Multi-Timescale Mitigation for Performance Variability Improvement in Time-Critical Systems

Ji-Yung Lin, Pieter Weckx, Subrat Mishra, Alessio Spessot, Francky Catthoor *Fellow, IEEE*

***Abstract*—Ensuring timing guarantee is crucial for time-critical applications. However, this task becomes more challenging with the increasing performance variability generated by complicated modern hardware and software. A widespread solution to the problem is real-time scheduling, which depends on worst-case execution time (WCET) and dynamic voltage frequency scaling (DVFS). Although these techniques provide the necessary guarantees, but they also exhibit important limitations from the long switching time of DVFS and the overly pessimistic execution time model of WCET. In this work, a multi-timescale mitigation methodology is proposed to improve the way of tackling performance variability in both timing guarantee and energy saving. By using both the DVFS and heterogeneous datapath (HDP) knobs, this methodology can push the timescale of mitigation down to the sub-millisecond level. Moreover, this methodology can calculate a tight upper bound of execution time at run-time using dynamic scenarios. Simulation shows that the proposed methodology can ensure zero deadline misses with a smaller safety time margin than the method using only DVFS and WCET. This advantage can translate into an energy reduction by half compared to the conventional WCET-based method with a single DVFS knob.**

***Index Terms*—mitigation, variability, real-time scheduling**

I. Introduction

WITH modern hardware and software becoming more dynamic, the increasing performance variability makes the execution time of an application more dispersed. For time-critical applications such as car auto-driving and multimedia streaming, performance variability is a threat to ensuring timing guarantees. There are two main sources of performance variability. First, variability can originate from inherent dynamic structures in software and hardware design [1], such as dynamic program flows, and non-deterministic hardware protocols like cache contention. Second, variability can be caused by the fluctuation in operating conditions [2], such as voltage, temperature, aging, etc.

To deal with performance variability, modern processors are equipped with knobs which allow processors to change their speed at run-time. Some typical knobs include dynamic voltage frequency scaling (DVFS) [14] and task mapping [4]. Among all knobs, DVFS, which allows processor cores to change its supply voltage and clock frequency to vary its speed and energy consumption, is especially popular since it is easy to use and provides a wide range of modes with good speed-energy trade-offs. Nowadays, DVFS plays an important role in modern mitigation methods for performance variability.

On top of the knobs, a run-time controller running the real-time scheduling algorithm decides how to switch the knobs, ensuring all deadlines are met. Some examples of these algorithms can be found in [5] [7] [12]. These algorithms are normally designed on the following principles: it schedules the future jobs based on their worst-case execution time (WCET), and then selects the most energy-saving mode that can still ensure that all deadlines are met. If a job ends earlier than the WCET, the unused execution time becomes extra time resource, which is called *slack*. The slack can be exploited to select a more energy-saving mode to further reduce the energy. The algorithm normally runs in the firmware or the operating system as a background process, and it is required to be lightweight, so that its impact on the performance of the main program is minimized.

The methods of real-time scheduling with WCET and DVFS provide necessary timing guarantee for time-critical applications. Still, these methods exhibit some important limitations. One of the limitations comes from the use of WCET, which is overly pessimistic and much greater than the real execution time. Therefore, the scheduling with WCET often results in overestimation of the execution time, which makes the controller overly conservative and use the modes of unnecessary high speeds and energy consumptions. Furthermore, WCET assumes a static execution time model, which views the job execution time as a predetermined statistical distribution. However, with the highly dynamic modern hardware and software with non-deterministic execution flows, calculation of WCET becomes convoluted and even unrealistic. Therefore, WCET is no longer sufficient to model the execution time of modern applications. To avoid these problems, in previous work some heuristics [13] [21] were invented to predict the execution time. However, these heuristics cannot fully rule out the possibility that the actual execution time is greater than the prediction, so the chances of deadline misses exist. Therefore, we consider that the best way to ensure timing guarantee without being pessimistic at the execution time, is to calculate a tight upper bound of the execution time based on the situations at run-time.

Another limitation comes from the switching time of DVFS, which is at least tens of microseconds [6]. The switching time should be much shorter than the job execution time for two reasons. First, the switching time cannot add too much time overhead to affect the performance of the main application. Second, if the switching time is too long, when a variability

J. Y. Lin and F. Catthoor are with imec, 3001 Leuven, Belgium, and also with ESAT, Katholieke Universiteit Leuven, 3001 Leuven, Belgium (e-mail: ji-yung.lin@imec.be; francky.catthoor@imec.be).

P. Weckx, S. Mishra and A. Spessot are with imec, 3001 Leuven, Belgium (e-mail: pieter.weckx@imec.be; subrat.mishra@imec.be; alessio.spessot@imec.be).

 https://doi.org/10.1109/TVLSI.2022.3177861

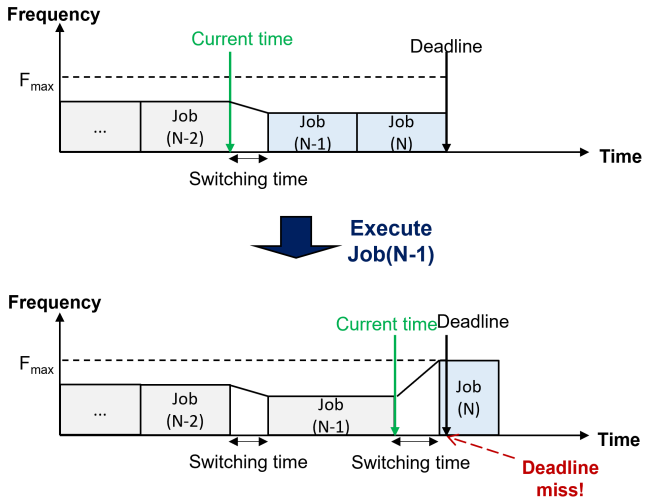

Fig. 1. Illustration of an example where the application misses its deadline due to the switching time and an unexpected increase in the execution time of Job(N-1) [17]

event occurs close to a time-critical deadline, the reaction time will be too long to provide full guarantees, as illustrated in Fig 1. Therefore, the smallest job length is constrained by the switching time. Practically, we can assume the switching time should be less than 1% of the job execution time. In this case, the job execution time should be several milliseconds at least, which becomes the limit of mitigation with DVFS. For shorter jobs, DVFS is not feasible anymore, and knobs with a shorter switching time must be adopted. Recently, we proposed the *heterogeneous datapath* (HDP) [17] as a promising fast-switching knob to mitigate for jobs at sub-millisecond level. To extend the timescale view to the fullest, one can envisage a multi-timescale mitigation methodology using knobs of different timescales as a total solution to all kinds of performance variability.

In this work, we propose a multi-timescale mitigation methodology which improves the performance of mitigation in both timing guarantee and energy saving. This methodology has two features: first, it exploits the concept of *dynamic scenarios* (DS) [8] and instantiates that towards a practical and effective approach which is applicable in our strict real-time scheduling context. This approach can derive a tight upper bound of execution time at run-time, so it improves the way of exploiting time resource for energy saving. Second, we propose an algorithm for deciding the modes for multiple knobs of different timescales. We demonstrate the methodology with the DVFS and the HDP knobs. Nevertheless, the principles of the methodology can be extended to any combination of knobs of different timescales. This work is an extension of our previous work of [17]. In our previous work, we proposed the HDP knob and elaborated how it can be used in run-time mitigation. Then in this work, we further discuss the other essential aspects in the realization of the overall mitigation methodology, including the mode scheduling algorithm and the multi-timescale concept which enables multiple knobs to work together.

This paper is structured as follows. In Chapter II, the proposed approach of dynamic scenarios with is elaborated with a single knob. Then, in Chapter III, the approach is further extended to the multi-timescale methodology using multiple knobs. Chapter IV provides the detailed simulation setup of our experiments on the proposed methodology. After that, Chapter V discuss the results of the experiments and Chapter VI is our conclusion.

II. Dynamic Scenario Based Approach

The dynamic scenario based approach is a workload-dependent mode scheduling approach which aims to proactively respond to the run-time performance variability to ensure timing guarantee. Its principle is to refine the execution time calculation based on the situation observed at run-time. In this way, the prediction can be less pessimistic than the WCET, but still gives an upper bound of the real execution time.

The conceptual framework of this approach is introduced in [8], in which the theoretical basis is established on a generalized mathematical model of real-time systems. Here, we instantiate those principles for mitigating timing variations, in a novel context with a new approach tuned to our specific objectives. Our approach is built upon the conceptual framework and realized for the practical use in the processors equipped with speed-varying knobs like HDP and DVFS. The methodology contains two parts, the design-time process and the run-time process. These two processes are explained in the following sections.

A. Design-time process

Fig. 2 shows the design-time process of the proposed dynamic scenario based approach. In the design-time process, first the target application is profiled. Based on the analysis of the profiling traces of execution time and energy, the application is segmented into atomic units called *thread nodes* (TN). Each TN is assigned a cycle budget, which represents the average number of cycles each TN required. These cycle budgets will be used as the references for scheduling at run-time. The TNs should be the basic units of workload execution at run-time, so their sizes must match the finest timescale of the knobs. For example, if HDP is our finest knob, which enables mitigation down to the job size of the order of tens of microseconds, then the smallest size of a TN can be in the size of tens of microseconds.

Then, the process of scenario set generation is performed, which is illustrated in Fig. 2. In this process, the TNs are classified into *run-time situations* (RTS) based on their characteristics. For example, the TNs which perform different iterations of the same loop can be classified into the same RTS. Then, the RTSs are further clustered into *dynamic scenarios*. Two main factors are taken into consideration during the clustering process. First, the clustering of scenarios is based on the run-time parameters, such as program variables and metadata of the input files. Different scenarios are positioned in different regions of the parameter space. Therefore, the clustering can be treated as the segmentation of the parameter

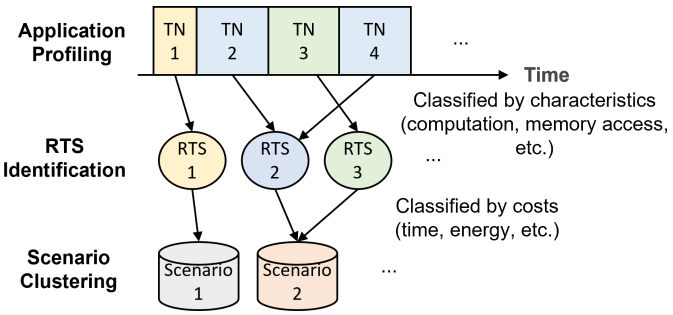

Fig. 2. Process of generating a scenario set in the design-time process of the dynamic scenario based approach

space into regions, each of which represents a scenario. Hence, at run-time the active scenario can be easily determined by reading the values of the run-time parameters. Second, each scenario should contain RTSs with similar costs of execution time and energy, so that when predicting the costs of the RTS by its scenario, the error will be minimal. In this work, the scenarios are clustered manually by analyzing the program flow. For applications of high complexity or unknown program flows, automatic clustering algorithms can be developed on the same principles to perform this task systematically [11].

After the clustering process, statistics (the worst RTS cost, average RTS cost, etc.) are recorded for each scenario to be used at run-time for the likely case prediction and the refined upper bound calculation of the execution time. These statistics can be easily stored in a look-up table. Here, we record both the worst number of cycles and the average number of cycles. At the same time, the scenario detector is developed, which is used at run-time to detect the scenarios of the upcoming TNs the processor is going to run. The design of the detector is merely querying the look-up table by the run-time parameters.

B. Run-time process

Fig. 3 show the block diagram of the run-time process of the dynamic scenario based approach. At run-time, the controller decides which mode to run at the start of every TN. The controller is supposed to run in the optimized firmware or in a standalone power management unit, so that the time overhead is minimized.

The overall procedure of the run-time process is shown in Algorithm 1. At the start of the workload, an *optimal schedule* generated by scheduling with the average number of cycles of each TN and the hard deadlines of the workload. This schedule is the presumed ideal schedule which can ensure timing guarantee and minimize energy consumption if there is no variability. The goal of the algorithm is to make the progress of the workload to be always ahead of the optimal schedule, but as close as the optimal schedule as possible. Then, the controller defines a deadline for each TN, which is equal to the TN finish time in the optimal schedule. These deadlines serve as checkpoints of the progress of the workload. In addition, the controller creates a buffer to keep track of the scenarios of a number of up-coming TNs, so the controller can look into the buffer to evaluate the future workload.

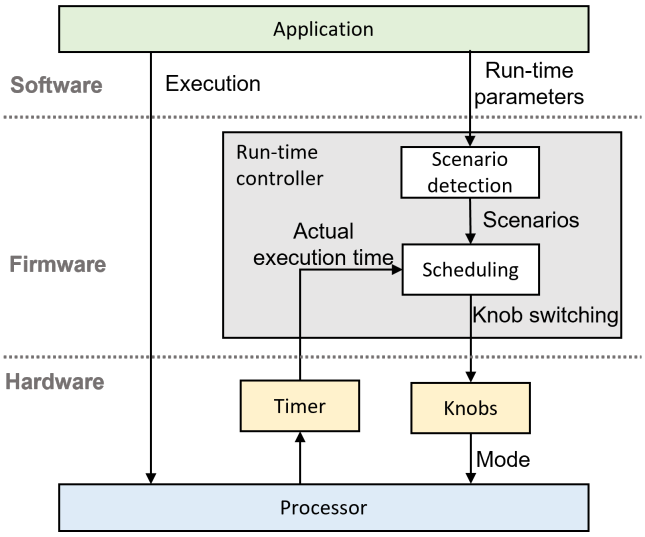

Fig. 3. Block diagram of the run-time process of the dynamic scenario based approach

Algorithm 1 Run-time process

Input: thread nodes $TN_{1...N}$, buffer size B
Output: modes which each TN uses $m_{1...N}$
$CB_{1...N} \leftarrow$ cycle budgets of $TN_{1...N}$
$DL_{i...N} \leftarrow OptimalSchedule(CB_{1...N})$
Initialize an empty buffer buf
for $i \leftarrow 1$ to B **do**
    $S_i \leftarrow ScenarioDetect(TN_i)$
    Push S_i to buf
end for
for $i \leftarrow 1$ to N **do**
    $f_{mr} \leftarrow RefinedUpperBound(buf)$
    $f_{opt} \leftarrow LikelyPrediction(buf)$
    $m_i \leftarrow$ the nearest faster mode of $max(f_{mr}, f_{opt})$
    Execute TN_i under mode m_i
    Pop S_i from buf
    if $i + B \leq N$ **then**
        $S_{i+B} \leftarrow ScenarioDetect(TN_{i+B})$
        Push S_{i+B} to buf
    end if
end for

At the start of the execution of each TN, the controller calls the scenario detector to identify the scenario of the last TN in the buffer. Then, two processes are performed to decide the mode of the knob to run the next TN. First, the controller calculates the timing guarantee criteria by a *refined upper bound* process, shown in Algorithm 2. The timing guarantee criteria are expressed by the minimum required speed for the next TN, which is the slowest speed to run the next TN while still ensuring timing guarantee. To calculate this, the controller generates a schedule which could create the greatest possible slack. This means that the controller assumes the other scenarios in the buffer to be their worst RTS execution time and run at the fastest mode. Only when the slack is available beyond what is required to meet the strictest deadline,

Algorithm 2 Refined Upper Bound

1: **procedure** REFINEDUPPERBOUND(buf)
2: $T_{cur} \leftarrow$ current time
3: $f_{max} \leftarrow$ maximum frequency
4: $B \leftarrow$ number of TNs in buf
5: $DL_{1...B} \leftarrow$ deadlines of TNs in buf
6: $WC_{1...B} \leftarrow$ worst cycles of scenarios of TNs in buf
7: $PDL_B \leftarrow DL_B$
8: **for** $i \leftarrow B$ **downto** 1 **do**
9: $PDL_{i-1} \leftarrow min(PDL_i - WC_i/f_{max}, DL_{i-1})$
10: **end for**
11: $f_{mr} \leftarrow WC_1/(PDL_1 - T_{cur})$
12: **return** f_{mr}
13: **end procedure**

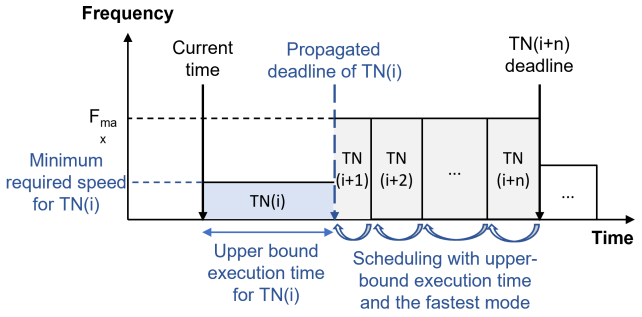

Fig. 4. Refined upper bound process to calculate the minimum required speed for the next TN for ensuring timing guarantee

Algorithm 3 Likely Prediction

1: **procedure** LIKELYPREDICTION(buf)
2: $T_{cur} \leftarrow$ current time
3: $B \leftarrow$ number of TNs in buf
4: $DL_{1...B} \leftarrow$ deadlines of TNs in buf
5: $AC_{1...B} \leftarrow$ average cycles of scenarios of TNs in buf
6: $f_{opt} \leftarrow sum(AC_{1...B})/(DL_B - T_{cur})$
7: **return** f_{opt}
8: **end procedure**

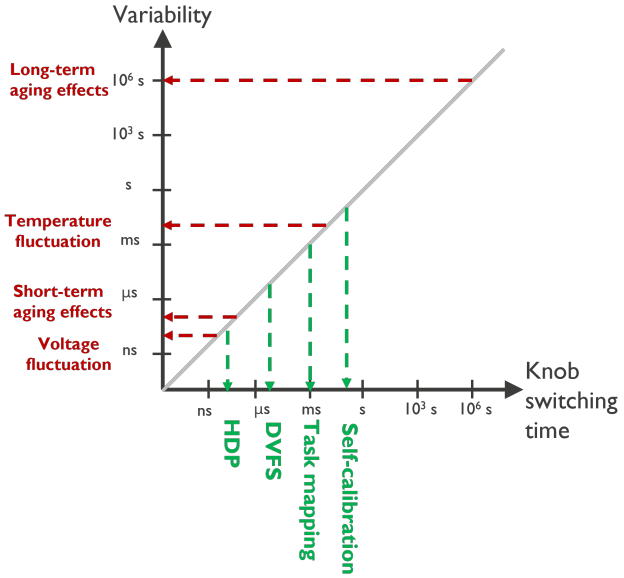

Fig. 5. Variability sources and mitigation knobs arranged by the timescales, which tells whether the knobs can react faster than the variability for successfully mitigation.

then that slack can be used for the next TN. Therefore, this process is performed for each deadline of the TNs in the buffer, and in the end only the strictest speed constraint is considered. Fig. 4 gives a visual illustration of the principle of this process.

Second, *likely prediction* is performed to calculate the optimal speed, which is the processor speed that is most likely to minimize the total energy consumption for the TNs in the buffer. This process is shown in Algorithm 3. In the beginning of the process, it would predict the real execution time of the TNs in the buffer, based on the statistics of the scenarios acquired in the design-time process. Heuristics can be applied to improve the accuracy of the prediction. In this work, we predict the TN execution time to be the average execution time of the scenario. Then, an optimal scheduling process is performed, in which the scheduler considers the execution time prediction to generate a schedule that minimizes the total energy consumption for the TNs in the buffer, and thus the mode to run the next TN is decided. Solving the optimal schedule at run-time will not be feasible since it takes too much computation, so heuristics need to be applied. In this work, our heuristics are designed based on the concept that it is near-optimal to run constantly at the slowest possible speed. Therefore, the controller calculates the optimal speed which is just enough to run all TNs to meet the deadline of the last TN in the buffer. Finally, the controller finds the slowest mode that is faster than both the optimal speed and the minimum required speed, as shown in Line 12 of Algorithm 1. This becomes the selected mode to run the next TN.

III. Multi-Timescale Mitigation Methodology

Different variability sources have different characteristics in terms of timescales. For example, an occurrence of voltage droop can happen in tens of nanoseconds, while the variability of temperature fluctuation normally accumulates over hundreds of microseconds. Likewise, knobs are different in their switching times. For example, the switching time of DVFS is normally several microseconds to tens of microseconds [6], while the self-calibration process normally takes hundreds of milliseconds. Fig. 5 shows a holistic view on the timescales of variability sources and mitigation knobs. The timescales of the knobs make them suitable for tackling different type of variability sources. To successfully mitigate for a variability source, knobs with a switching time faster than the variability source must be adopted. Otherwise, the knobs will not be able to react to the variability in time before the timing guarantee is violated. Therefore, a total solution for all kinds of performance variability should adopt multiple knobs of different switching times, in which each kind of variability is dealt with by the knobs of its respective timescale.

Here, we will demonstrate the concept of the multi-timescale mitigation methodology with a realization using HDP and DVFS. For these two knobs, there is a trade-off

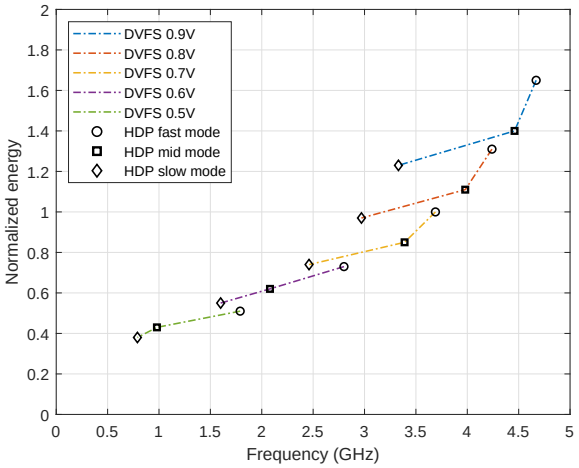

Fig. 6. Speed and energy trade-off of the ALUs running in each of the possible combinations of the DVFS mode (0.5V-0.9V) and the HDP mode (fast, mid, and slow)

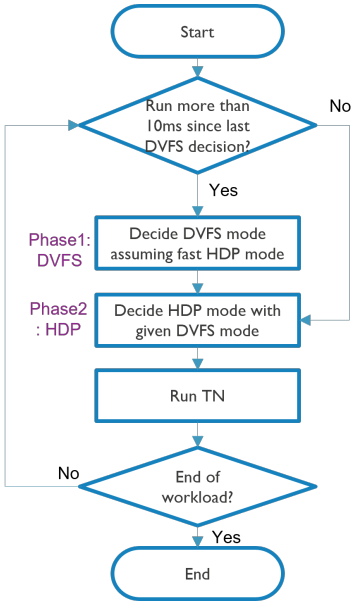

Fig. 7. Flow diagram of the run-time process of the multi-timescale DS based approach, in which the modes of knobs are decided sequentially from the fast ones to the slow ones

between their reaction speed and impact ranges. An example of the speed and energy of the DVFS and HDP modes is shown in Fig. 6, which comes from our experiment of benchmarking a JPEG-encoding workload on our HDP design of the ALU of the Ariane core [9] with the simulation flow in Chapter IV. DVFS enables a wide range of speed and energy variation, but its switching time is longer, so it is suitable to react to the large and slow variability. On the other hand, HDP can react in a very short time and provide fine-grained modes to fine-tune the running speed, but its speed range and energy variation is limited, so it is suitable to response proactively to mitigate the small and sudden variability. In this methodology, HDP and DVFS work in a complimentary way, so the advantages of both knobs can be exploited.

Regarding the design of the knobs, the speed-energy trade-off is not the only factor to be considered, since the fast switching time could allow the mode with lower energy efficiency to have more energy saving in some situations. To realize the full potential of the mitigation, the HDP design should make the most of the energy-speed curve [20]. This means that at the high-frequency end the HDP should push the frequency as high as possible before the slope of the curve becomes so steep that the additional energy investment is not worth it any longer. Also, at the low-frequency end the HDP should push the frequency as low as possible before the energy is flattened. In this way, the processor can have a full range of modes to tackle different kinds of situations.

The flow diagram of the run-time process of the multi-timescale mitigation methodology is shown in Fig. 7. The algorithm sequentially selects the modes of the knobs from the slow-switching ones to the fast-switching ones. In this example, the first time is to select the DVFS mode, and the second time is to select the HDP mode. The switching rates of HDP and DVFS are different. HDP can switch at the start of each TN. On the other hand, DVFS can only switch once over multiple TNs. Otherwise, its switching time overhead would greatly lengthen the execution time. Here, we limit the interval between two DVFS switches to be larger than a threshold value $T_{th,DVFS}$. That is to say, at the start of each TN, the controller will check whether the time since the last time the controller decided the DVFS mode is more than $T_{th,DVFS}$, and it would only consider switching the DVFS knob if the answer is yes. Here, the role of DVFS is to react to the large slack accumulated over a long time span which HDP is not sufficient to exploit due to its limit impact range. The choice of $T_{th,DVFS}$ is a balance between the granularity of mitigation and the overhead on execution time. If $T_{th,DVFS}$ is too long, the DVFS might not be able to react to the variability faster than $T_{th,DVFS}$. On the other hand, if $T_{th,DVFS}$ is too short, the DVFS might switch too frequently so that the switching time significantly lengthen the execution time. In our methodology, we set $T_{th,DVFS}$ to 10 ms, so that the DVFS switching time overhead is less than 1% of the execution time. The principle of our methodology can be extended to any knobs of any timescales.

In principle, the process of deciding modes of multiple knobs is compatible to any scheduling algorithms. However, the dynamic scenario based approach can calculate the bound of execution time in a finer timescale, so it can exploit the benefits of multiple knobs much better than other algorithms. Here, we instantiate the multi-timescale mitigation methodology with an enhanced version of the dynamic scenario based approach. Algorithm 4 shows the detailed flow of the run-time process of the enhanced approach. In Line 10, the controller would check whether the time since last DVFS decision point is longer than $T_{th,DVFS}$ to decide whether to select a new DVFS mode.

In Line 11, the controller calls the multi-timescale version of the refined upper bound process. The detail of the process is shown in Algorithm 5. A major change is that the controller would consider that the DVFS mode has to run over multiple

Algorithm 4 Run-time Process of Multi-Timescale Mitigation

1: **Input**: thread nodes $TN_{1...N}$, buffer size B
Output: DVFS & HDP modes which each TN uses $m_{DVFS,1...N}$, $m_{HDP,1...N}$
2: $CB_{1...N} \leftarrow$ cycle budgets of $TN_{1...N}$
3: $DL_{i...N} \leftarrow OptimalSchedule(CB_{1...N})$
4: Initialize an empty buffer buf
5: **for** $i \leftarrow 1$ to B **do**
6: $S_i \leftarrow ScenarioDetect(TN_i)$
7: Push S_i to buf
8: **end for**
9: **for** $i \leftarrow 1$ to N **do**
10: $DVFS\_triggered \leftarrow CheckDvfsTriggered()$
11: $f_{mr} \leftarrow RefinedUpperBound\_MT(buf)$
12: $f_{opt} \leftarrow LikelyPrediction\_MT(buf, DVFS\_triggered)$
13: $f_{res} \leftarrow max(f_{mr}, f_{opt})$
14: **if** $DVFS\_triggered$ **or** $i = 1$ **then**
15: $m_{DVFS,i} \leftarrow$ the nearest faster DVFS mode of f_{res} assuming using fastest HDP mode
16: **else**
17: $m_{DVFS,i} \leftarrow m_{DVFS,i-1}$
18: **end if**
19: $m_{HDP,i} \leftarrow$ the nearest faster HDP mode of f_{res} assuming using $m_{DVFS,i}$
20: Execute TN_i under mode $m_{DVFS,i}, m_{HDP,i}$
21: Pop S_i from buf
22: **if** $i + B \leq N$ **then**
23: $S_{i+B} \leftarrow ScenarioDetect(TN_{i+B})$
24: Push S_{i+B} to buf
25: **end if**
26: **end for**

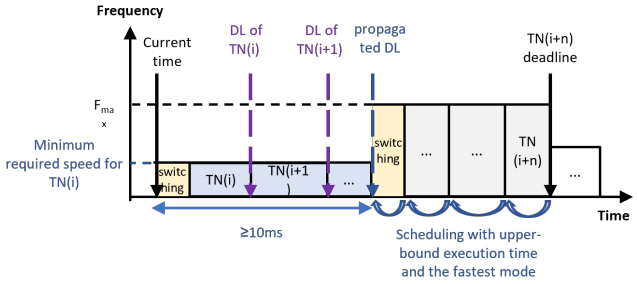

Fig. 8. Enhanced refined upper bound process to calculate the minimum required speed for the next TN for ensuring timing guarantee in multi-timescale mitigation methodology

TNs until the next DVFS switch, as illustrated in Fig. 8. Therefore, the worst-case situation is that the processor runs in a certain DVFS mode for $T_{th,DVFS}$, and then switches to the fastest DVFS mode until the end of the buffer. A speed requirement is derived f_{mr} based on this case, as shown in Line 15-19. In addition, the controller also derives the speed requirements to meet all deadlines before the next DVFS switch. Based on all of these requirements, the controller derives the lowest speed which can ensure all timing guarantees, as shown in Line 21-23.

Algorithm 5 Refined Upper Bound of Multi-Timescale Mitigation

1: **procedure** REFINEDUPPERBOUND_MT(buf)
2: $T_{cur} \leftarrow$ current time
3: $f_{max} \leftarrow$ maximum frequency
4: $N \leftarrow$ number of TNs in buf
5: $DL_{1...B} \leftarrow$ deadlines of TNs in buf
6: $WC_{1...B} \leftarrow$ worst cycles of scenarios of TNs in buf
7: $T_{sw,DVFS} \leftarrow$ DVFS switching time
8: $T_{th,DVFS} \leftarrow$ period between two DVFS switching
9: *// Propagate deadlines*
10: $PDL_B \leftarrow DL_B$
11: **for** $i \leftarrow B$ **downto** 1 **do**
12: $PDL_{i-1} \leftarrow min(PDL_i - WC_i/f_{max}, DL_{i-1})$
13: **end for**
14: *// Find the last TN of DVFS interval*
15: $j \leftarrow 1$
16: **while** $PDL_j < T_{cur} + T_{th,DVFS} + 2T_{sw,DVFS}$ **do**
17: $j \leftarrow j + 1$
18: **end while**
19: $f_{mr} \leftarrow sum(WC_{1...j})/(PDL_j - T_{cur} - 2T_{sw,DVFS})$
20: *// Check deadlines in this DVFS interval*
21: **for** $k \leftarrow 1$ to j **do**
22: $f_{mr} \leftarrow max(f_{mr}, sum(WC_{1...k})/(DL_k - T_{cur} - T_{sw,DVFS}))$
23: **end for**
24: **return** f_{mr}
25: **end procedure**

In Line 12, the controller calls the multi-timescale version of the likely prediction process, which takes DVFS switching time into consideration in calculation. The detail of the process is shown in Algorithm 6. This enhanced version takes the DVFS switching time into consideration. Finally, in Line 14-18, the controller would only decide the new DVFS mode when DVFS is triggered. Otherwise, the current DVFS mode will be used for the next TN. To ensure the timing guarantees, when deciding the new DVFS mode, the controller assumes the fastest HDP mode is used. Once the DVFS mode is decided, the new HDP mode will be decided based on the condition of the selected DVFS mode.

IV. Simulation Setup

Fig. 9 shows the simulation flow of the proposed methodology. The methodology is simulated on the design of the RISC-V Ariane core [9], which is a modern representative 64-bit processor core. It is a building block of OpenPiton [18], a RISC-V multi-core processor platform which is available through open access.

A. Processor implementation and simulation

We equip the Ariane core with both the DVFS and HDP knobs. The design of the HDP knob is the same on in [17],

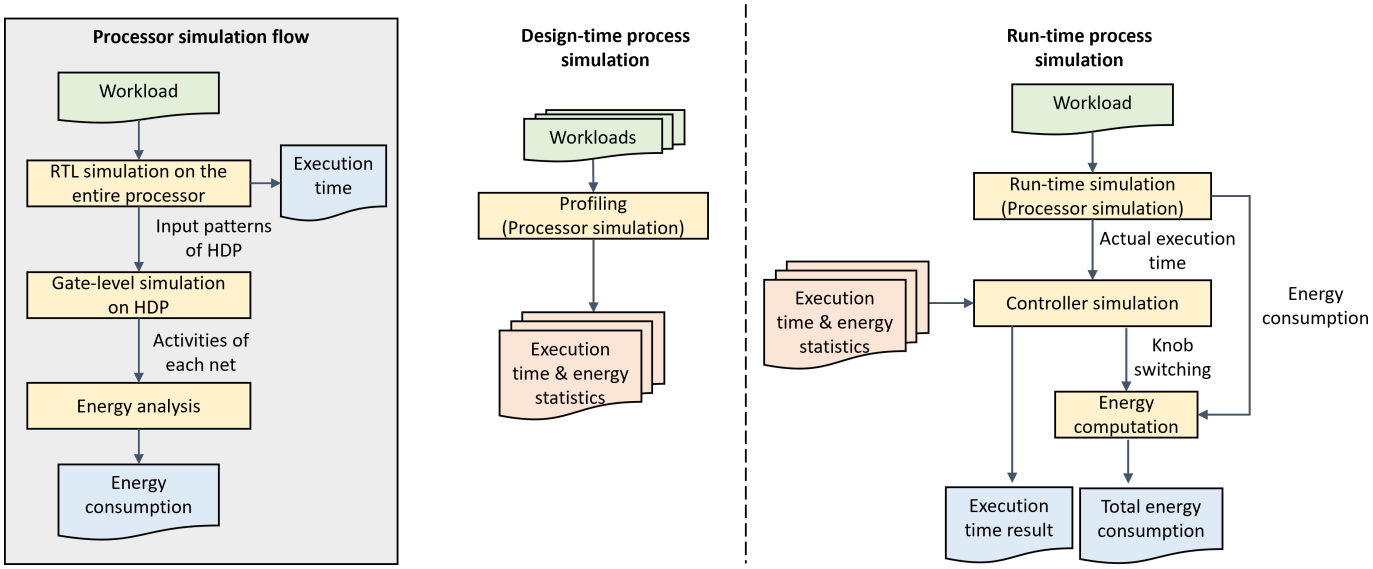

Fig. 9. The cycle-accurate simulation flow to acquire the execution time and energy consumption when running a workload under mitigation [17]

Algorithm 6 Likely Prediction of Multi-Timescale Mitigation

procedure LIKELYPREDICTION_MT(buf, $DVFS\_triggered$)
  $T_{cur} \leftarrow$ current time
  $B \leftarrow$ number of TNs in buf
  $DL_{1...B} \leftarrow$ deadlines of TNs in buf
  $AC_{1...B} \leftarrow$ average cycles of scenarios of TNs in buf
  $T_{sw,DVFS} \leftarrow$ DVFS switching time
  if $DVFS\_triggered$ **then**
    $f_{opt} \leftarrow sum(AC_{1...B})/(DL_B - T_{cur} - T_{sw,DVFS})$
  else
    $f_{opt} \leftarrow sum(AC_{1...B})/(DL_B - T_{cur})$
  end if
  return f_{opt}
end procedure

which is implemented on the ALU unit by extending its design to three units. The ALU has gone through the full physical design process so that the accurate timing and energy can be acquired in simulation. The summary of the HDP implementation in shown in Table I. In our simulation, the time and dynamic energy overheads of HDP switching are not incorporated, since in the previous work it showed that the HDP switching time is only tens of cycles [17], which is only about 0.1% of execution time in our sub-millisecond mitigation context, so it would only have negligible influence on time and energy. Moreover, the energy overheads from the idle ALUs are also very small compared to the energy of the active ALU, thanks to the application of microarchitectural power-gating technique [19]. Even through the duration of power-gating can be as short as tens of microseconds, it is still thousands of times longer than the break-even point, which is approximately ten cycles [19]. Therefore, the energy consumptions of the idle ALUs are also left out of the analysis.

TABLE I
SUMMARY OF THE ALU DESIGNS IN HDP [17]

Design	Fast ALU	Mid ALU	Slow ALU
Clock frequency	3.69 GHz	3.39 GHz	2.46 GHz
Number of gates	18099	18092	12827
Multi-Vth libraries used	LVT, SVT	SVT, HVT	SVT, HVT
Technology	3nm [15]		
Operating condition	TT/0.7V/25C		

The DVFS knob is realized by a timing and energy calculation model. We create five DVFS modes, with voltages ranging from 0.5V to 0.9V. The operating frequency of each mode is simulated by static timing analysis on the ALU design, while the energy consumption of each mode is calculated by scaling from the energy consumption of the 0.7V mode, based on the simple dynamic power consumption model as follows.

$$P = \alpha C V^2 f \tag{1}$$

where α is the activity factor, C is the equivalent capacitance, V is the voltage, f is the clock frequency respectively. We assume that the number of cycles of a workload is constant regardless of the DVFS mode, so the energy consumption can be approximated as follows.

$$E = N_{cycle} \alpha C V^2 \tag{2}$$

Therefore, the scaling of energy consumption among the DVFS modes is proportional to V^2. Here, we do not consider static energy, since in simulation we found it much smaller than the dynamic energy due to all the leakage reduction techniques (high-V_{th} library, power gating). A summary of the DVFS model is shown in Table II, and the speed-energy relations of all available DVFS and HDP modes are shown in Fig. 6. In addition, we assume that the switching time of DVFS is a constant of 100 μs regardless of the modes which the processor switches from and to. This switching time is at the same level of the empirical numbers measured from modern processors [6].

TABLE II
SUMMARY OF DVFS MODEL

Voltage (V)	Frequency (GHz)			Energy scaling factor ($\sim V^2$)
	Fast HDP	Mid HDP	Slow HDP	
0.9	4.67	4.46	3.33	1.65
0.8	4.24	3.98	2.97	1.31
0.7	3.69	3.39	2.46	1.00
0.6	2.80	2.08	1.60	0.73
0.5	1.79	0.98	0.79	0.51

To accurately simulate the execution time and the energy consumption of the core, we have developed a cycle-accurate processor simulation flow [17]. First, we ran RTL-level simulation on the RTL design of the core, running the target workload. The input patterns of the ALU are extracted during the simulation, and the execution time results are acquired at the same time. Then, A gate-level simulation is performed on the netlist of each ALU. The input patterns from the RTL simulation are applied to the design in the simulation to acquire the activities of each net in the ALUs. Afterwards, these activities are used by the commercial power analysis tool to compute the energy consumption of the ALUs.

B. Simulation of the multi-timescale mitigation methodology

During the design-time process simulation, first we profile the application by the processor simulation with different input files, to acquire the statistics of execution time and energy. Then, based on its program flow, the application is segmented into TNs of execution time of hundreds of microseconds. Once the execution time and energy of all TNs are acquired, we analyze these data to search for suitable run-time parameters for scenario clustering. The statistics of each scenario are calculated and stored in a look-up table. At run-time, this table is read by the controller for timing guarantee criteria calculation and TN cost prediction.

During the run-time process simulation, first we perform the processor simulation running target workload with the target inputs to acquire their execution time and energy. These results are read by a controller simulator which, combined with all the data provided by the design-time process, derives the knob-switching decisions the controller makes during the whole process of the workload. The results of these knob-switching decisions are combined with the energy analysis results to calculate the total energy consumption under run-time mitigation.

V. EXPERIMENTAL RESULTS

The applications we target for are the ones the highly dynamic workload with strict real-time requirements. To evaluate the effectiveness of our methodology in mitigation for this kind of applications, we select two representative workloads in the experiments. The first one is the lower sub-band quantization block of the ADPCM-encoding application of TACLeBench [16]. The second one is the JPEG-decoding application of MiBench [10]. Both workloads represent typical real-time signal and data processing applications, where the input of workload is a sequence of frames, and the deadlines for processing these frames are periodic, called frame period. In conventional real-time scheduling, a job is defined as processing a single frame, but in our methodology, more fine-grained mitigation can be achieved by cutting the job of processing a frame into multiple TNs, each of which has an execution time of tens to hundreds of microseconds. The execution time of these workloads is highly dispersed with different input data. Therefore, they are representative of the highly dynamic workload that our methodology targets for.

In the experiment, we compare six methodologies of different algorithms and knobs. In terms of the algorithms, we select three different configurations, each is representative of an algorithm type: the first type is the best-effort (BE) algorithms, represented by the algorithm of Pering et al [21]. The controller of this type always greedily predicts the execution time of processing one frame regardless of the content of the frame. In Pering's algorithm [21], the prediction is the exponential moving average of the executed cycles of the past frames. This way, the prediction will be close to the real execution time most of the time, so the controller can aggressively reduce energy. However, this algorithm cannot ensure timing guarantee, so it is not a viable algorithm for mitigation for time-critical applications. Here, it only serves as a reference of the best-case energy consumption. The second type is the WCET-based algorithms, represented by the cycle-conserving algorithm of Pillai et al [7]. In this type, the prediction of the execution time of processing a frame is always the WCET regardless of the content of the frame. This configuration represents the conventional algorithm of real-time scheduling. The third type is the dynamic scenario (DS) based approach we proposed.

In terms of the knobs, we have two different configurations. The first one is the single-DVFS configuration, which only switches the DVFS knob, and the HDP knob always stays in the fast mode. This configuration represents the conventional mitigation method which only has the DVFS knob and one datapath designed for high frequency. The best-effort and the WCET algorithms of this configuration can switch DVFS modes only in the beginning of a frame, since they are ignorant of the TNs, while the DS algorithm can switch at the start of a TN. The second one is the multi-timescale configuration, which uses both the DVFS and the HDP knob. This configuration represents the proposed multi-timescale mitigation methodology. For the best-effort and the WCET-based algorithms, their multi-timescale versions are implemented based on the flow in Fig. 7. Therefore, they can switch the knobs at the start of a TN. Still, their execution time predictions are frame-based and are constant in spite of the run-time situation, so they cannot fully enjoy the advantages of a finer timescale. Combining different configurations of the knobs and the algorithms, we have six different methodologies for comparison. By comparing results of different configurations, we can decouple the impacts from two different features of our methodology: either from the knobs or from the algorithms.

A. Case I: Quantization of ADPCM Encoding

Here, we assume an application of the lower sub-band quantization block of ADPCM-encoding, where frames of

TABLE III
SCENARIO LOOK-UP TABLE FOR THE ADPCM QUANTIZATION APPLICATION

Scenario	Volume	Avg cycle count	Max cycle count
1	0	42276	42276
2	1 - 10^4	56180	76640
3	10^4 - 10^5	100451	124669
4	10^5 - 4×10^6	147584	169068
5	4×10^6 - 7×10^6	164788	206888
6	$>7\times10^6$	202253	265514

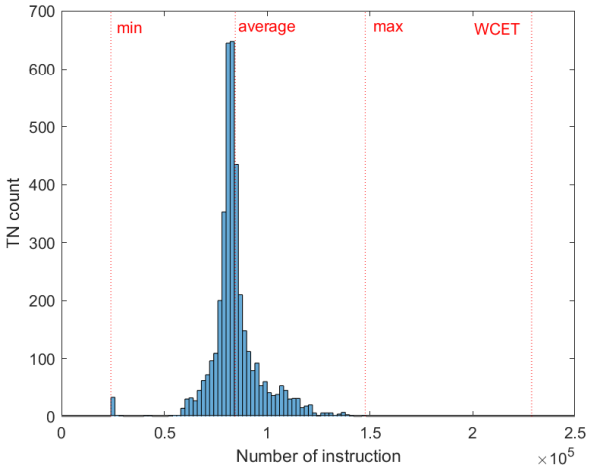

Fig. 10. Distribution of the instruction count of the TNs of the ADPCM quantization workload, in which the great difference between WCET and realistic cases are shown

audio samples are sequentially fed as the inputs, and frames of quantization levels come out as the outputs. Though the original application includes the entire ADPCM encoding flow, we only experiment on the quantization block of the low sub-band, which is the most dynamic part in the flow, to maximize the dynamism of the workload. It is not unrealistic to run the quantization block alone, since one can easily modify the ADPCM encoding application to parallelize the quantization part from the other parts. Technically, we simulate the entire ADPCM encoding application, but only record the execution time and energy consumption of the lower sub-band quantization part of the application. The execution time and energy consumption of different frames are summed up to represent the case that the quantization part is run alone. The inputs of the application are audios of 22050 Hz sample rate. The frame size of the audio stream is 2000 samples. We divide the workload in a way that each TN represents processing of 100 samples, which means that one frame is composed of 20 TNs.

In the design-time process, we profile six audio files of 10-second long. Fig. 10 shows the distribution of the number of instructions of all profiled TNs. From the distribution, we can observe that the workload is highly dynamic. It also clearly shows that the number of instructions of the WCET case derived from program flow analysis is much greater than the maximum number of instructions we observe. Therefore, it proves our point that WCET is too pessimistic to represent the real execution time. In scenario clustering, we use the volume as the run-time parameter. The volume is defined as the sum of the absolute values of all samples in the TN. Six scenarios are created with these profiling results of 1000 TNs, whose inputs are composed by an audio stream of 100 TNs repeated by ten times. The total execution time of this workload is 34 ms under the 0.7V DVFS mode and the fast HDP mode.

Fig. 11 shows the selected DVFS and HDP modes during the entire run of the 1.0-ms frame period case. And Fig. 12 shows the traces of slack of all mitigation methods. The slack is defined as the amount of time by which a TN ends before its deadline. The optimal behavior of the slack trace is to be always positive and as close to zero as possible, which means that the processor runs in the slowest possible speed without any deadline violations. For the BE methods, the slack is close to zero all along the workload, which is near-optimal for energy saving. However, as shown in Fig. 12, their slacks can sometimes be negative, which means potential deadline violations. Therefore, these methods cannot ensure timing guarantee.

For the methods that can ensure timing guarantee (WCET and DS), we can identify three phases of mitigation over the entire workload from the traces of slack. First, in the beginning of the workload, fast modes are used to accumulate slack to ensure a safe margin to avoid deadline violations. Then, when sufficient slack is secured to ensure timing guarantee, the processor run at a speed in which the slack keeps at a stable level. In this phase, the processor can switch among neighboring modes to fine-tune the running speed or react to small variability. Finally, when the workload is about to finish, the processor would exploit the slack to save energy by running in slow modes.

While all four curves generally follow the three phases of mitigation, their behaviors are quite different. Comparing the traces of different knob configurations, we observe that the traces of the single-DVFS configuration tend to have sharp turns, while the traces of the multi-timescale configuration are much smoother. This shows that the fine-grained modes and the small timescale enabled by HDP can make mitigation performed in a smoother way. Furthermore, by comparing the traces of the two DS methods, it also shows that the multi-timescale method requires less slack to ensure timing guarantee, which results in less time and slower modes in the first phase. Also, the multi-timescale method starts the third phase earlier, so it can exploit more slack by the end of the workload. These behaviors show the benefits of the multi-timescale mitigation in providing more energy saving.

In terms of the algorithms, the benefit of the dynamic scenarios is shown Fig. 11a and Fig. 12 by comparing the results of the two methods with the single-DVFS configuration, in which the dynamic scenario based approach can start the second phase much earlier than the WCET-based method. This shows that the dynamic scenario based approach requires less slack to ensure timing guarantee due to the tight upper bound execution time calculation. The smaller slack indicates that the processor can run at a slower speed, which is certainly beneficial to energy saving. Finally, combining the benefits of both features, the proposed methodology has a smooth

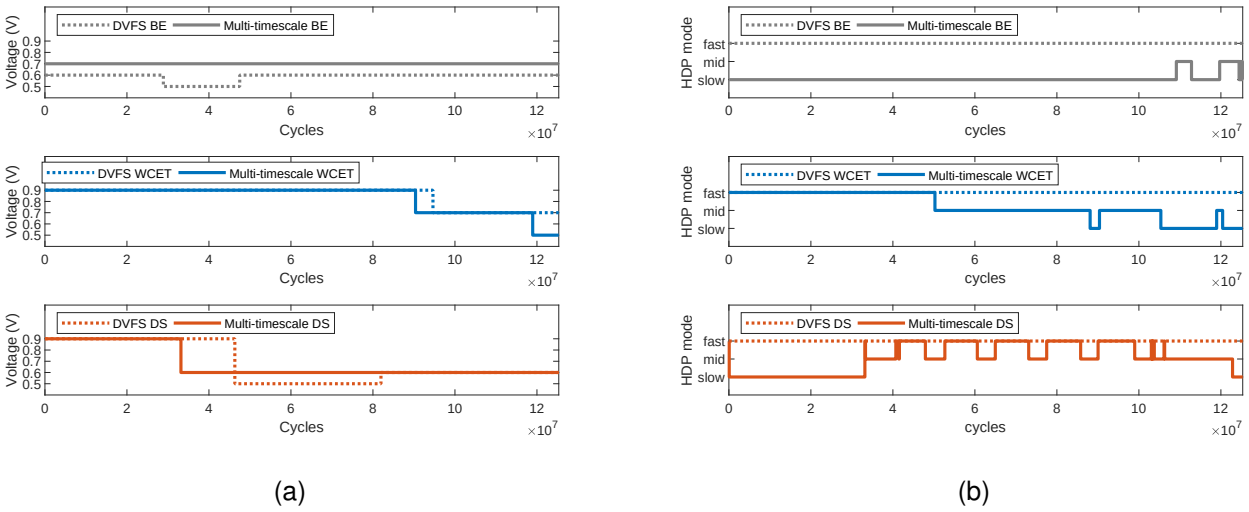

Fig. 11. Selected modes over the entire run of the ADPCM quantization workload with the 1.0-ms frame period. (a) DVFS mode (b) HDP mode

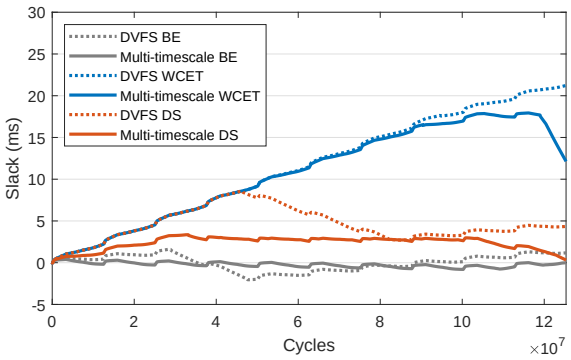

Fig. 12. Trace of slack over the entire run of the ADPCM quantization workload with the 1.0-ms frame period

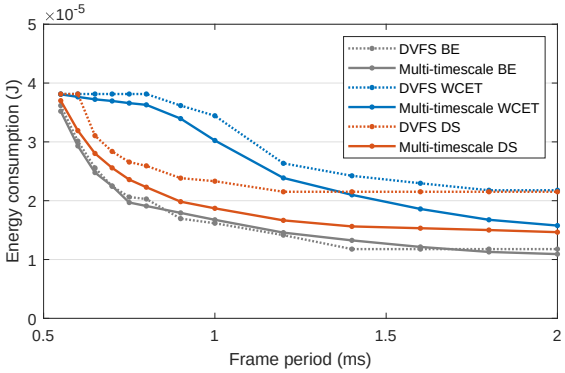

Fig. 13. ADPCM quantization workload energy consumption of different methods under different time-criticality conditions

mitigation process over the whole workload with only the least amount of slack required to be secured. Its curve is always positive and close to zero, which is near to the optimal behaviors we described above. This means it is the best method among all in keeping the processor in the slowest possible speed without violating any deadlines.

Fig. 13 shows the energy consumption of different methods under different time-criticality situations, represented by different frame period. The energy differences among all methods are more prominent in the middle of the curves. At the left end of the curves, the timing is so strict that all methods use the fastest mode most of the time, while at the right end of the curves, the timing is so relaxed that all methods use the slowest mode most of the time. In these situations the choice of the mitigation method is less relevant.

Comparing different methods, Fig. 13 shows that the proposed multi-timescale DS methodology has remarkable energy saving of 32% in average and 46% in maximum, compared to the conventional WCET-based single-DVFS method. Also, the benefit is significant in all time-criticality situations. As compared to the single-DVFS best-effort method, which are near-optimal for energy saving, the energy consumption of the multi-timescale DS methodology is larger by only 17% in average and 33% in maximum. Regarding the timing guarantee, the single-DVFS best-effort method misses 13% deadlines in average, while the multi-timescale DS methodology has zero deadline misses. Since the best-effort method can cause potential deadline misses, it is not an available solution for time-critical applications. For these applications, the multi-timescale DS methodology is the best solution, in which a small energy overhead is a reasonable price to pay to ensure the timing guarantee.

The separate contributions of each feature can also be derived from Fig. 13. Regarding the algorithm, the benefit of the DS based algorithm can be evaluated from the comparison between the DVFS WCET method and the DVFS DS method. It shows that the algorithm improves the energy consumption by 17% in average. Its benefit is more prominent in the time-critical region, since its advantage of deriving a tighter execution time upper bound is more important in the time-critical situation. Regarding the knobs, the benefit of the multi-timescale methodology using multiple knobs can be evaluated

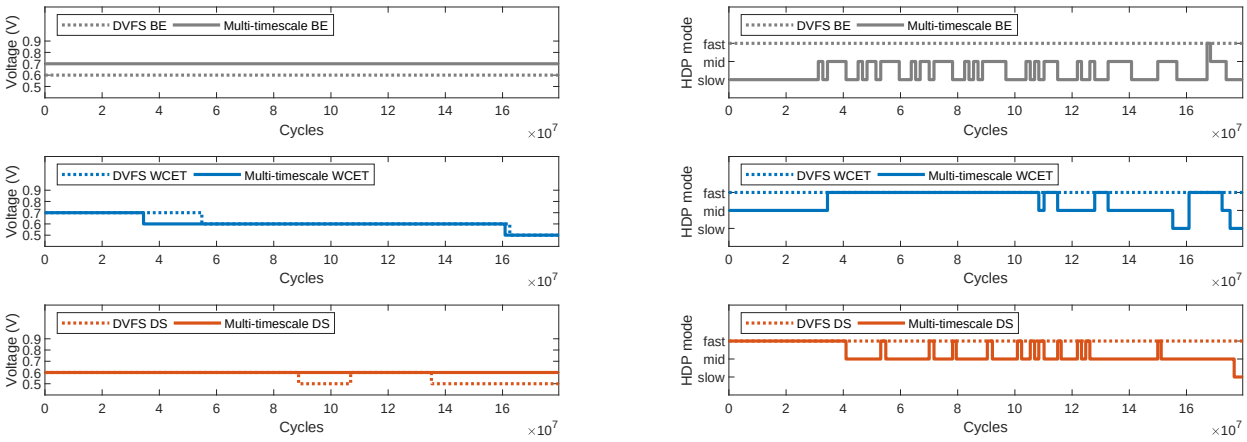

Fig. 14. Selected modes over the entire run of the JPEG decoding workload with a 8.0-ms frame period. (a) DVFS mode (b) HDP mode

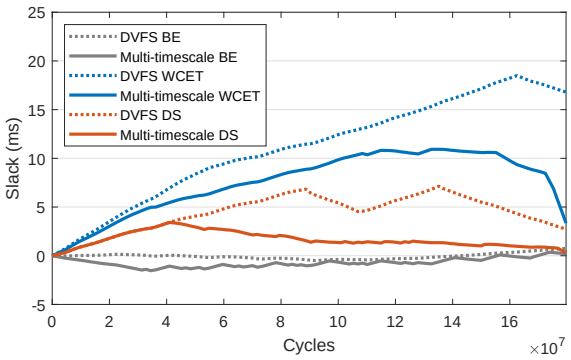

Fig. 15. Trace of slack over the entire run of the JPEG decoding workload with a 8.0-ms frame period

TABLE IV
SCENARIO LOOK-UP TABLE FOR THE JPEG-DECODING APPLICATION

Run-time parameters		Scenario	Cycle count	
Program flow	**Bits per pixel**		**max**	**avg**
Initialization	0 - 4.00	1	4771771	4703069
Decode the first MCU row	0 - 1.35	2	1423793	1271159
	1.35 - 4.00	3	1614701	1434089
Decode a middle MCU row	0 - 1.35	4	1609520	1414888
	1.35 - 2.10	5	1864506	1569669
	2.10 - 4.00	6	2175664	1767012
Decode the last MCU row	0 - 1.45	7	1838664	1661917
	1.45 - 2.50	8	2055572	1790937
	2.50 - 4.00	9	2266653	1991362

from the comparison between the DVFS DS method and the multi-timescale DS method. It shows the improved knob configuration saves extra 16% energy in average. Its benefit is more prominent when the timing is loose, since its advantage of allowing the processor to run in slower modes for longer time, is more prominent in the non-time-critical situation. In conclusion, it is the combined effect of both features which makes our methodology have a significant energy saving in all time-criticality situations.

B. Case II: JPEG Decoding

The inputs of the application are streams of JPEG images extracted from sample QCIF videos (174 x 144 resolution). Each JPEG image represents a frame of the video. The JPEG images are sequentially provided to the application to simulate a multimedia streaming workload of a constant frame rate. The JPEG decoding application generally consists of two main parts. The first part is the initialization phase, in which the application builds up the data structure and reads the input image. Then in the second part, the application decodes the image by rows of Minimum Coded Units (MCU). Each MCU row is a 174 x 16 fraction of the image. In this case, a TN is defined as the process of initialization or decoding an MCU row. Since one image is composed of nine MCU row, the process of decoding one image is divided into 10 TNs in this way.

At design-time, we profiled 12 video streams. The number of frames of each video ranges from 150 to 494. Two parameters are selected as the run-time parameters for the scenario detector. One is the different processes in the program flow (initialization, decoding the first MCU row, decoding a middle MCU row, decoding the last MCU row). Each of these processes exhibits distinct characteristics of execution time and energy consumption. An especially interesting observation is that even in the iteration of the main loop of decoding MCU rows, the costs of the first and the last iterations are systematically different from the intermediate iterations. This shows that the program flow can serve as a parameter for predicting execution time, and has much to be exploited for mitigation.

The other one is the *bit per pixel* (BPP) of the image, which is defined as the length of encoded bit stream divided by the number of pixels. BPP represents the data density of the image, which is related to the computation effort to decode an image. In simulation, BPP is highly correlated to

TABLE V
MITIGATION RESULT SUMMARY WITH DIFFERENT METHODOLOGIES AND BENCHMARKS

Methodology	Benchmark							
	Case I: Quantization of ADPCM encoding				Case II: JPEG decoding			
	Deadline miss rate	Energy ratio*			Deadline miss rate	Energy ratio*		
		min	avg	max		min	avg	max
DVFS BE	13%	0.47	0.59	0.95	52%	0.74	0.84	0.93
Multi-timescale BE	29%	0.49	0.58	0.92	46%	0.72	0.85	1.00
DVFS WCET	0%	1.00	1.00	1.00	0%	1.00	1.00	1.00
Multi-timescale WCET	0%	0.72	0.90	1.00	0%	0.82	0.91	0.99
DVFS DS	0%	0.66	0.83	1.00	0%	0.80	0.89	1.00
Multi-timescale DS	0%	0.54	0.68	0.97	0%	0.75	0.84	0.92

* Energy ratio is calculated w.r.t. the energy of the DVFS WCET method.

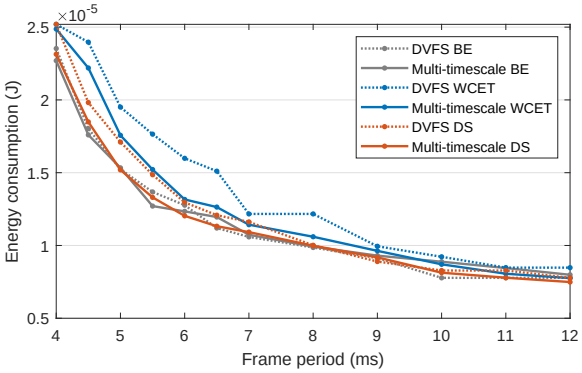

Fig. 16. JPEG decoding workload energy consumption of different methods under different time-criticality conditions

the execution time of decoding the image. Also, the calculation of BPP is simple and can be done before reading the content of the encoded bit stream. Therefore, BPP calculation can be performed for the upcoming TNs in the buffer to detect the scenario. These properties make BPP a suitable run-time parameter for scenario detection.

As a result, 9 dynamic scenarios are created. The details of the scenario set are shown in Table IV. At run-time, an image stream of 10 images is used as the target workload, which totals 100 TNs.

Fig. 14 shows the DVFS and HDP modes selected during the entire run of the 8.0-ms frame period case. And Fig. 15 shows the traces of slack of each of these mitigation methods. The behaviors of each methodology in these figures are similar to Case I, though the application is different. This shows that the characteristics of methodologies we discussed in Case I are not specific to a certain application. They are universal across workloads. Under a different workload, the multi-timescale DS method can still run in a smooth slack curve which is always positive and close to zero. This proves that the merits of this method are general and independent of workload.

Fig. 16 shows the energy consumption of different methods under different frame periods. Compared to the conventional single-DVFS WCET-based method, the multi-timescale DS method can save 16% energy in average and 25% energy in maximum. And compared to the DVFS best-effort method, the energy consumption of the multi-timescale DS is about the same and 5% greater in the worst case. In some cases, the multi-timescale DS method consumes even less energy, which is 6% in the best case. This result does not mean that the proposed method can ensure timing guarantee without an energy penalty, since the energy consumption can be affected by multiple factors, such as the energy efficiency of the selected modes and the activities in the workload. Nevertheless, it shows that in some context the energy penalty can be minimal. Also, the energy differences among the methodologies are smaller than Case I, which indicates that the JPEG-decoding workload is less dynamic, so the potential of energy saving by mitigation is smaller. Therefore, the energy saving achieved by mitigation is highly workload dependent.

C. Result Summary

We summarize the mitigation results of both benchmarks under all frame periods in Table V, in which the average energy ratio is calculated w.r.t. the energy of the DVFS BE method. The differences in results of both benchmarks may be attributed to several factors. First, the degree of dynamism of the workload could affect the performance of mitigation methodologies. A workload which is more dynamic in execution time, like Case I, could provide more potential of mitigation benefits. This explains why the energy consumption differences of methodologies are more prominent in Case I. Second, for the BE algorithm, the deadline miss rate varies between benchmarks, depending on the differences between the prediction and the real execution time. On the other hand, the WCET-based algorithm and the DS algorithm can both achieve zero deadline misses regardless of the benchmark.

VI. CONCLUSIONS

We propose a multi-timescale mitigation methodology for ensuring timing guarantee. The proposed methodology improves the way to derive the upper bound of execution time at run-time with the dynamic scenario based approach. Furthermore, it incorporates both the HDP and DVFS knobs to push the timescale of mitigation down to sub-millisecond level. Simulation shows that when running a highly dynamic workload, the proposed methodology can ensure meeting all deadlines, and at the same time reduce the dynamic energy consumption by 32% in average and 46% in maximum compared to the conventional WCET-based method with a single DVFS knob.

ACKNOWLEDGMENTS

The authors are grateful to Mr. Matheus Cavalcante and Mr. Florian Zaruba from ETH Zurich, Switzerland, for assistance in simulation on the RISC-V Ariane core.

REFERENCES

[1] D. Skinner and W. Kramer, "Understanding the causes of performance variability in HPC workloads," *Proceedings of the 2005 IEEE International Symposium on Workload Characterization*, 2005

[2] K. A. Bowman, "Adaptive and Resilient Circuits: A Tutorial on Improving Processor Performance, Energy Efficiency, and Yield via Dynamic Variation," in *IEEE Solid-State Circuits Magazine*, vol. 10, 2018

[3] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," in *ACM Computing Surveys*, vol. 37, Issue 3, 2005

[4] P. Yang *et al.*, "Energy-aware runtime scheduling for embedded-multiprocessor SOCs," in *IEEE Design & Test of Computers*, vol. 18, no. 5, 2001

[5] K. Lampka and B. Forsberg, "Keep it slow and in time: Online DVFS with hard real-time workloads." in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016

[6] S. Park *et al.*, "Accurate Modeling of the Delay and Energy Overhead of Dynamic Voltage and Frequency Scaling in Modern Microprocessors," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 5, 2013

[7] P. Pillai and K. G. Shin., "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP '01)*, 2001

[8] S. Munaga and F. Catthoor, "Systematic Design Principles for Cost-Effective Hard Constraint Management in Dynamic Nonlinear Systems," in *International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS)*, 2(1), 2011

[9] F. Zaruba and L. Benini, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, 2019

[10] M. R. Guthaus *et al.*, "MiBench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization*, 2001

[11] E. Hammari *et al.*, "Identifying data-dependent system scenarios in a dynamic embedded system," in *The International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2012

[12] D. Rodopoulos, F. Catthoor and D. Soudris, "Tackling Performance Variability Due to RAS Mechanisms with PID-Controlled DVFS," in *IEEE Computer Architecture Letters*, vol. 14, no. 2, 2015

[13] S. J. Tarsa, A. P. Kumar and H. T. Kung, "Workload prediction for adaptive power scaling using deep learning," in *2014 IEEE International Conference on IC Design & Technology*, 2014

[14] F. Yao, A. Demers and S. Shenker, "A scheduling model for reduced CPU energy," in *Proceedings of IEEE 36th Annual Foundations of Computer Science*, 1995

[15] D. Yakimets *et al.*, "Power aware FinFET and lateral nanosheet FET targeting for 3nm CMOS technology," in *2017 IEEE International Electron Devices Meeting (IEDM)*, 2017

[16] H. Falk *et al.*, "TACLeBench: A benchmark collection to support worst-case execution time research," in *Open Access Series in Informatics*, vol. 55, pp. 2.1-2.10, 2016

[17] J. Y. Lin, P. Weckx, S. Mishra, A. Spessot, and F. Catthoor, "Proactive Run-Time Mitigation for Time-Critical Applications Using Dynamic Scenario Methodology," in *Proceedings of the 2022 Design, Automation & Test in Europe*, 2022, in press

[18] J. Balkind *et al.*, "OpenPiton at 5: A Nexus for Open and Agile Hardware Design," in *IEEE Micro*, vol. 40, no. 4, pp. 22-31, 1 July-Aug. 2020

[19] Zhigang Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson and P. Bose, "Microarchitectural techniques for power gating of execution units," in *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, 2004

[20] D. Markovic, V. Stojanovic, B. Nikolic, M. A. Horowitz and R. W. Brodersen, "Methods for true energy-performance optimization," in *IEEE Journal of Solid-State Circuits*, vol. 39, no. 8, pp. 1282-1293, Aug. 2004

[21] T. Pering, T. Burd and R. Brodersen, "Voltage scheduling in the IpARM microprocessor system," in *Proceedings of the 2000 international symposium on Low power electronics and design*, pp. 96-101, Aug. 2000

Ji-Yung Lin received the M.Sc. degree in electronics engineering from National Taiwan University, Taiwan in 2013. From 2013 to 2018, he worked for TSMC, Taiwan, where he was engaged in design enablement for advanced technologies. He is currently pursuing the Ph.D. degree in electrical engineering in Katholieke Universiteit Leuven, Belgium, where he conducts research on reliability of digital systems in IMEC. His research interests include variability and reliability of VLSI, as well as design and automation of digital circuits and systems.

Pieter Weckx received the B.Sc degree in Electronic Engineering, M.Sc. degree in Nanoscience and –technology and Ph.D degree in Engineering from the Katholieke Universiteit Leuven - Belgium, in 2009, 2011 and 2016 respectively. In 2015 he joined imec where he is currently R&D Manager leading the research group for future scaled CMOS technologies, spanning from device to components for systems.

Subrat Mishra received his dual M.Tech. and Ph.D degree in Electrical Engineering from Indian Institute of Technology, Bombay (IITB), India in 2018. In May 2018, he joined imec as a post-doctoral research fellow in the circuit and device architecture team. In June 2019, he received Marie Sklodowska Curie fellowship under the EU's Horizon 2020 research and innovation program. His research interests include device to circuit and system level reliability and mitigation approaches in CMOS technology.

Alessio Spessot received the Laurea (cum laude) degree in physics from the University of Trieste, Italy, and the Ph.D. degree in solid-state physics from the University of Modena and Reggio Emilia, Italy, in 2003 and 2006, respectively. He has worked for STMicroelectronics, Numonyx and Micron till 2015, being involved in CMOS, Flash, and emerging memories devices (process development, integration, characterization, reliability, and modeling). Since 2016 he has been with imec (Leuven, Belgium), where he is a technical Director, exploring the roadmap of Logic and Memory Device from a Design and System Technology Co-Optimization perspective. He has authored more than 120 papers, and holds multiple US/EU issued patents.

Francky Catthoor (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Katholieke Universiteit Leuven, Leuven, Belgium, in 1987.

Between 1987 and 2000, he has headed several research domains in the area of synthesis techniques and architectural methodologies. Since 2000, he has been strongly involved in other activities with IMEC, Leuven, Belgium, including coexploration of application, computer architecture and deep submicron technology aspects, biomedical systems and IoT sensor nodes, and photovoltaic modules combined with renewable energy systems, all with IMEC. He is currently an IMEC Senior Fellow. He is also a part-time Full Professor with Electrical Engineering Department, Katholieke Universiteit Leuven (ESAT).

Dr. Catthoor has been an Associate Editor for several IEEE and ACM journals.