Proactive Run-Time Mitigation for Time-Critical Applications Using Dynamic Scenario Methodology

J. Y. Lin[1,2] P. Weckx[1] S. Mishra[1] A. Spessot[1] F. Catthoor[1,2]
[1]imec vzw., Leuven, Belgium
[2]Katholieke Univeriteit Leuven, ESAT, Belgium
ji-yung.lin@imec.be

***Abstract*—Energy saving is important for both high-end processors and battery-powered devices. However, for time-critical application such as car auto-driving systems and multimedia streaming, saving energy by slowing down speed poses a threat to timing guarantee of the applications. The worst-case execution time (WCET) is a widespread solution to this problem, but its static execution time model is not sufficient anymore for highly dynamic hardware and applications nowadays. In this work, a fully proactive run-time mitigation methodology is proposed for energy saving while ensuring timing guarantee. This methodology introduces heterogeneous datapath options, a fast fine-grained knob which enables processors to switch between datapaths of different speed and energy levels with a switching time of only tens of clock cycles. In addition, a run-time controller using a dynamic scenario methodology is developed. This methodology incorporates execution time prediction and timing guarantee criteria calculation, so it can dynamically switch knobs for energy saving while rigorously still ensuring all timing guarantees. Simulation shows that the proposed methodology can mitigate a dynamic workload without any deadline misses, and at the same time energy can be saved.**

Keywords—mitigation, variability, real-time scheduling, energy, dynamic scenarios, ALU, RISC-V

I. Introduction

Ensuring timing guarantee is crucial for time-critical applications such as car auto-driving system and multimedia streaming. Violating timing guarantee might cause severe consequences such as quality loss or safety issues. However, performance variability is a challenge to ensuring timing guarantee. Performance variability can originate from various sources, including dynamic program flow, fluctuation in operating conditions (voltage, temperature, aging, etc.), and non-deterministic hardware protocols like cache contention. To deal with performance variability, modern processors are equipped with knobs which can vary processor speed at run-time. Some typical knobs include dynamic voltage frequency scaling (DVFS) [1] and task mapping [2]. On top of these knobs, a run-time controller running the scheduling algorithm decides how to switch the knobs, ensuring all deadlines are met. Some examples of these algorithms can be found in [3][5].

Though this methodology is widely adopted, there are some insufficiencies in current methods. First, switching knobs such as DVFS requires a switching time of at least tens of microseconds [4]. For tasks which require short execution time, such as multimedia-streaming applications of a frame rate of hundreds of Hertz and above, the time overhead of even one switching of the DVFS knob could take up more than 1% of the available execution time in a frame. Hence, when a timing problem happens close to a time-critical deadline, the reaction time will be too long to provide full guarantees, as illustrated in Fig. 1. Moreover, multiple DVFS switching

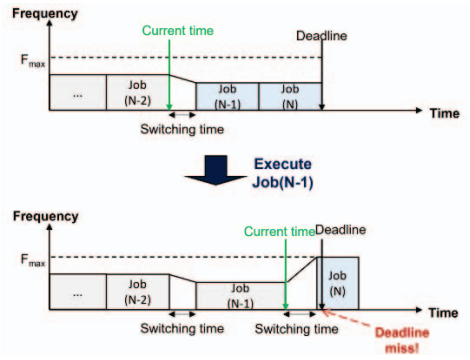


Fig. 1. Illustration of a example where DVFS cannot meet a time-critical deadline. An unexpected increase in the excution time of Job(N-1) makes the application miss its final deadline.

occurrences significantly lengthen the execution time and therefore introduces energy overhead. Hence, the mitigation should involve more fine-grained control at the sub-microsecond level.

Second, for ensuring timing guarantee, most algorithms use the worst case execution time (WCET) in scheduling [3][5]. However, WCET is very pessimistic and much greater than the actual execution time most of the time, so scheduling with WCET would greatly overestimate the execution time, and therefore reduce the energy saving. Futhermore, WCET assumes a static execution time model, which is insufficient to describe the highly dynamic hardware and applications nowadays. On the other hand, some algorithms use constant execution time [8], or predict the execution time by heuristics [14]. These algorithms cannot fully ensure all timing guarantees since there is still a possibility that the actual execution time is greater than the prediction.

In this work, a fully proactive run-time mitigation methodology is proposed, which can ensure timing guarantee while minimizing energy at the same time. In this methodology, we designed the *heterogeneous datapath* (HDP) knob, which is a fast-switching knob with a very short switching time of only tens of clock cycles. It enables the processor to mitigate in a very fine-grained time granularity of tens to hundreds of microseconds, when needed. Combined with a controller which swtiches the HDP knob in run-time, this methodology can proactively respond to any source of performance variability to ensure all timing guarantees.

II. Heterogeneous Datapath

A heterogeneous datapath (HDP) is a datapath with multiple modes. In each mode, the datapath executes the same function in different speed and energy level. At run-time, the processor can dynamically switch the mode to run the datapath in the desired speed and energy. Past literature has proposed

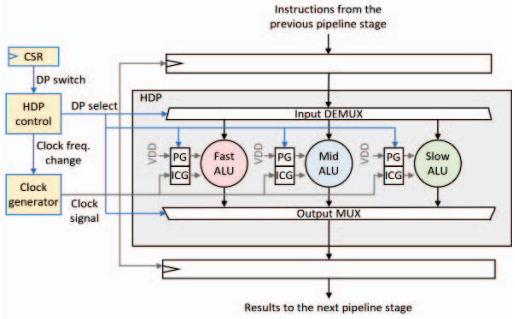


Fig. 2. Schematic of the heterogeneous datapath, implemented in the ALU

TABLE I. SUMMARY OF THE ALU DESIGNS IN HDP

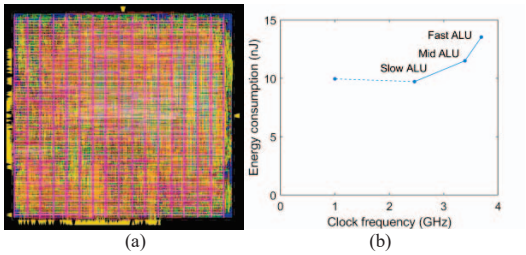
Design	Fast ALU	Mid ALU	Slow ALU
Clock frequency	3.69 GHz	3.39 GHz	2.46 GHz
Number of gates	18099	18092	12827
Multi-Vth libraries used	LVT, SVT	SVT, HVT	SVT, HVT
Technology	3nm [13]		
Operating condition	TT/0.7V/25C		

similar concepts of datapaths with multiple modes, such as variable-latency functional units for minimizing average latency [7], and heterogeneous functional units in superscalar processors for maximizing throughput [9]. However, to our knowledge, this work is the first work to extend this concept to mitigation for performance variability.

In this work, the heterogeneous datapath is implemented in the ALU of the RISC-V Ariane core [10]. Fig. 2 shows the schematic of the heterogeneous datapath, in which three ALUs with the same function are designed with different clock frequencies and energy levels. At run-time, only one ALU is active at one time. To reduce the energy overhead, clock gating and power gating (PG) techniques are implemented in each ALU domain. When an ALU is idle, its integrated clock gater (ICG) and power switch shut down the local clock network and power supply respectively to prevent unnecessary energy consumption. The input demultiplexer (DEMUX) and the output multiplexer (MUX) are added to the interface of the ALUs to assign input signals to the active ALU and receive output signals from the active ALU.

A control and status register (CSR) [11] is assigned for controlling the HDP. To switch the mode of the HDP, the processor simply MCUs a CSR-write instruction to change the value stored in the CSR. Once sensing the CSR value change, the HDP control unit will wait for the instructions currently executed in the active ALU to be finished. The waiting time is 66 cycles at most, which is the longest latency of all ALU functions. At the same time, the control unit can wake up the ALU which the processor is going to switch to. After that the control unit switches the active ALU and signals the clock generator to change the clock frequency. Finally, the control unit enables clock gating and power gating on the idle ALU which was previously active.

The switching time of HDP, including the waiting time, the changing time of the clock frequency, and the wake-up time of power gating, only adds up to tens of cycles in total, enabling sub-microsecond reaction time. In a fine-grained control situation in which the HDP switches once in tens of microseconds, the switching time would only take up about



(a) (b)

Fig. 3. (a) Physical design of the fast ALU. (b) Energy consumption v.s. clock frequency trend of the three ALUs in HDP. Energy consumption results are simulated with a workload of decoding one block of an JPEG image

0.1% of the execution time. Therefore, HDP is suitable for a fine-grained control situation.

Our heterogeneous datapath design is based on the 3 nm technology [13]. The summary of each ALU design is shown in Table I. Each ALU uses two kinds of devices of different threshold voltages (Vth), so the energy consumption can be optimized by using lower-Vth devices on non-critical paths. Fig. 3(b) shows the ALU energy consumption simulated with a workload of decoding one block of an JPEG image. The results show that compared to the fast ALU, the mid ALU can save 15% energy by 8% speed degradation, while the slow ALU can save 28% energy by 33% speed degradation. In the region where the clock frequency is lower than the slow ALU, the energy trend saturates and even increases due to larger leakage energy coming from longer execution time. Therefore, designs in this region are not reasonable choices. We expect the energy efficiency of HDP could be better in the future by the improvement of EDA tools, since the design optimization of HDP by state-of-the-art EDA tools is limited due to the complexity of the timing constraints. Also, the HDP concept can be applied to other processor components, so that processor-wide energy saving can be achieved.

We implemented the controller with dynamic scenario methodology [6] to switch the HDP knob in run-time. This methodology performs scheduling by combining both the likely case prediction and the refined upper bound of the execution time, so it can proactively respond to any source of performance variability to ensure all timing guarantees. The principles of this approach are elaborated in [6]. Here, we instantiate those principles for mitigating timing variations, in a novel context with a new approach tuned to our specific objectives. However, this novel context extension is not the focus of this paper, and it will be discussed in future work.

III. SIMULATION SETUP

Fig. 4 shows the simulation flow of the fine-grained run-time mitigation framework. The methodology is simulated on the RTL-design of the RISC-V Ariane core [10].

A. Processor simulation flow

We developed a cycle-accurate processor simulation flow to accurately simulates the execution time and the energy consumption of the processor. First, RTL-level simulation is performed to run the workload on the RTL model of RISC-V Ariane core. During the simulation, the execution time results are acquired. At the same time, the input patterns of the ALU are recorded. Then, a gate-level simulation is performed on the netlist of each ALU. This simulation uses the input patterns recorded from the RTL simulation to reconstruct the activities of each net in the ALUs. At last, the ALU netlist and

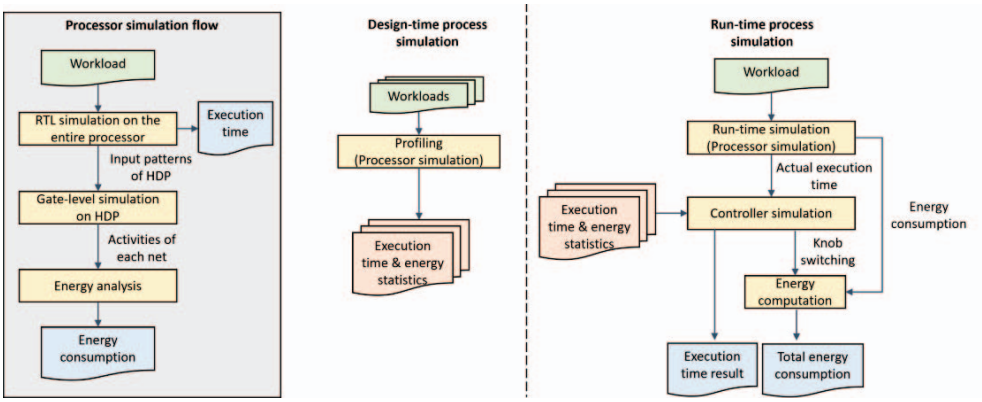


Fig. 4. Simulation flow of the run-time mitigation framework

the activities in its net are read by the commercial power analysis tool to calculate the energy consumption of the ALUs for all HDP modes.

In our simulation, the time and dynamic energy overheads of HDP switching are not incorporated, since they are very small compared to the total execution time and the ALU energy consumption for the reasons described in Chapter II, so they can be neglected. The leakage energy consumption from the idle ALUs is also left out of the analysis, since these ALUs are power-gated and their remaining energy consumption is very small compared to the active ALU.

B. *Overall simulation flow*

The overall simulation flow is composed of two parts, the design-time process simulation and the run-time process simulation. During design-time process simulation, we profile the application by processor simulation with different input files, to acquire the statistics of execution time and energy for the use of run-time scheduling of dynamic scenario methodology. Then during run-time process simulation, first we perform processor simulation running target workload with the target inputs to acquire its execution time and energy. These results are read by a controller simulator of dynamic scenario methodology, which uses all the data provided by design-time process to derive 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.

IV. Expermental Results

The work load we use in the experiment is the JPEG-decoding application of MiBench [12]. The inputs of the application are streams of JPEG images extracted from sample QCIF videos (174 x 144 resolution). The JPEG images are sequentially provided to the application to simulate a multimedia streaming workload of a constant frame rate, so there is a hard deadline for decoding each image. The process of decoding one image is cut into 10 fine-grained jobs, so that the execution time of each job is about hundreds of microseconds. One job represents the process of application initialization or decoding a row of minimum coded units (MCU). At run-time, an image stream of 10 images is used as the target workload, which totals 100 jobs. The deadline of each image is 6.5 ms, so equivalently the frame rate is 154 Hz.

Fig. 5 shows the traces of clock frequency and slack of the entire run. The slack is defined as the amount of time by which

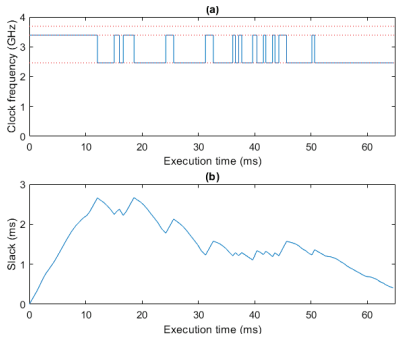


Fig. 5. Results of mitigation of JPEG image stream decoding workload by dyanmic scenario methodology (a) Trace of clock frequency over the whole execution time. The red dash lines represent the clock frequency levels of three ALUs. (b) Trace of slack over the whole execution time.

a job ends before its deadline. The whole run can be divided into three phases. First, at the start of the workload, the processor is running in fast modes to make sure that even in the worst case the deadlines will not be violated. During this phase, the slack increases rapidly. Then, starting from about 10ms, the processor has gained enough slack to ensure timing guarantee, so it starts to run at a slow speed to save energy. In this phase, the processor is changing between two neighboring modes to run equivalently at an intermediate speed of the two modes, and the slack is in a stable level with fluctuation due to variability. Finally, starting from about 50ms, the workload is about to finish, so the processor runs at a slow mode to consume the residual slack to save energy.

We compare our methodology with three representative reference cases, which reflect the different existing state-of-the-art options. The first one is the case without mitigation. In this case, the processor has only one ALU which is the fast ALU of the HDP, so the processor can only run at the fastest mode and does not have the option to switch to the slower modes to save energy. The second reference is a WCET-based method, in which the processor does not switch modes in the middle of the decoding of an image. This means the controller only switch knobs every 10 jobs. Also, this method always predicts the execution time to be the WCET of decoding an image to ensure timing guarantee. The third reference is the PID-controlled scheduling proposed in [3], which adopts a PID controller to regulate the slack to be around zero.

Fig. 6 shows the traces of slack of three different mitigation methods. Of these methods, the PID-controlled scheduling is the one which cannot ensure the timing

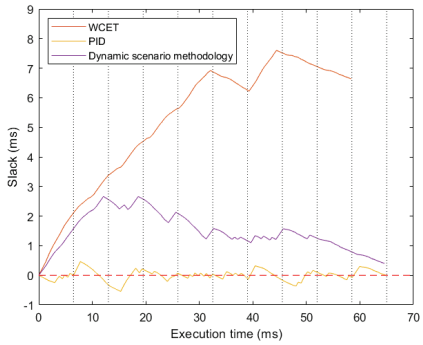


Fig. 6. Traces of slack of three mitigation methods. The black dotted lines indicate the deadlines of one image frame. Negative slack values at the time of these deadlines means deadline misses.

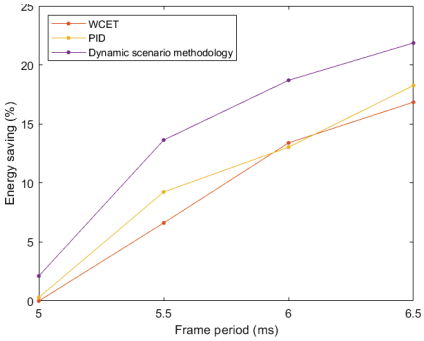


Fig. 7. Relations between energy saving of all methods and time criticality of the workload, represented by the frame period of the image stream

guarantee. At the time of several deadlines, the slack values of PID-controlled scheduling are negative, which indicates deadline misses are encountered. During the whole workload, this method misses 50% of the deadlines. Therefore, this method is not eligible to be a scheduling algorithm for time-critical applications. For the other two methods, the slack is above zero all the time, which means the timing guarantee can be ensured. However, the dynamic scenario methodology with HDP can keep the slack value at a much lower level than the WCET-based method, which indicates our methodology is better at making use of run-time variability to save energy.

Fig. 7 shows the relations between energy saving of all methods and time criticality of the workload, represented by the frame period of the image stream. In terms of energy saving, the dynamic scenario methodology with HDP is best of all methods, particularly when the workload has strict deadlines. As shown in the case of 5.5-ms frame period, the dynamic scenario methodology consumes 13.6% less total energy than the non-mitigation case, while the WCET-based method and PID-controlled method only consume 6.6% and 9.2% less total energy respectively than the non-mitigation case. Equivalently, the energy saving of the dynamic scenario methodology is 2.1 times of the WCET-based method.

V. Discussions

Thanks to the fast-switching characteristic, HDP enables mitigation in more fine-grained time granularity down to sub-microsecond level. However, HDP should not be regarded as a replacement for coarse-grained knobs such as DVFS. In fact, HDP is intended to work together with DVFS in a complimentary way. One can envisage a hierarchical mitigation framework, where HDP can response proactively to mitigate small and sudden variability such as random telegraph noise, and DVFS can react to variability of large scale and small gradient such as temperature fluctuation. It requires the cooperation of knobs of various time ranges and energy efficiency scale to cope with all kinds of variability.

VI. Conclusions

In this paper, we proposed a fully proactive run-time mitigation methodology for ensuring timing guarantee while minimizing energy consumption. This methodology features a fine-grained HDP knob. Simulation with a dynamic scenario methodology controller shows that the proposed methodology can mitigate a dynamic workload without violating any deadlines. Energy consumption results also indicate that the proposed methodology can save more energy than other mitigation methods.

Acknowledgment

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

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

[3] 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

[4] 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

[5] 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

[6] 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

[7] T. Sato and I. Arita, "Execution Latency Reduction via Variable Latency Pipeline and Instruction Reuse." in *International Euro-Par Conference Manchester on Parallel Processing (Euro-Par '01)*, 2001

[8] 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

[9] L. Gwennap, "MIPS R10000 Uses Decoupled Architecture," in *Microprocessor Report*, 1994

[10] 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

[11] A. Waterman and K. Asanović, "The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 1.12-draft", RISC-V Foundation, 2019

[12] 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

[13] 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

[14] 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