Using Imprecise Computing for Improved Non-Preemptive Real-Time Scheduling

Lin Huang*, Youmeng Li†, Sachin S. Sapatnekar‡ and Jiang Hu*
*Department of Electrical and Computer Engineering, Texas A&M University
†Department of Software Engineering, Tianjin University
‡Department of Electrical and Computer Engineering, University of Minnesota
liushui0820@tamu.edu, liyoumeng@tju.edu.cn, sachin@umn.edu and jianghu@tamu.edu

***Abstract*—Conventional hard real-time scheduling is often overly pessimistic due to the worst case execution time estimation. The pessimism can be mitigated by exploiting imprecise computing in applications where occasional small errors are acceptable. This leverage was previously investigated for preemptive scheduling. We study how to make use of imprecise computing in uniprocessor non-preemptive real-time scheduling, which is known to be more difficult than its preemptive counterpart. Several heuristic algorithms are developed for periodic tasks with independent or cumulative errors due to imprecision. Simulation results show that the proposed techniques can significantly improve task schedulability and achieve desired accuracy–schedulability tradeoff. The benefit is further confirmed by a prototyping implementation in Linux system.**

I. Introduction

Imprecise computing, which is sometimes called approximate computing [1], is an unconventional approach to low power systems and is recently being actively studied. It is based on the observation that occasional small computing errors are acceptable for applications like video/audio processing, recognition and mining, which are of growing interest. By intentionally allowing such errors in design, a computing system can operate with reduced power or improved speed.

Imprecise computing can benefit real-time systems as well. In a virtual reality tracking system, for instance, deadline violations cause video discontinuity, which may hamper mission success. By contrast, errors at a few pixels of a small number of frames are usually indiscernible to human eyes and have much less serious consequence than deadline violations. As imprecise computing permits relatively short execution time, it can be adopted to avoid deadline violations when computing resource is under stress. In general, it offers an opportunity for improving schedulability and reducing the pessimism in real-time scheduling. Indeed, the benefit of imprecise computing for real-time scheduling was noticed a long time ago [2]. The early works [2]–[4] are restricted to preemptive cases. Meanwhile, imprecision must be applied prudently so that errors are well controlled in all conditions. In fact, our techniques can achieve such control for realistic computing cases such as IDCT and Newton-Raphson method.

DAC '18, June 24-29, 2018, San Francisco, CA, USA
 ACM 978-1-4503-5700-5/18/06 $15.00
https://doi.org/10.1145/3195970.3196134

In this work, we investigate how to use imprecise computing for improving non-preemptive real-time scheduling of independent tasks on uniprocessor. Compared to preemptive cases, non-preemptive scheduling implies less context switching overhead and has more predictable characteristics [5], [6]. In some scenarios, task preemption is even impossible or formidably expensive [5]. On the other hand, it is proved [5] that non-preemptive scheduling for periodic tasks with specific release times is NP-hard and its schedulability test is also NP-hard. Considering imprecise computing greatly escalates scheduling complexity and makes some preemptive problems NP-hard [2].

We propose several heuristic algorithms for scheduling periodic tasks and considering imprecision with independent or cumulative errors. To the best of our knowledge, this is the first work on using imprecise computing for non-preemptive real-time scheduling. Moreover, it is not trivial to extend preemptive techniques [2], [3] to our case. For tasks satisfying schedulability conditions in imprecise mode, our heuristics can guarantee that there is no deadline violation. In these algorithms, errors due to imprecision are controlled either offline with guarantee or online in the best effort. Simulation results from random and realistic cases indicate that our scheduling methods can indeed improve schedulability and significantly mitigate the pessimism of the worst case execution time model. The advantage is further demonstrated by a prototyping implementation in Linux system.

II. Preliminaries

A. Non-preemptive Real-Time Scheduling

We consider to schedule a set of independent tasks $T = \{\tau_1, \tau_2, ..., \tau_n\}$ onto a single processor. Each task $\tau_i, i = 1, 2, ..., n$, is a 3-tuple (r_i, c_i, d_i) representing task release time, execution time and deadline. A periodic task τ_i is composed by multiple jobs $\{\tau_{i,1}, \tau_{i,2}, ...\}$, where $\tau_{i,j}$ is the j_{th} occurrence of task τ_i. The jobs of τ_i are released with period p_i, i.e., their release times and deadlines satisfy $d_{i,j} = r_{i,j} + p_i = r_{i,j+1}, j = 1, 2, ...$ In a hard real-time system, the execution of every task must be completed before its deadline. In practice, an execution time c_i may vary in a wide range. In order to ensure that no deadline violation is incurred by this uncertainty, the Worst Case Execution Time (WCET) $w_i > c_i$ is conventionally used in scheduling. We focus on non-preemptive systems, where once a job starts, it must execute continuously on the processor till its completion.

B. Schedulability of Non-Preemptive Real-Time Scheduling

The schedulability conditions for non-preemptive scheduling on uniprocessor are derived in [5] and shown as follows.

Theorem 1. *Let $T = \{\tau_1, \tau_2, ..., \tau_n\}$, where $\tau_i = (p_i, w_i)$, be a set of periodic tasks sorted in non-decreasing order of period, i.e., if $i < j$, then $p_i \leq p_j$. If the following two conditions are satisfied, then T is schedulable.*

$$\sum_{i=1}^{n} \frac{w_i}{p_i} \leq 1 \quad (1)$$

$$w_i + \sum_{j=1}^{i-1} \left\lfloor \frac{L-1}{p_j} \right\rfloor \cdot w_j \leq L, \ \forall i, 1 < i \leq n, \ \forall L, \ p_1 < L < p_i. \quad (2)$$

For jobs of a periodic task τ_i, their release time $r_{i,j}$ and deadline $d_{i,j}$ can be uniquely decided by the initial job release time $r_{i,1}$ and period p_i. Theorem 1 is necessary and sufficient for arbitrary initial release time, and thus only p_i needs to be considered here. The WCET w_i is employed to estimate c_i. Hence, a task is characterized by (p_i, w_i) instead of (r_i, c_i, d_i). In the sequel, release time r_i is still needed for considering specific tasks. For a specific set of release times, the theorem is sufficient but no longer necessary. Moreover, finding necessary and sufficient conditions for specific release times is an NP-hard problem. It is proved in [5] that if a set of tasks are schedulable according to Theorem 1, then non-preemptive Earliest Deadline First (EDF) scheduling [7] can always find a feasible schedule.

Condition (1) is to ensure the overall processor utilization does not exceed 1. This condition alone is the necessary and sufficient condition for a preemptive system to be schedulable, and the complexity of verifying this condition is constant. Condition (2) is to ensure that the workload for any time interval of any task period is no greater than the length of the interval. The complexity of evaluating this condition is $O(p_n)$, where p_n is the largest task period, and therefore is pseudo-polynomial.

C. Imprecise Computing in Non-Preemptive Scheduling

Imprecise computing can be realized by either accuracy configurable circuits [8], [9] or algorithmic imprecision. Please note the imprecise computing is for only datapath or numerical computations, and not applicable to control flow or state computations. In a non-preemptive system, different task executions can be performed with different accuracy levels but the accuracy of one execution cannot be changed in the middle. We consider only one imprecision level in this work. Since our approaches are fundamentally enumerating discrete solution options, additional imprecision levels would not entail significant algorithm change.

The WCET x_i of task τ_i in imprecision mode must satisfy $x_i < w_i$, where w_i is the WCET for accurate mode. Meanwhile, an execution of job $\tau_{i,j}$ in imprecision mode produces a computing error $\epsilon_{i,j}$. Please note the error is single valued and would require composition of multiple errors from a multi-output system. By statistical analysis and pre-characterization, the mean error e_i for task τ_i can be obtained prior to scheduling. In independent error model, an error of job $\tau_{i,j}$ does not carry over to its subsequent job $\tau_{i,j+1}$ even if $\tau_{i,j+1}$ is in imprecision mode. In video rendering, for example, image processing error of one non-reference frame does not affect the next frame. In this scenario, one wishes to minimize the average error of a task. In the cumulative error model, an error of job $\tau_{i,j}$ propagates to its subsequent job. For example, in a target tracking computation, an error at one moment j is inherited in the computation of the next moment $j+1$ and can be eliminated only when the computation at moment $j+1$ is in accurate mode. To restrain such cumulative error, the number of consecutive jobs in imprecision mode must be limited.

III. Online Scheduling of Tasks with Independent Errors

In this section, an online algorithm is introduced for solving periodic tasks where imprecision errors are independent. The problem formulation is given below.

Problem 1. *Given a set of periodic tasks $T = \{\tau_1, \tau_2, ..., \tau_n\}$, decide if each job $\tau_{i,j}$ is executed in accurate or imprecise mode, and its start time $s_{i,j}$ such that $r_{i,j} \leq s_{i,j}$, $s_{i,j} + c_{i,j} \leq r_{i,j} + p_i$ and the total error among all jobs in a hyper-period $\sum_{\forall i,j \in P} \epsilon_{i,j}$ is minimized.*

Please note hyper-period P is the least common multiple of all task periods. $c_{i,j}$ is the actual execution time regardless the accuracy level of job $\tau_{i,j}$ execution, and $\epsilon_{i,j}$ is the actual error when $\tau_{i,j}$ is executed in imprecision mode. If a given set of tasks passes its schedulability check using Theorem 1 based on imprecision mode WCET, our algorithm can guarantee that there is no deadline violation. A main effort of the algorithm is to increase the use of accurate mode in order to minimize imprecision errors. Our algorithm is EDF with Explicit Slack Reclamation, where the accuracy selection is based on explicit slack reclamation with constant complexity. When a job is to start, the algorithm checks if there is enough slack available for this job to be executed in accurate mode. There are three kinds of slacks: (1) individual slack; (2) idle-time slack; and (3) inter-job slack.

An individual slack $\psi_{i,j}$ is intrinsically available to every job $\tau_{i,j}$ and is estimated with the initial schedulability check before any job is started. A scaling factor γ is associated with every condition in Theorem 1 and their values can be found by solving the following equations:

$$\gamma \sum_{i=1}^{n} \frac{x_i}{p_i} = 1$$

$$\gamma_i^L \cdot (x_i + \sum_{j=1}^{i-1} \left\lfloor \frac{L-1}{p_j} \right\rfloor \cdot x_j) = L, \forall i, 1 < i \leq n, \forall L, p_1 < L < p_i.$$

We define γ_{min} to be the minimum γ value, i.e., $\gamma_{min} = \min_{\forall i, \forall L}(\gamma, \gamma_i^L, ...)$. If the set of tasks are schedulable when all of their jobs are in imprecision mode, then $\gamma_{min} \geq 1$. Hence, the individual slack $\psi_{i,j} = (\gamma_{min} - 1) \cdot x_i$. The individual slacks are computed only once at the beginning and their values can be used repeatedly throughout the online scheduling.

When a job $\tau_{i,j}$ is being scheduled online, and its nominal finish time $f_{i,j}$ is less than the minimum between its deadline $d_{i,j}$ and the release time r_{next} of its next job, then the idle time slack is $\psi_{i,j}^{idle} = \min(d_{i,j}, r_{next}) - f_{i,j}$. The nominal finish time $f_{i,j}$ for a job $\tau_{i,j}$ is the finish time assuming no slack reclamation is conducted and can be estimated as $current\_time + x_i + \psi_{i,j}^{k,l}$, where $\psi_{i,j}^{k,l}$ is inter-job slack introduced as follows.

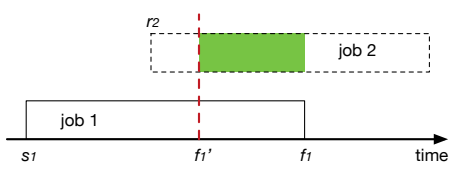

Fig. 1. If job 1 actually finishes at f_1', which is earlier than its nominal finish time f_1, and job 2 is to be executed next, job 1 provides inter-job slack to job 2 as in the green region. The release times for job 2 is $r2$.

An inter-job slack $\psi_{i,j}^{k,l}$ is the slack generated by job $\tau_{k,l}$ due to its early completion and passed to its next job $\tau_{i,j}$. This concept is elaborated through the example in Figure 1, where job 1 finishes at f_1' which is earlier than its nominal finish time f_1 and the next job according to EDF (job 2 here) has release time earlier than f_1. Then, the processor time from f_1' to f_1 is a slack time that can be applied to job 2, which is the green region in Figure 1. In general, the inter-job slack produced by $\tau_{k,l}$ and passed to $\tau_{i,j}$ is defined by

$$\psi_{i,j}^{k,l} = \max(f_{k,l} - \max(r_{i,j}, f'_{k,l}), 0) \tag{3}$$

Every job intrinsically has individual slack, which can be zero. It can be consumed by choosing accurate mode for this job. Even if a job is executed in imprecision mode, its individual slack expires when the job is completed. However, an individual slack can be recycled as inter-job slack. An idle-time slack generates opportunistically, and expires if it is not consumed. An inter-job slack $\psi_{i,j}^{k,l}$ can be consumed by executing job $\tau_{i,j}$ in accurate mode. If it is not consumed, it can assist $\tau_{i,j}$ to generate new inter-job slack or expires.

In the EDF with explicit slack reclamation algorithm, we use the following slack based check. When a job $\tau_{i,j}$ is being scheduled, its total slack is evaluated by

$$\psi_{i,j}^{total} = \psi_{i,j} + \psi_{i,j}^{idle} + \psi_{i,j}^{k,l}.$$

If $\psi_{i,j}^{total} \geq w_i - x_i$, this job is executed in accurate mode and otherwise in imprecision mode.

IV. Collaborative Scheduling of Periodic Tasks with Independent Errors

This section introduces three collaborative methods for solving Problem 1. Each of these methods is composed by an offline scheduling part and an online adjustment part.

A. Offline ILP and Online Adjustment

The offline scheduling and accuracy selection is conducted for one hyper-period using integer linear programming (ILP). During task executions, some jobs are opportunistically adjusted from imprecise to accurate mode. The adjustment is performed with reference to the ILP result and thus no schedulability check as Theorem 1 is needed. The ILP result provides an upper bound guarantee to imprecision errors.

A decision variable $y_{i,j}$ is defined to be 1 if $\tau_{i,j}$ is executed in imprecision mode and 0 otherwise. The offline scheduling finish time for job $\tau_{i,j}$ is denoted as $\hat{f}_{i,j}$. An indicator function $u_{i,j}(t)$ is equal to 1 if time t is in-between the start and finish time of $\tau_{i,j}$, and 0 otherwise. The ILP formulation is as follows.

$$\begin{aligned}
\underset{y}{\text{minimize}} \quad & \sum_{\forall \tau_{i,j} | [r_{i,j}, d_{i,j}] \subseteq [0,P]} e_i \cdot y_{i,j} \\
\text{subject to} \quad & s_{i,j} \geq r_{i,j}, && \forall \tau_{i,j} | [r_{i,j}, d_{i,j}] \subseteq [0,P] \\
& \hat{f}_{i,j} = s_{i,j} + w_i + (x_i - w_i) \cdot y_{i,j}, && \forall \tau_{i,j} | [r_{i,j}, d_{i,j}] \subseteq [0,P] \\
& \hat{f}_{i,j} \leq r_{i,j} + p_i, && \forall \tau_{i,j} | [r_{i,j}, d_{i,j}] \subseteq [0,P] \\
& \sum_{\forall \tau_{i,j} | [r_{i,j}, d_{i,j}] \subseteq [0,P]} u_{i,j}(t) \leq 1 && \forall t, 0 < t \leq P \\
& s_{i,j} \in \mathbb{Z}_{\geq 0}, \quad y_{i,j} \in \{0,1\} && \forall \tau_{i,j} | [r_{i,j}, d_{i,j}] \subseteq [0,P]
\end{aligned}$$

where $s_{i,j}$ is the start time for job $\tau_{i,j}$, x_i is the constant WCET for task τ_i in imprecision mode and P is the hyper-period.

In the online adjustment, if a job $\tau_{i,j}$ finishes earlier than the offline finish time $\hat{f}_{i,j}$ specified by ILP, the next job can start immediately without waiting till its start time specified by ILP. The order of job executions is fixed and conforms to the ILP result. When a job $\tau_{i,j}$ is able to start at current time t_{cur} and $y_{i,j} = 1$, it is executed in accurate mode if and only if $t_{cur} + w_i \leq \hat{f}_{i,j}$, which is the finish time by ILP. Thus, the complexity of this online adjustment is constant.

B. ILP with Post-Processing and Online Adjustment

The ILP method described in Section IV-A can guarantee the optimal solution according to the WCET estimation, but the actual execution time is usually shorter than WCET. We propose a post-processing to the ILP such that online adjustment may have more opportunities to improve the result. Meanwhile, the ILP constraints and optimality are not affected. The offline post-processing is based on three observations.

- For a job $\tau_{i,j}$, if the processor is idle after $\hat{f}_{i,j}$, which is obtained from ILP, we postpone its $s_{i,j}$ as much as possible without missing its deadline or conflicting with the execution of its next job. This is because the online accuracy adjustment is based on the condition of $t_{cur} + w_i \leq \hat{f}_{i,j}$ and increased $s_{i,j}$ as well as $\hat{f}_{i,j}$ would improve the chance of changing a job from imprecision to accurate mode. The increased offline $s_{i,j}$ does not affect the actual start time at runtime, when a job always starts immediately upon the availability of processor without waiting for $s_{i,j}$.
- If ILP schedules job $\tau_{k,l}$ to start immediately after $\tau_{i,j}$ and assigns both the jobs with the same accuracy level, we may swap their execution order such that the job with earlier release time starts earlier. This is based on the observation that a job with early release time has relatively large chance to reclaim the slack generated from prior job executions.
- If ILP schedules an accurate job to be executed right after an imprecise job, we swap the order of the two jobs subject to release time and deadline constraints. If an imprecise job is scheduled to be executed later, it may have more chance of reclaiming slacks from prior job executions. For example, in Figure 2, the imprecise job $\tau_{1,4}$ may reclaim slacks from $\tau_{3,2}$ after the swapping.

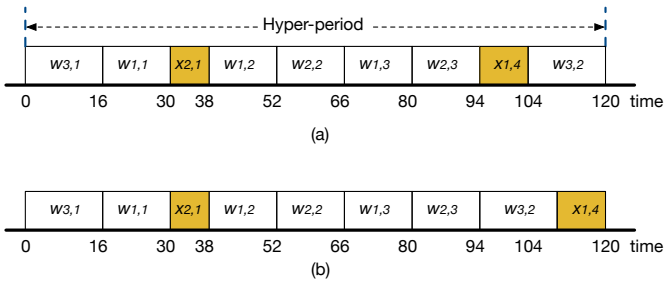

Fig. 2. (a) ILP scheduling; (b) swapping imprecise job $\tau_{1,4}$ to be executed later in post-processing.

Since all changes in the post processing are monotone and cannot be reversed, convergence is guaranteed. After the post-processing, the online adjustment part is the same as Section IV-A.

C. Flipped EDF and Online Adjustment

We propose another offline scheduling algorithm with all jobs being in imprecision mode. When it works together with online adjustment, this offline scheduling achieves comparable or even better results compared to the ILP-based collaborative methods. For a hyper-period P, this algorithm schedules jobs from the last moment of P and proceeds backward to time 0, the starting point of hyper-period. Among all unscheduled jobs, it always chooses the job with the latest release time to schedule first. It selects imprecision mode for every job and schedules a job as late as possible without deadline violation or conflicting with jobs that have already been scheduled. One can think of this algorithm as EDF being performed in a flipped manner, where time axis is reversed and the release time and deadline of each job are exchanged. If the original EDF is like as-soon-as-possible scheduling, our flipped EDF is like as-late-as-possible scheduling.

According to [5], EDF can guarantee to find a feasible solution if the schedulability test of Theorem 1 is passed. Since the flipped EDF is fundamentally equivalent to EDF, it enjoys the same guarantee of finding feasible solution. After the offline flipped EDF, the online adjustment is performed in the same way as Section IV-A.

V. Scheduling Periodic Tasks with Cumulative Errors in Imprecision

When errors are cumulative, an error at one job $\tau_{i,j}$ can carry over to its next job $\tau_{i,j+1}$ and so on if the jobs of task τ_i are in imprecision mode contiguously. The errors can be cleared out only when at least one job execution is in accurate mode. As such, users wish to avoid that the jobs of a task continuously operate in imprecision mode. Hence, we attempt to constrain the number of consecutive imprecise job executions for each task like [2] and the problem formulation is as follows.

Problem 2. *Given a set of periodic tasks $T = \{\tau_1, \tau_2, ..., \tau_n\}$, decide if each job $\tau_{i,j}$ is executed in accurate or imprecision mode, and its start time $s_{i,j}$ such that $r_{i,j} \leq s_{i,j}$, $s_{i,j} + c_{i,j} \leq r_{i,j} + p_i$ and the number of consecutive jobs in imprecision mode for each task τ_i is no greater than constraint B_i.*

A. Online Heuristic

This online heuristic schedules jobs using EDF. In addition, it decides if to execute a job in accurate or imprecision mode. When a job $\tau_{i,j}$ is scheduled to start, there are four scenarios for deciding its accuracy mode. In the first scenario, there is error constraint violation if $\tau_{i,j}$ is executed in imprecision mode while schedulability check passes for accurate mode. Then, accurate mode is selected for this scenario. The second scenario is mirror case to the first one, where an imprecise execution would not violate the error constraint but accurate execution fails schedulability check. Imprecision mode is selected for this scenario. The third scenario is a difficult one, where both imprecision mode would violate error constraint and accurate mode does not satisfy schedulability conditions. In this scenario, we choose imprecision mode such that the theoretical guarantee of no deadline violation is still fulfilled.

The fourth scenario seems easy but actually can affect subsequent selections. This is when imprecision mode would not violate the error constraint and accurate mode passes schedulability check. We define and compare error slack and latency slack to tell if choose accurate mode for less error or imprecision mode for less risk of deadline violation, which can eventually become error constraint violation according to the third scenario. The error slack is defined as $ErrorSlack_i = (B_i - \phi_i)/B_i$, where ϕ_i is the number of consecutive imprecise executions of jobs of task τ_i immediately before $\tau_{i,j}$. The latency slack is defined as $LatencySlack_{i,j} = (d_{i,j} - s_{i,j} - w_i)/p_i$. Please note the error slack is normalized within $(0, 1]$ while the latency slack is normalized within $[0, 1]$. Then, we check the ratio $\rho = LatencySlack_{i,j}/ErrorSlack_i$. If $\rho < \theta$, where θ is a user specified threshold, job $\tau_{i,j}$ is executed in imprecision mode as the latency slack is tighter. The schedulability check here can be realized using the explicit slack reclamation as in Section III.

B. Offline Dynamic Programming

The offline dynamic programming is a traversal of all jobs in a super period following the EDF principle. A super period is the minimum consecutive set of hyper-periods that can cover all scenarios of errors satisfying constraint B_i for all tasks. When a job is encountered in the traversal, both its accurate and imprecise executions are considered as two candidate solutions. Thus, the dynamic programming is an enumeration of different precision options in a decision tree. However, we do not need to examine the entire tree (or solution space). If a candidate solution has either deadline or error constraint violation, it is pruned without being propagated to consider with the next job. Please note that different precision choices may lead to different job execution orders, although all of them follow the principle of EDF. To further improve the computing efficiency, we propose two other solution pruning techniques.

The first pruning technique is based on solution dominance. Consider a set of candidate solutions $\mathcal{S}^k = \{S_1^k, S_2^k, ...\}$ for the same k jobs that have been processed in the traversal so far. A solution $S_i^k \in \mathcal{S}^k$ is characterized by $n+1$ tuple $(f_i^k, \lambda_{i,1}^k, \lambda_{i,2}^k, ..., \lambda_{i,n}^k)$, where f_i^k is the finish time of all the k jobs and $\lambda_{i,l}^k$ is the cumulative error for task τ_l. If $f_i^k = f_j^k$ and $\lambda_{i,l}^k \geq \lambda_{j,l}^k, 1 \leq l \leq n$, then solution S_i^k is dominated by S_j^k and can be pruned without being further propagated.

The second pruning technique is based on processor utilization. For a solution S_i^k, we estimate the best case processor utilization for the unprocessed jobs in the remaining time of the super period. By "best case", we mean the allowable error budget is maximally used by the unprocessed jobs. If this utilization exceeds 1, the corresponding solution is pruned without further propagation.

Proposition 1. *The dynamic programming algorithm guarantees to find a precision assignment for EDF to satisfy both the deadline and error constraints if such assignment exists.*

Proof: Omitted due to space limit. ■

VI. Experiment Results

In our experiment, we compare the following methods:

- EDF-Accurate: EDF scheduling with all jobs in accurate mode.
- EDF-Imprecise: EDF scheduling with all jobs in imprecision mode.
- EDF+ESR: our EDF scheduling with explicit slack reclamation for periodic tasks with independent errors (Section III).
- ILP+OA: our ILP scheduling with online adjustment for periodic tasks with independent errors (Section IV-A).

TABLE I. TESTCASE CHARACTERISTICS AND SCHEDULABILITY.

Tesecases	#tasks	Utilization Accurate	#jobs/P	Schedulability	
				Accurate	Imprecise
Rnd1	3	1.13	9	No	Yes
Rnd2	3	1.88	9	No	No
Rnd3	5	1.93	15	No	Yes
Rnd4	3	1.6	9	No	Yes
Rnd5	3	0.45	17	No	Yes
Rnd6	7	3.8	22	No	Yes
Rnd7	10	4.43	38	No	Yes
Rnd8	12	2.91	60	No	Yes
Rnd9	15	1.93	24	No	Yes
Rnd10	17	4.99	126	No	Yes
Rnd11	20	3.57	105	No	Yes
Rnd12	22	5.47	130	No	Yes
Rnd13	25	7.12	163	No	Yes
IDCT	5	1.02	35	No	No

- ILP+Post+OA: our ILP and post-processing scheduling with online adjustment for periodic tasks with independent errors (Section IV-B).
- Flipped EDF: our flipped EDF scheduling with online adjustment for periodic tasks with independent errors (Section IV-C).
- EDF+ESR(C): our EDF scheduling with explicit slack reclamation for periodic tasks with cumulative errors (Section V-A).
- DP(C): our dynamic programming based offline scheduling with accuracy selection for periodic tasks with cumulative errors (Section V-B).

A. Simulation Results

Simulations are performed on 13 random testcases and 1 realistic case. In the random cases, actual job execution times are modeled as random variables following Gaussian distribution. The WCET is obtained by the mean value plus 6σ of the distribution, which is further augmented by a margin. The WCET to the best case execution time ratio is around 10. Errors from imprecision executions are also simulated as random variables satisfying Gaussian distribution. The error statistics are derived based on accuracy configurable circuit design [9]. The realistic case is IDCT computation on grayscale and RGB images of various resolution, which form 5 different tasks. The WCET and imprecise errors of IDCT computation are obtained from actual measurement. The testcase characteristics are summarized in column 2, 3 and 4 of Table I. The third column is processor utilization when every job is executed in accurate mode and the fourth column is the number of jobs in hyper-period. Experimental results are based on simulating 10K hyper-periods for each task. Schedulability check according to Theorem 1 is performed on these testcases for two scenarios: (1) all jobs in accurate mode and (2) all jobs in imprecise mode. The results are listed in the rightmost two columns of Table I. None of these cases is schedulable in accurate mode according to Theorem 1. When all the jobs are in imprecision mode, every testcase except Rnd2 and IDCT is schedulable based on Theorem 1.

The main results are shown in Table II. The EDF-accurate results do not have errors but have many deadline violations. Please note EDF-accurate can successfully schedule Rnd1 and Rnd5, although the cases fail the schedulability check. This discrepancy is largely due to the difference between actual execution time and WCET. All the other methods can always satisfy deadline constraints, but usually result in errors caused by imprecise computing. Our post-processing can reduce the normalized average error from 0.63 of the ILP to 0.55. The best result comes from Flipped EDF, which finds all accurate executions without deadline violation for Case 1 and 5. However, the Flipped EDF is restricted to cases with prior knowledge on task release time compared to online approaches. Moreover, its offline computation does not provide guarantee on errors like the ILP-based approach.

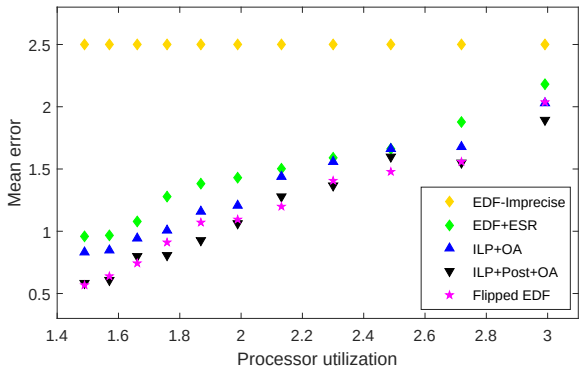

Fig. 3. Mean error versus utilization.

We study the error versus processor utilization tradeoff and the result is depicted in Figure 3. The utilization is $\sum_{i=1}^{n} w_i/p_i$ based on the WCET of accurate executions. Please note utilization is greater than 1 for all the cases and this means these tasks are not schedulable according to Theorem 1 using the accurate execution WCET. However, all of these cases are successfully scheduled without deadline violation by using imprecise computing. Except EDF-imprecise, which always uses imprecision mode, all methods can reduce errors when utilization decreases. Among the methods, ILP+Post+OA and Flipped EDF produce the best result. We evaluated the computation runtime of these methods. Online computing usually takes a few μs and the ILP runtimes range from seconds to minutes.

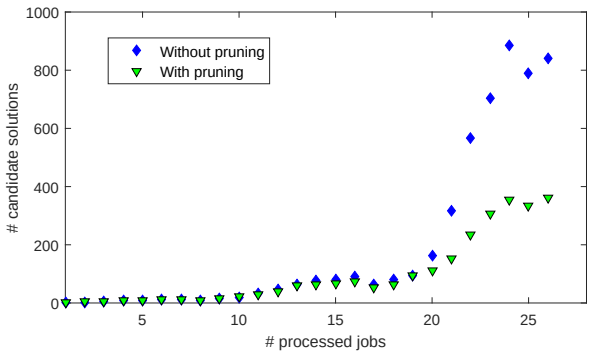

Fig. 4. The number of candidate partial solutions with and without pruning.

The simulation results for periodic tasks with cumulative errors are shown in Table III. EDF+ESR(C) can guarantee to satisfy all deadline constraints. We intentionally make the cases very tight such that error constraint violations occur. The error constraint B_i for a task τ_i ranges from 1 to 6 in these cases. The results of EDF-Accurate and EDF-Imprecise are not shown here as the former one never causes errors and the later one has 100% error constraint violations. For cases Rnd1, Rnd4, Rnd5, Rnd9 and IDCT, the DP(C) can find feasible solutions satisfying both deadline and error constraints while EDF+ESR(C) results in violations of error constraints. Figure 4 shows the effectiveness of solution pruning in the dynamic

TABLE II. SIMULATION RESULTS FOR PERIODIC TASKS WITH INDEPENDENT ERRORS. ERROR STANDARD DEVIATION IS σ.

Test cases	EDF-Accurate	EDF-Imprecise		EDF+ESR(I)		ILP+OA		ILP+post+OA		Flipped EDF	
	Deadline violations	Mean error	σ	Mean error	σ	Mean error	σ	Mean error	σ	Mean error	σ
Rnd1	0	2.59	1.82	1.15	1.21	0.40	0.72	0.22	0.56	0	0
Rnd2	55%	2.16	1.64	1.51	1.39	1.19	1.19	0.67	0.86	0.72	0.93
Rnd3	29%	51.85	39.64	27.65	34.55	32.03	35.65	23.79	33.61	19.86	32.42
Rnd4	38%	3.60	3.44	2.77	3.35	1.10	1.08	0.94	0.99	2.06	3.07
Rnd5	0	0.62	0.31	0.22	0.19	0.29	0.22	0	0	0	0
Rnd6	27%	2.58	1.17	2.25	1.07	2.39	1.11	2.10	1.05	2.04	1.04
Rnd7	29%	82.03	27.25	68.87	25.01	65.26	24.25	62.78	23.49	55.83	22.13
Rnd8	43%	82.14	21.79	58.67	18.21	52.67	17.37	43.25	15.56	41.53	14.96
Rnd9	4%	2.59	1.12	1.23	0.77	0.44	0.46	0.32	0.39	0.32	0.39
Rnd10	28%	20.08	3.74	15.24	3.26	13.60	3.07	11.99	2.85	12.36	2.93
Rnd11	31%	5.09	1.03	3.93	0.90	3.09	0.83	2.51	0.74	2.40	0.71
Rnd12	24%	10.09	1.81	8.03	1.62	6.84	1.51	6.79	1.50	6.85	1.52
Rnd13	18%	87.25	27.24	70.10	24.78	42.95	19.30	41.17	18.85	44.90	19.89
IDCT	1%	2.71	1.52	0.81	0.66	0.17	0.12	0.03	0.02	0.02	0.02
Average	23%	25.38	-	18.74	-	15.89	-	14.04	-	13.49	-
Normalized	-	1	-	0.74	-	0.63	-	0.55	-	0.53	-

TABLE III. STRESS TEST RESULTS FOR PERIODIC TASKS WITH CUMULATIVE ERRORS.

Testcases	Rnd1	Rnd2	Rnd3	Rnd4	Rnd5	Rnd6	Rnd7
Error Violations (EDF+ESR(C))	20%	26%	28%	21%	6%	53%	50%
Feasible?(DP(C))	Yes	No	No	Yes	Yes	No	No
Testcases	Rnd8	Rnd9	Rnd10	Rnd11	Rnd12	Rnd13	IDCT
Error Violations (EDF+ESR(C))	40%	12%	39%	46%	58%	49%	13%
Feasible?(DP(C))	No	Yes	No	No	No	No	Yes

programming for case Rnd7. We observe that the pruning can greatly reduce runtime as well.

B. Linux Prototyping Results

TABLE IV. TASKS IN LINUX SYSTEM PROTOTYPING.

Task	Accurate WCET(s)	$\hat{\varepsilon}_{accurate}$	Imprecise WCET(s)	$\hat{\varepsilon}_{imprecise}$
τ_1	0.96	0.00001	0.55	20
τ_2	1.21	0.00001	0.27	0.5
τ_3	2.01	0.00001	1.18	5

We implemented EDF-Imprecise, EDF+ESR, Flipped EDF, ILP+Post+OA in Linux 4.6 on a 1200MHz ARM Cortex-A53 processor. The testcase is Newton-Raphson method for solving nonlinear equations numerically. The convergence criterion is $\hat{\varepsilon}_{accurate}$ ($\hat{\varepsilon}_{imprecise}$), which is tight (loose) for accurate (imprecision) mode. The WCETs are obtained by measuring the longest runtime among multiple random tests and augmenting with additional margin. There are three periodic tasks for solving three different kinds of equations. The statistics of this testcase are summarized in Table IV. Please note the equation for task 2 is relatively well behaved such that its execution time can reduce quickly when the convergence criterion is relaxed. The mean error results are depicted in Figure 5. Our ILP+Post+OA and Flipped EDF lead to much smaller errors than EDF-Imprecise. And the relative overhead ratio of these methods is at the level of 0.0001%.

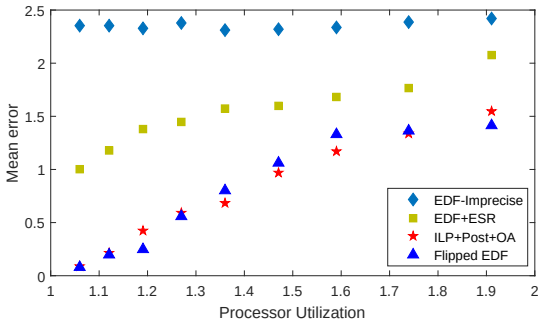

Fig. 5. Mean error versus utilization from Linux prototyping.

VII. CONCLUSIONS

This paper reports the first study result on using imprecise computing for non-preemptive real-time scheduling, to the best of our knowledge. Several heuristic algorithms are developed for periodic tasks with independent or cumulative errors. If a set of tasks pass an initial schedulability check where all jobs are assumed to be executed in imprecision mode, all of our algorithms can guarantee to find solution without deadline violation. At the same time, our algorithms either guarantee certain imprecision error bound or minimize errors in the best effort. Experimental results from both simulation and Linux system prototyping implementation show that using imprecision can greatly improve schedulability while our techniques provide desired error and deadline tightness tradeoff.

ACKNOWLEDGEMENT

This work was supported in part by the NSF under awards CCF-1525925 and CCF-1525749.

REFERENCES

[1] J. Han and M. Orshansky, "Approximate computing: an emerging paradigm for energy-efficient design," *ETS*, pp. 1–6, 2013.

[2] J. W. S. Liu, K. J. Lin, W. K. Shih, A. C. Yu, J. Y. Chung, and W. Zhao, *Algorithms for Scheduling Imprecise Computations*, pp. 203–249. Springer US, 1991.

[3] J.-Y. Chung and J. W. S. Liu, "Algorithms for scheduling periodic jobs to minimize average error," *RTSS*, pp. 142–151, 1988.

[4] H. Aydin, R. Melhem, and D. Mosse, "Optimal scheduling of imprecise computation tasks in the presence of multiple faults," *RTCSA*, pp. 289–296, 2000.

[5] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," *RTSS*, pp. 129–139, 1991.

[6] R. Jejurikar and R. Gupta, "Energy aware non-preemptive scheduling for hard real-time systems," *ECRTS*, pp. 137–144, 2005.

[7] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, pp. 46–61, Jan. 1973.

[8] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, "Design of voltage-scalable meta-functions for approximate computing," *DATE*, pp. 1–6, 2011.

[9] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," *ICCAD*, pp. 48–54, 2013.