the performance by incorporating a way-sharing based technique (WH LLC) at the LLC (detailed in Sec. IV-B1). By logically increasing the cache associativity on-the-fly, WH LLC reduces the number of cache misses, that limits the number of off-chip (memory) accesses. Thus, the running time of the task is reduced, and it generates a set of idle processor-cycles (which will be called private slack for individual tasks in this paper from here onward) at the end of the execution of each individual tasks in the predetermined schedule. Next, our online technique will utilize the private slack for each task in a couple of ways (see Sec. IV-B2). The tasks, those have been scheduled with their highest version, will exploit the private slack only for improving energy efficiency by turning off a set of LLC ways on-the-fly for reducing LLC leakage power consumption. This dynamically trimmed LLC might affect the performance by increasing the number of cache misses. However, our online mechanism periodically monitors the performance and turns on cache ways, if needed, to maintain the predetermined schedule. On the other hand, the tasks scheduled with a result-accuracy, having room for further improvement, might exploit the private slack by running the highest possible versions from their optional parts to enhance the result-accuracy. Note that, in both the cases, the predetermined schedule will not be violated. However, our online mechanism can be tuned further, to balance the power-performance trade off as per the system requirements.

Before applying WH LLC, we first analyzed nine PARSEC applications by running them in gem5 for a stipulated number of clock cycles with our simulation setup (see Sec. V-B). Most of the prior analyses of the PARSEC regarding cache access patterns have shown the sufficiency of using 70 – 100M clock cycles, as by considering this analysis overall trend of cache access patterns can be realized for most of the PARSEC applications. In ACCURATE, we have used 80M clock cycles (in RoI) for all of our simulations related to background analyses.

Our simulation shows, a significant amount of their execution-times, these PARSEC applications spend in accessing memory, which is shown in Figure 4. In case of memory intensive applications, like Can, Ded, Fluid, and Stream, more than 50% of the execution-times are spent on accessing memory. The adopted LLC-based way-sharing technique, WH LLC, and a prior way-sharing policy Zcache that significantly curtail the memory accesses by reducing capacity and conflict misses through better utilization of the LLC space and thus, improve performance. We further implemented and compared WH LLC and Zcache with our simulation setup (mentioned above), and showed the performance improvements for the individual benchmarks in Figure 5. As per this figure, WH LLC outperforms Zcache for all of these nine applications with 10.5% improvement in IPC (on average), whereas Zcache achieves 5.6% average IPC improvement, which motivated us to adopt WH LLC in the time-critical environment of ACCURATE.

1) Improving Performance at the LLC: Prior empirical analyses showed that, due to locality of reference, the LLC accesses of applications are distributed non-uniformly across different granularity levels (bank, set, way, etc.) of the LLC, that keeps a big chunk of the LLC portion underutilized. Several dynamic associativity management (DAM) based techniques have been evolved to logically handle such load distributions by providing heavily used cache sets the privilege of using the idle ways of the underutilized ones. Figure 6 illustrates the entire WH LLC mechanism for an 8-way set associative (A) cache having 8 cache sets (S). First, a number of cache sets are grouped together to form a Fellow-group based on their usages, such that each group contains a mix of lightly and heavily used cache sets. Next, each of these cache sets is divided into two logical regions: normal ways (NT) and reserved ways (RT), where any cache set within a fellow group can use RT portions of all member cache sets.

In Figure 6, cache sets 0, 1, 3 and 5 are in a same fellow group, and can share their RT ways, and similarly, cache sets 2, 4, 6 and 7 will also share their RT ways, respectively. Logically, the associativity of each cache set is now increased to 20 (from originally 8), which drastically reduces the capacity and conflict misses at the heavily used cache sets and improves the overall system performance.

Note that, WH LLC handles the existing diversities in cache set usages during different execution phases of the task, by dynamically restructuring these Fellow-groups. The functional
correctness of the addressing mechanism in addition with the detailed discussion on this way-sharing mechanism is out of the scope of this paper.

Fig. 6: An example of WH_LLC.

Figure 7 illustrates how WH_LLC will improve the performance in ACCURATE. The darker task-blocks for individual tasks imply the modified execution spans of the respective ones with WH_LLC in action, while the corresponding brighter portions with dotted borderlines are representing the older schedule (see Figure 3). We have also shown the generated private slack only for T3. Practically, the improved memory latency by employing WH_LLC will boost up the overall performance, which is reflected through the reduced execution times for the individual tasks. The change in execution time (Exec. Time) for T3 after applying WH_LLC is explicitly shown in the figure. Note that, the performance improvements for the tasks in Figure 7 are not to scale/measure. Our simulation results in Sec. V will show the changes in performance for the individual tasks consisted of PARSEC benchmarks (see Table VI).

Algorithm 1: Per-core power reduction and result-accuracy enhancement within a FRAME

![Algorithm 1](image)

2) Enhancing Power Efficiency and Result-Accuracy: Incorporating WH_LLC logically divides each LLC set into two parts, as discussed earlier. Hence, shutting down a physical cache way will have different impacts on the task’s performance, depending upon if it is an NT or an RT way. Figure 8 shows how way shutdown will change the associativity for an 8-way LLC, having a fellow-group size of 4 with 4 dedicated ways per set for RT. Shutting down 2 physical cache ways from NT portion will reduce the logical associativity to 18. On the other hand, if 2 physical ways can be turned off from the RT part, logical associativity will be reduced by 2 × 4, i.e. 8, so finally it will be 12. And shutting down 2 ways individually from NT as well as from RT will entail the logical associativity to 10, which is still higher than the original one (8). So, by employing WH_LLC, even after shutting down 50% (physical) ways from a cache bank, we can still maintain an associativity of 10. This can however partially curtail the gained benefits of WH_LLC, but will still be able to maintain the performance over the baseline while significantly reducing the power consumption. Note that, in this work, we set the upper limit for way shutdown at 50% from each of the NT and RT ways. For all tasks, that have been scheduled with their highest version, the way-shutdown will be applied for reducing LLC power consumption. To avoid any implementation conflicts, ACCURATE does not allow concurrent execution of dynamic LLC resizing and reconstruction of the Fellow-group in WH_LLC.

Algorithm 1 to 6 present the complete procedure for performing way-shutdown at the individual LLC banks along with the result-accuracy enhancement. Once the schedule is

3By considering our system configuration (see Sec. V-B), we restricted ourselves to ensure the available cache size at least 50% during execution based on prior cache requirement analyses of PARSEC [5]. Note that, the value of this limit is application dependent.
generated, the individual tasks’ start- as well as end-times are
determined. ACCURATE: Online next converts all such timing
parameters to cycles and stored in dispatch table, whereas the
duration (in cycles) of the deadline is named as FRAME.

Algorithm 1 takes the following parameters as the inputs: Interval_length, Sleep_Thr, Turn_ON_OH, and
#available_higher_versions_of_Oi. During execution, Al-
gorithm 1 checks the LLC usages periodically at the end of each
Interval_length number of cycles, which is set by
considering prior analyses of LLC usages [33, 32, 28].
Sleep_Thr is a minimum threshold value for a slack-span
which is also known as the processor’s break-even time [39],
and whose value is architecture dependent. Turn_ON_OH
represents the time taken for the core to be turned on from
its sleep mode. The number of available higher versions
of Oi, task Ti over its scheduled one is represented by
#available_higher_versions_of_Oi.

cycle_cnt, a variable, keeps track of the number of
cycles within a FRAME. #Off Ways_at_NT[B] and
#Off Ways_at_RT[B] counters keep track of the number of
turn off NT and RT ways, respectively, at a particular
LLC bank B. We also use a flag No LLC_resize_flag[Ti] to
decide (initialized to 0 at line 3), if LLC resizing for Ti will
be enabled. The end time-stamp for the individual tasks (within
a FRAME on the assigned core) is modified and called as
extended end time (Extended_End_Time_i), which is
defined as follows:

• Extended_End_Time_i is the scheduled start time of
  the next task (say Tj) assigned on the same core, if the
  current task is not the last task on its assigned core within
  the same FRAME.

• Extended_End_Time_i is set to the length of the
  FRAME for the last task of a particular core within the
  FRAME.

For example, Extended_End_Time_T2 at core P2 in
Figure 3 is 46, which is the start time of T4. The
Extended_End_Time_T5 will be 100, as T5 is the last task
of the FRAME at P1. For ease of understanding, all of these
time values can be assumed as cycles, e.g. 100 time units can
be considered as 100 cycles.

With the onset of the FRAME, the algorithm first checks
if any initial slack exists at the current core by looking at
the dispatch table. Such slack can only exist, if the tasks are
waiting at the current core for the execution of the source task
at some other core. For a sufficiently large init_slack having
a length of at least Sleep_Thr + Turn_ON_OH, sleep mode
will be enabled at the current core for the duration of the slack
(line 7 to 10). For enabling sleep mode at the core,
Sleep-Manager subroutine, i.e. Algorithm 2 is called, that
maintains a counter (gated_cycles) during sleep, and turns
the core on if the counter is exhausted (line 1 to 6).

For each ready task (Ti), Algorithm 1 first checks if the
task is scheduled with its highest version, and the execution
will be started (line 11 to 14). If a task is not scheduled with
its highest version, the system checks for the best possible
schedulable higher version available for the task by executing
Enhance-Accuracy process given in Algorithm 3 (see line 1
to 5). Before inspecting the availability of the higher Oi, the
algorithm will start executing Mi (line 18), and on completion
the time left for executing Oi, i.e. Cycles_Left_iOi, will be
determined (line 19). Based upon the available higher versions
which can be fitted within the time left, Oi will be updated
with the best possible one by calling Algorithm 3 and will be
executed accordingly (line 20 to 22). In our example, we were
able to dynamically schedule and execute the higher version

Algorithm 2: Sleep-Manager
Input: gated_cycles
1 update_cycle = gated_cycles + Turn_ON_OH;
2 Apply power gating at the core;
3 while gated_cycles > 0 do
4   gated_cycles--;
5 Turn on the core
6 return update_cycle;

Algorithm 3: Enhance-Accuracy
Input: #available_higher_versions_of_Oi
1 if #available_higher_versions_of_Oi ≥ 1 then
2 while Cycles_Left_iOi > Exec_Depth_iOi do
3   if Cycles_Left_iOi < Exec_Depth_next_iOi || Curr_Oi
4     = Highest_Oi then
5     # Update and return Oi;
6     # Go to next version of Oi;

Algorithm 4: Task-Execution
Input: Ti
1 if Ti is fetched then
2 Set the predetermined V/F level and start execution
3 while Task is being executed do
4   if cycle_cnt == Interval and No LLC_resize_flag[Ti] ≠ 0 then
5     Interval = cycle_cnt + Interval_length;
6     For each bank (B) do in parallel (Line 8 to 9):
7     # Call Algorithm 5 with #Off Ways_at_NT[B] and
8     #Off Ways_at_RT[B] as inputs, and update the cycles after
9     LLC-resizing;
10   cycle_cnt += Algorithm 5(#Off Ways_at_NT[B],
11   #Off Ways_at_RT[B]);
12 # Execute as normal;
13 # update the counter at the end of each clock cycle;
14 cycle_cnt +=

Algorithm 5: Dynamic LLC Resizing
Input: POWER_DOWN, POWER_UP, Limst
1 resize_cycles =0, total_cycles =0;
2 ratio = #misses(B)/# accesses(B);
3 if (ratio < POWER_DOWN) then
4   if (#Off Ways_at_NT[B] < Limit) then
5     # Select a victim way from NT;
6     total_cycles = Algorithm 6(resize_cycles, Way_i);
7     #Off Ways_at_NT[B];
8     else if (#Off Ways_at_RT[B] < Limit) then
9     # Select a (physical) victim way from RT;
10     total_cycles = Algorithm 6(resize_cycles, Way_i);
11     #Off Ways_at_RT[B];
12     else if (ratio > POWER_UP) then
13     if (#Off Ways_at_RT[B] > 0) then
14     Turn a (physical) way on from RT;
15     #Off Ways_at_RT[B] --;
16     else if (#Off Ways_at_NT[B] > 0) then
17     Turn a way on from NT[B];
18     #Off Ways_at_NT[B] --;
19 return total_cycles;
for $T_6$ (see Figure 9) by prudentially exploiting its private slack (included in $\text{Cycles}_\text{Left}(O_i)$). Note that, our algorithm does not allow dynamic LLC-resizing if a task’s version can be updated online, which, if allowed, might lead to deadline violation. Hence, the flag $\text{No LLC_resize_flag}[T_i]$ is set to 1 for the tasks whose version can be updated dynamically (see line 16). Our algorithm also looks for the availability of the sufficiently large slack-span after execution of each task, and on availability of such slacks, sleep mode will be enabled at the processor core by calling $\text{Algorithm 2}$ (line 23 to 26).

To execute tasks, $\text{Algorithm 1}$ calls $\text{Task-Execution}$ method given in $\text{Algorithm 4}$ that executes each task in the following manner. Once a task is fetched, the predetermined V/F level for this task will be set at the assigned processor core and the execution will be started (see line 2). During execution of a task, $\text{cycle\_cnt}$ is updated at each clock cycle, and this value is used to determine if an Interval is encountered and current task is eligible for LLC resizing (i.e. $\text{No LLC_resize_flag}[T_i] \neq 1$) (see line 4). Once the $\text{cycle\_cnt}$ is at the Interval, and the task is eligible for LLC resizing, the algorithm will attempt to resize the LLC by calling $\text{Algorithm 5}$ $\text{ACCURATE}$ is implemented with a multi-banked LLC, in which we will enable our way-level dynamic LLC resizing strategy at each bank $B$. Hence, $\text{Algorithm 5}$ will be called for all of these LLC banks (line 6 to 8). However, once resizing is done, the execution will proceed normally.

Existing diversities in cache access pattern across different execution phases of individual applications exiguate diverse cache requirements on-the-fly. As the time-criticality is enforced, keeping track of the task’s cache requirements during different execution phases is inevitable, which can be monitored by considering the miss rate at the bank level granularity. Therefore, at first, a ratio is calculated by $\#\text{misses}(B)/\#\text{accesses}(B)$ for the individual banks ($B$) on completion of an interval (Interval) (see line 2). If this ratio is smaller than $\text{POWER\_DOWN}$ (line 3), the algorithm will first check if the number of turned off ways ($\#\text{Off\_ways\_at\_NT}(B)$) is less than the maximum allowed ($\#\text{Limit}$) and then an NT way is selected as victim, and it will be shutdown eventually after invalidation or eviction of its blocks (line 4 to 7). If the number of turned off ways ($\#\text{Off\_ways\_at\_NT}(B)$) reaches the maximum allowed ($\#\text{Limit}$), then if the number of turned off ways in the RT portion ($\#\text{Off\_ways\_at\_RT}(B)$) is less than the maximum allowed ($\#\text{Limit}$) (line 9), a way from RT will be turned off after invalidation or eviction of its blocks (line 10 to 12). Note that, during eviction of the blocks from the victim way, the bank can still serve external memory accesses. The main difference is that, an eviction caused by a cache miss will not evict the data from the victim way.

On the other hand, if the ratio is larger than $\text{POWER\_UP}$ (line 14) and there exists at least one power-gated way at the RT portion, then one RT way is turned on on line 15 to 17. If RT has no gated ways at present, our algorithm will attempt to turn on a powered off NT way (see line 17 to 20). Note that, incorporation of two separate limits for ratio, where $\text{POWER\_UP}$ is larger than $\text{POWER\_DOWN}$ reduces the chance for oscillating resizing where one (physical) way is repeatedly turned on and off during stable execution phases. Depending on the system parameters and the average expected workload of the system, a suitable Interval length and other thresholds ($\text{POWER\_UP}$ and $\text{POWER\_DOWN}$) can be determined (see Sec. V-B1). Hence, these may either be set at design time or may be made configurable. The number of sets that can be evicted per cycle during way shutdown is to be limited by the number of memory ports (per bank). Note that, the block invalidation or eviction at the LLC ways are performed by $\text{Evict-Way}$ method in $\text{Algorithm 6}$ (line 6 and 11). As long as the blocks are available at a particular way, this algorithm will either write the block back to main memory, if dirty, or invalidate the block. Once this operation is done, the way will be turned off (line 1 to 6).

3) ACCURATE: Online Computational Overheads:

**Theorem 1.** The amortized complexity of $\text{ACCURATE: Online}$ ($\text{Algorithm 1}$ to $\text{Algorithm 6}$) is $O(n \log n)/\text{FRAME}$ per time-slot.

**Proof.** $\text{Algorithm 1}$ is the heart of $\text{ACCURATE: Online}$ technique that executes at each core, which at first investigates the dispatch table to identify if there exists a slack at the beginning of the $\text{FRAME}$. Such slacks can be determined just by looking at the dispatch table, hence, it incurs a computational overhead of $O(1)$. A step-wise analysis of computational overhead of $\text{Algorithm 1}$ due to the called functions/algorithms is as follows:

1. On presence of a slack at the beginning of the $\text{FRAME}$ the core will be gated, only if the slack-span is sufficiently large, by calling $\text{Algorithm 2}$ that keeps track of the time during sleeping. As sleep duration typically takes a small value, $\text{Algorithm 2}$ will incur a computational overhead of $O(1)$.

2. The “for loop” from line 11 to 26 may be executed $O(n)$ times in worst-case, although the number of tasks assigned to a core usually takes a small value.

   - In worst-case, the loop will execute line 16 to 22. This loop calls $\text{Algorithm 3}$ and 4. The “while loop” in $\text{Algorithm 3}$ can have a worst-case complexity of...
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2022.3161407, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Mathematically, NAQ can be formulated as:

$$NAQ = \frac{\sum_{i=1}^{n} Acc_{\text{i}} \times T_i}{\sum_{i=1}^{n} Acc_{\text{i}}}$$

for the entire PTG and the maximum possible achievable QoS (see Equation 2) achieved

1. Algorithm 4, 5 and 6 are called during task execution. For all practical purposes, computational overheads of these algorithms may be considered to be constant, however, implementation overheads for Algorithms 5 and 6 are limited [27].

2. Hence, the worst-case computational complexity of Algorithm 1 is $O(n \cdot k)$.

3. The number of processor cores is constant. Hence, at any FRAME, the total overhead for generating the schedules over all processor cores for the duration of a FRAME is $O(n \cdot k)$ in the worst case.

4. As the FRAME length is in $O(D_{PTG})$, the amortized complexity of ACCURATE: Online is $O(n \cdot k) / O(D_{PTG})$.

V. RESULTS AND ANALYSIS

In this section, we will illustrate the efficacy of ACCURATE by evaluating ILP based task allocation and scheduling (see Sec. V-A) and runtime energy efficiency and performance improvement (see Sec. V-B). Based upon the tasks’ parameters (e.g. execution time-spans, interdependencies among the tasks) and the number of available processor-cores along with the V/F levels, the tasks are allocated by the ILP based scheduling. Once the task-allocation is over, with the onset of the execution, our online cache based policy trims the execution spans of the individual tasks by activating WH_LLIC. In case the current task is scheduled with its highest version, then LLC leakage consumption will be reduced through selective power gating of the cache ways. On the other hand, while the task is scheduled with compromised accuracy, by trimming the execution span with WH_LLIC, the highest possible version of the task is selected for execution. Towards standardizing our evaluations, we have considered task-execution parameters as per AC real-time task-model of [3] in case of our offline strategy, whereas our online architectural technique is evaluated by employing a mixture of compute and memory bound PARSEC benchmark applications [5]. Moreover, a prior art claimed the eligibility of PARSEC in real-time environment [40].

A. Evaluating ACCURATE: ILP based scheduling

Performance evaluation has been carried out through a comprehensive set of simulation based experiments, considering a homogeneous multiprocessor system that executes a set of real-time precedence constrained tasks. Normalized Achieved QoS (NAQ) is the principal metric based on which the evaluation has been performed. NAQ can be defined as the ratio between the actually achieved QoS (see Equation 2) for the entire PTG and the maximum possible achievable QoS by executing the highest version of each task node. Mathematically, NAQ can be formulated as:

$$NAQ = \frac{\sum_{i=1}^{n} Acc_{\text{i}} \times T_i}{\sum_{i=1}^{n} Acc_{\text{i}}}$$

It can be inferred that NAQ contributes to derive a measure of the efficacy of the offline phase. Specifically, it determines how much optional portion of each task has been executed, depending upon the chosen version, by satisfying the constraints.

Now, to show the efficacy of our offline technique, we model a multiprocessor system along with a task-set as follows:

- **Processor System:** For our experiment, we consider a multiprocessor platform equipped with 4 Alpha 21364 cores, where per core $Pow_{BGT}$ is set at 2.7W which is obtained through power-profiling for individual tasks in McPAT [9].

- **Task Characteristic:** Each PTG consists of a set of subtasks under dependency constraints with a deadline $D_{PTG}$. Each subtask ($T_i$) is a multithreaded task (see Table VI), where all threads of a single task are executed on the same core (in a quasi-parallel manner) which is characterized by the execution times, $ET_i$. We also assumed that a subtask can consume between $4 \times 10^7$ and $6 \times 10^8$ clock cycles [3]. Note that these WCET values of tasks have been assumed to be calculated by employing the framework as stated in [41]. This framework enables to quantify the possible overestimation of WCET upper bounds obtained by the static analysis. The prime objective was to derive a lower bound on the WCET to complement the upper bound. As ACCURATE employs hybrid offline-online approach, such static analysis will be beneficial for us to eliminate the overestimation, and we can expect much realistic WCET.

It is further assumed that each task node can have a maximum of 5 versions, i.e. $k = 5$. The assumptions regarding execution lengths also include memory cycles for our individual tasks, consisting of PARSEC benchmark applications [5], [35]. The total execution requirement of a PTG ($C_{PTG}$) is defined as the sum of the execution times of its subtasks, $C_{PTG} = \sum_{i=1}^{n} ET_i$. Hence, the utilization $U_i$ of a PTG can be denoted as $C_{PTG} / D_{PTG}$. The average utilization of a PTG is taken from normal distribution by considering normalized frequency 0.5. Given the PTG’s utilization, we can obtain the total system-utilization ($Sys_{util}$) by summing up the utilization of all the PTGs. Given the system utilization, the total system workload ($Sys_{work}$) or system pressure can be derived by: $Sys_{work} = \frac{Sys_{util}}{m} \times 100\%$. For a given system utilization, we have generated the PTGs by following the method proposed by Qamhieh and Midonnet [42]. Given a $Sys_{work}$, a set of DAGs is created. The number of DAGs ($\rho$) within a set can be measured as:

$$\rho = \frac{m \times Sys_{work}}{U_i}$$

In our generated PTGs, the minimum number of tasks (nodes) is equal to 5 and the maximum number of nodes is set to 20. For each of our PTGs in the set, the number of nodes have been randomly generated within the specified limit. It can also be noted that, as individual utilization ($U_i$) of a DAG is lower than the given system workload ($Sys_{work}$), the number of DAGs ($\rho$) within the set will always be higher than $m$. All of our experiments are carried out by using the CPLEX optimizer version 12.10.0, with a timeout of 5 hours.
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2022.3161407, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

1. Task Temporal Parameters: For each task $T_i$, based on which portion of the $l_{en_i}$ is considered as its mandatory portion ($M_i$), we considered the following cases: (i) $\text{man}_\text{low}: M_i \sim U(0.2, 0.4) \times l_{en_i}$ (low portion of a task $T_i$’s length ($l_{en_i}$) is for the mandatory portion). (ii) $\text{man}_\text{med}: M_i \sim U(0.4, 0.6) \times l_{en_i}$ (medium portion of a task $T_i$’s length ($l_{en_i}$) is for the mandatory portion). (iii) $\text{man}_\text{high}: M_i \sim U(0.6, 0.8) \times l_{en_i}$ (high portion of a task $T_i$’s length ($l_{en_i}$) is for the mandatory portion).

2. Frequency Level: We have chosen two distinct normalized frequency levels as: $f_{\text{norm}} = 0.5$ and 1 for task execution. The respective actual V/F settings for our considered cores are given in Table V.

Scalability analysis of ILP: Figure 10 depicts average solving time per number of tasks (nodes) in each PTG. We observed that, when the number of tasks in each PTG is within 10, the average solving time remains comparable. This implies, if the number of tasks lies within 10, the increase in solving time does not significantly vary with the number of tasks. However, when the number of tasks increases further, i.e. more than 10, the average solving time also increases. This observation is also supported by the complexity analysis provided in Table II. Empirically, we further noticed, with $n = 20$, the ILP generates on average 5000 constraints for which the solving time reaches approximately 140 minutes.

Fig. 10: Analysis of running time of ILP-formulation.

Fig. 11: Change in NAQ for various system workloads.

![Fig. 11: Change in NAQ for various system workloads.](image)

B. Evaluating ACCURATE: Online LLC-based Technique

The evaluation of the WH_LLCC-based dynamic accuracy-enhancement and power minimization is carried out by employing architectural simulators, where our entire online technique (discussed in Sec. IV-B) has been implemented. Before demonstration of our results, we will first discuss the simulation setup.

1) Simulation Setup: We simulated two 4 core based homogeneous TCMP with 4 replicated tiles (see Figure 13) in gem5 full system simulator [8] as our baseline system, where each of these TCMP is representing a single processing element (i.e. $P_i$ in Figure 13). However, each tile of these TCMP contains an In-Order (InO) Alpha 21364 core together with its private
L1 (Data and Instruction) caches. The whole L2 cache (LLC in our case) is physically distributed/sliced uniformly among the tiles, called L2-bank, but logically the L2-banks share a single address space. The tiles are connected through a 2D-mesh NoC, hence, each tile is also equipped with a router (depicted by the circles in Figure 13). We implemented Algorithm 1 to 6 in the Ruby module of gem5, and associated performance overheads for implementing these algorithms are also considered in our simulation. For estimating power/energy consumption (based on 32nm technology nodes), performance traces are fed to another simulator, McPAT [9]. The incurred energy overheads for implementing the online mechanism of ACCURATE are also derived from McPAT.

Table V contains the configuration parameters for the processor cores and memories used in the evaluations. We generated our tasks by using PARSEC benchmark suite [3] which can be fitted in an AC based paradigm [7, 43]. In their work, Sidiroglou et al. showed how PARSEC benchmark programs can be used in the approximation paradigm through the loop perforation technique [43]. To simulate our application (mentioned in Table III), we use 6 tasks where each processor (i.e. each 4-core based TCMP) executes the allocated tasks without any preemption. The tasks are framed by randomly combining executions of multiple PARSEC benchmark programs together, where each one might also appear multiple times (see Table VI). This implies, each of our tasks is multi-programmed, hence, our application (A) is a collection of multi-programmed tasks. Basically, in Table VI we show how each $T_i$ in Figure 2 (described in Sec. IV-A) is formed by PARSEC benchmark programs. Towards simulating the whole system with PARSEC, we further scale up the values of $M_i$, $O_i$ and $D_{PTG}$ by 100 million. Note that, the individual task cycles include both processor and memory cycles for the specific cache configuration given in Table V. Towards empirically validating and verifying ACCURATE with the contemporary workloads, we employ multithreaded PARSEC benchmark programs, where each individual program is executed with 4 threads. However, the discussion related to the detailed allocation of the benchmarks and their threads inside each task to the cores of the TCMP, which is internally managed by our simulation setup, is out of scope of this paper.

The Baseline values in all of our results that evaluate runtime techniques of ACCURATE are produced by executing the schedule generated by ILP based scheduling (discussed in Sec. IV-A) without incorporating any changes during execution. Also note that, as we mentioned earlier, all timing parameters derived from the scheduling strategy are converted to clock cycles while filling up the dispatch table with the task details. The task details regarding their execution length (for mandatory and optional parts) in cycles for a particular configuration of the processing platforms needs to be made available beforehand. Details of the processing platform includes the number of cores per processor (e.g. it is 4 in ACCURATE), available operational processing frequencies, cache configurations and memory sizes (see Table V). The processor and memory cycles for each task are also derived prior task-scheduling through pre-executions of the tasks. The percentage of execution time spent for memory accesses are shown in Figure 3 for individual PARSEC benchmark program.

**Fig. 14:** Range of cache miss ratio (ratio in Algorithm 5).

Table V: System parameters and their values

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology used</td>
<td>32nm</td>
</tr>
<tr>
<td>Max. V/F</td>
<td>1.02v/1800MHz</td>
</tr>
<tr>
<td>Min. V/F</td>
<td>0.7v/900MHz</td>
</tr>
<tr>
<td>ALU per core</td>
<td>1</td>
</tr>
<tr>
<td>Fetch Width</td>
<td>2</td>
</tr>
<tr>
<td>Issue width</td>
<td>4</td>
</tr>
<tr>
<td>#Float_Reg</td>
<td>32</td>
</tr>
<tr>
<td>Cache Model</td>
<td>SNUCA</td>
</tr>
<tr>
<td>L1 I/D Cache</td>
<td>64KB, 4-ways</td>
</tr>
<tr>
<td>L1 Latency</td>
<td>3 Cycle</td>
</tr>
<tr>
<td>Cache Block Size</td>
<td>64 Bytes</td>
</tr>
<tr>
<td>L2 Latency (512KB)</td>
<td>10 Cycles</td>
</tr>
<tr>
<td>Memory bank</td>
<td>1GB, 4KB/page</td>
</tr>
</tbody>
</table>

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
2) Change in Performance at the task level: After implementing WH LLC and dynamic way-shutdown techniques (Algorithm 1 and 5) in ruby module of gem5, we noticed the changes in IPC (Instructions Per Cycle) at the task levels during execution. Employing WH LLC significantly boosts LLC performance, by reducing capacity and conflict misses that further reduces off-chip accesses and resulting into improved IPC. But, incorporation of way-shutdown (proposed in Algorithm 5) further aggravates performance gained through WH LLC, however this performance degradation is compensated by a remarkable reduction in leakage consumption (discussed next). We further compared WH LLC with another DAM based prior work, Zcache [10], that yields increased LLC associativity rather than the actual number of ways by increasing the number of replacement candidates. Figure 15 shows the impacts on performance of WH LLC, ACCURATE (WH LLC + LLC resizing), and Zcache for the individual applications over the baseline. WH LLC is able to improve performance by 10% on an average for all tasks, with a minimum improvement of 9.5% in case of T1. However, this result shows ACCURATE curtails performance gained by WH LLC for individual applications, but is still able to maintain a better IPC over baseline, which ensures meeting of the real-time constraints.

Among all of our tasks (mentioned in Table VI), T2 and T5 are memory intensive, whereas the other tasks are comprised of mixed (memory plus computational) workloads. Hence, the performance degradation is comparatively higher in case of T2 and T5 in ACCURATE, than in the other tasks. However, our dynamic way turn on mechanism (in Algorithm 1) safeguards the executions from violation of deadlines by providing more cache space to the tasks, on demand. Note that, even after shutting down cache ways on-the-fly, our technique still shows better performance than the baseline, as well as Zcache. Our technique ACCURATE still maintains a mean performance improvement of 6.4% over baseline, which is 10.4% with only WH LLC (over the baseline), whereas Zcache boosts performance up by 5.7% over baseline. Moreover, this empirical result implies that, any task for which a higher version is available, with an additional execution span (in clock cycles) of within 10% of the currently scheduled version, can enhance the result-accuracy by executing its higher version. Additionally, energy efficiency can be enhanced by enabling the sleep mode, subject to availability of the private slack.

3) Reduction in LLC-Leakage: We set the upper limit for way-shutdown to 50% in Algorithm 5 [44] that reduces around 36% of leakage power on an average across the applications. Figure 16 exhibits the reduction in LLC-leakage consumption for the individual applications, where the leakage reduction is more in case of the mixed workload based tasks (T1, T3, T4 and T6). Requirement of higher run-time cache space curtails the leakage reduction for the memory intensive tasks (T2 and T5) for which Algorithm 1 was unable to maintain a lower cache size for a long time-span on-the-fly. Note that, we executed all of these tasks with their respective highest versions (i.e. the best possible ones) along with the assigned V/F level (at the core) (by scheduling mechanism in Sec. IV) towards illustration of the efficacy of our online mechanism.

4) EDP Gains: For the same set of applications executing with their respective highest version, our cache based online technique shows lesser EDP gains in the cases of memory intensive tasks (T2 and T5), due to their comparatively lesser reduction in LLC-leakage. On the other hand, mixed workloads (T1, T3, T4 and T6) are able to provide higher EDP gains due to higher reduction in the LLC-leakage consumption while applying ACCURATE. Figure 17 shows significant gains in EDP across the tasks while applying ACCURATE. Our online LLC based strategy is able to offer a significantly higher average EDP gain of 24% and this gain lies within the range of 19 – 28% for our task-set. Note that, EDP for each application includes the power consumed by both the processor-cores and the two levels of caches.

C. Gains from ACCURATE in a nutshell

The offline mechanism first generates the schedule and is able to achieve around 85% NAQ (see Sec. V-A), while maintaining the system constraints. Our online cache based strategy shows a significant performance improvement of 6.4% on average (see Sec. V-B2), while reducing 36% LLC leakage power consumption on an average (see Sec. V-B3) by shutting down a number of LLC-ways. The overall performance improvement of the online policy ensures to meet the timing constraints determined by the offline scheduling. However, while maintaining the deadline constraint, our cache based online technique is able to reduce a significant amount of energy by generating private slacks, which are employed for sleep that enables to noticeable overall energy reduction of 44% (see Figure 18).

By employing Algorithm 1 and 5, we have modified the schedule online, which is reported in Table VII. For tasks T1, T2, T3 and T5, Algorithm 1 applies WH LLC along with the way-shutdown, whereas for T4 and T6, Algorithm 1 attempted to improve the result-accuracy. The Scheduled Time-span column in Table VII shows the output of our offline technique, and the next two columns present the actual runtime with WH LLC and ACCURATE (that includes WH LLC and dynamic LLC resizing), respectively. In our schedule, for T4 and T6 we have scopes to improve the result-accuracy, as they are not scheduled with their respective highest versions. Our algorithm is able to improve the result-accuracy online for T6, which is highlighted in green background whereas red background in case of T4 implies it can not be executed with its higher version due to violation of the schedule. For
T6, the actual running time with WH_LLCE and ACCURATE are lower than its predetermined execution spans, and note that for T6, way-shutdown was not performed. The private slacks generated at the end of the execution of any tasks will be employed for sleep. Note that, during execution of source (T1) as well as sink (T6) tasks, only one core where the source/sink task is assigned will be active, and the rest will be kept in sleep mode. By executing higher version in case of T6, our technique is able to achieve a result-accuracy of 47, which was 45 at the end of our offline scheduling. Finally, our overall energy savings for individual task-level is shown in Figure 18. This figure shows, by incorporating way-shutdown and sleep, we achieve 44% savings in overall energy consumption for our task-set. So, the amalgamation of these techniques in ACCURATE (offline plus online) can offer an energy-efficient AC real-time task-allocation strategy with higher achievable QoS.

TABLE VII: Final Schedule with enhanced result-accuracy.

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Scheduled Time-span</th>
<th>Run-time only with WH_LLCE</th>
<th>Run-time with ACCURATE</th>
<th>Private Slack</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>16</td>
<td>14.5</td>
<td>14.9</td>
<td>1.1</td>
</tr>
<tr>
<td>T2</td>
<td>30</td>
<td>26.8</td>
<td>28.5</td>
<td>1.5</td>
</tr>
<tr>
<td>T3</td>
<td>30</td>
<td>27</td>
<td>27.9</td>
<td>2.1</td>
</tr>
<tr>
<td>T4</td>
<td>26</td>
<td>23.4</td>
<td>23.4</td>
<td>1.6</td>
</tr>
<tr>
<td>T5</td>
<td>23</td>
<td>20.5</td>
<td>22.0</td>
<td>1.0</td>
</tr>
<tr>
<td>T6</td>
<td>28</td>
<td>27.2</td>
<td>27.6</td>
<td>0.4</td>
</tr>
</tbody>
</table>

VI. CONCLUSIONS

QoS improvement in AC real-time system without violating the precedence-power-temperal constraints has become an active research topic in recent time. Accuracy of such AC tasks can be stimulated by executing more from their optional parts along with executing their respective mandatory parts. In this paper, ACCURATE proposed (i) an efficient scheduling strategy towards maximizing result-accuracy for a set of AC real-time applications modeled as PTGs on multi-cores, along with (ii) an online cache based mechanism towards further refinement of the result-accuracy together with reducing run-time energy of the underlying circuitry.

Once the tasks are allocated to the processor-cores by employing an ILP based scheduling technique, our online strategy orchestrates a DAM based way-sharing mechanism at the shared LLC to significantly reduce the running time of the applications. This improved performance is traded off towards enhancing result-accuracy by executing more workload from the optional part of the applications and by turning off a controlled number of LLC ways to enhance energy efficiency, dynamically, while respecting the system-wide constraints. Our evaluation reveals that, the offline strategy of ACCURATE achieves 85% QoS while maintaining the system constraints and the cache based online mechanism reduces LLC leakage by 36% on an average with 24% average gain in EDP and 6.4% improvement in performance for our 4-core based chip-multiprocessor baseline system.

ACKNOWLEDGMENTS

This work is supported by the UK Engineering and Physical Sciences Research Council (EPSRC) through grants EP/R02572X/1, EP/P017487/1, EP/V000462/1, EP/V034111/1, and is also funded by Marie Curie Individual Fellowship (MSCA-IF), EU (Grant Number 898296).

REFERENCES


This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/


Sangeet Saha is currently associated with Department of Computer Science, University of Huddersfield, UK as a Lecturer and with the Embedded and Intelligent Systems (EIS) Research Group, University of Essex, UK as a visiting fellow. Sangeet received his PhD degree in Information Technology from University of Calcutta, India in 2018 as a TCS (TATA) research scholar. After submitting his PhD thesis in 2017, he worked as a visiting scientist at Indian Statistical Institute (ISI) Kolkata, India. From May 1, 2018, to October 31, 2021, he was a Senior Research Officer in EPSRC National Centre for Nuclear Robotics, based in the EIS Lab, University of Essex. He is a recipient of YERUN Research Mobility Award 2021. His current research interests include real-time scheduling, scheduling for reconfigurable computers, real-time and fault-tolerant embedded systems, and cloud computing. He published several of his research contributions in conferences like CODES+ISSS, ISCAS, NASA AHS, etc. and in journals like ACM TODAES, IEEE TMSCS, and Journal of Supercomputing (Springer).

Shousaku Chakraborty is currently associated with Department of Computer Science, NTNU, Trondheim, Norway as a Post-Doc researcher (through Marie Curie Individual Fellowship from European Union [Grant No. 898296]). Primarily, his broad area of research is Computer Architecture, however, specifically, 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, ASP-DAC, CODES+ISSS, ACM SAC, IPDPS, VLSI-SoC, GLSYLSI etc. He has also published several of his research outcomes in journals like ACM TACO, ACM TECS, ACM TODAES, IEEE T-SUSC, The Journal of Supercomputing (Springer), etc. He also serves as reviewers of Journal of Supercomputing (Springer), ACM TECS, etc. Prior to his joining at NTNU, Norway, Shousaku earned his PhD degree in Computer Science and Engineering from IIT Guwahati, India in February 2018.

Xiaojun Zhai (Senior Member, IEEE) received the Ph.D. degree from the University of Hertfordshire, U.K., in 2013. He is currently a Senior Lecturer with the Embedded Intelligent Systems Laboratory, University of Essex. He has authored/co-authored over 100 scientific articles in international journals and conference proceedings. His research interests include the design and implementation of the digital image and signal processing algorithms, custom computing using FPGAs, embedded systems, and hardware/software co-design. He is also a member of BCS, and a Fellow of HEA.

Shaobai Ehsan received the B.Sc. degree in electrical engineering from the University of Engineering and Technology, Taxila, Pakistan, in 2003, and the Ph.D. degree in computing and electronic systems (with specialization in computer vision) from the University of Essex, Colchester, U.K., in 2012. He has an extensive industrial and academic experience in the areas of embedded systems, embedded software design, computer vision, and image processing. His current research interests are in intrusion detection for embedded systems, local feature detection and description techniques, and image feature matching and performance analysis of vision systems. He was a recipient of the University of Essex Post Graduate Research Scholarship, the Overseas Research Student Scholarship, and the prestigious Sullivan Doctoral Thesis Prize awarded annually by the British Machine Vision Association.

Klaus McDonald-Maier is currently the Head of the Embedded and Intelligent Systems Laboratory, University of Essex, Colchester, U.K. He is also the Chief Scientist with UltraSoC Technologies Ltd., the CEO of Metrac Ltd., and a Visiting Professor with the University of Kent. His current research interests include embedded systems and system-on-chip 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.