LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-JRNL-809714

Dynamic Undervolting to Improve Energy Efficiency on Multicore X86 CPUs

P. Koutsovasilis, K. Parasyris, C. D. Antonopoulos, N. Bellas, S. Lalis

May 6, 2020

Transactions On Parallel and Distributed Systems

Disclaimer

Dynamic Undervolting to Improve Energy Efficiency on Multicore X86 CPUs

Panos Koutsovasilis, Konstantinos Parasyris, Christos D. Antonopoulos, Nikolaos Bellas, Spyros Lalis

Abstract—Chip manufacturers introduce redundancy at various levels of CPU design to guarantee correct operation, even for worst-case combinations of non-idealities in process variation and system operating conditions. This redundancy is implemented partly in the form of voltage margins. However, for a wide range of real-world execution scenarios these margins are excessive and merely translate to increased power consumption, hindering the effort towards higher energy efficiency in both HPC and general purpose computing.

Our study on the x86-64 Haswell and Skylake multicore microarchitectures reveals wide voltage margins, which vary across different microarchitectures, different chip parts of the same microarchitecture, and across different workloads. We find that it is necessary to quantify voltage margins using multi-threaded and multi-instance workloads, as characterization with single-threaded and single-instance workloads that do not stress the CPU to its full capacity typically identifies overly optimistic margins that lead to errors when applied in realistic program execution scenarios.

In addition, we introduce, deploy and evaluate a run-time governor that dynamically reduces the supply voltage of modern multicore x86-64 CPUs. Our governor employs a model that takes as input a set of performance metrics which are directly measurable via performance monitoring counters and have high predictive value for the minimum tolerable supply voltage (V_{min}), to predict and apply the appropriate reduction for the workload at hand. Compared with the conventional DVFS governor, our approach achieves up to 42% energy savings for the Skylake family and 34% for the Haswell family for complex, real-world applications.

Index Terms—Energy Efficiency, Undervolting, Low-Energy and Low-Power Technologies Measurement, Modeling

1 Introduction

Aggressive CMOS technology scaling into lower nanometer geometries has led to variability of transistor characteristics, resulting into increased failure rates in modern CPUs. Traditionally, techniques for dealing with transistor variability involve extra provisioning in logic and memory circuits, in the form of increased voltage margins, reduced operating frequencies and error correction circuitry. Such voltage margins are specified at design time by taking into account the implementation technology, power budget, worst case timing paths, operating conditions and fabrication process variations.

One of the major concerns when building the new generation of High Performance Computing (HPC) systems and overcoming the exascale barrier, is power consumption. HPC systems and datacenters typically have a power consumption envelope in the order of MWatts. Voltage margins lead to significant power overheads, which is in conflict with the challenges of limiting power dissipation. The average power cost of these margins can be in the order of 35% [1], yet most of the time these margins are excessive and translate to unnecessary power overhead, as the worst-case combinations that were considered at design time may appear only rarely or even not at all during the life cycle of a given processor. Providing an end-to-end approach that effectively reduces these margins, could have a significant impact on such systems, as the resulting energy gains would enable the utilization of extra resources in order to support additional parallelism and increased computational capacity.

A critical challenge though, is to reduce the margins as much as possible yet without compromising the reliability of the system. To this end, we have designed, developed, deployed and evaluated a run-time Extended Dynamic Voltage Scaling (xDVS) governor for off-the-self systems, which dynamically and adaptively eliminates voltage margins by applying a reduced, yet safe CPU supply voltage to lower power and energy consumption. The xDVS governor monitors the utilization of CPU resources and uses a prediction model to estimate and apply a new safe supply voltage to the CPU. To train the prediction model we perform an offline characterization of voltage margins on Haswell i7–4790 and Skylake Xeon E3–1220 v5 processors, using a diverse set of benchmarks which stress different components of the CPU microarchitecture.

The main contributions and outcomes of our work are:

i. We characterize the voltage margins of the x86-64 Skylake and Haswell processors for both single- and multi-instance/threaded benchmarks. To the best of our knowledge, we are the first to evaluate the voltage margins of a 14nm (Skylake) processor and also the first to evaluate the V_{min} of x86 multi-core CPUs using multi-instance/threaded benchmarks. Our offline characterization spans from common benchmark suites to stress tests and power viruses. We show that it is critical to characterize voltage margins using multi-instance/threaded workloads since in 82% (73%) of the cases, such characterization in Haswell (Skylake) results into more conservative margins compared with characterization using only the single-instance versions. We also find that the voltage margins of the tested parts of Skylake and the Haswell family reach up to 22% and 13%

- *P. Koutsovasilis, C. D. Antonopoulos, N. Bellas and S. Lalis are with the Department of Electrical and Computer Engineering, University of Thessaly, Greece.E-mail: {pkoutsovasilis, cda, nbellas, lalis}@uth.gr*
- *K. Parasyris was with the Department of Electrical and Computer Engineering, University of Thessaly, Volos 38446, Greece.E-mail: parasyris1@llnl.gov*

Manuscript received August 02, 2018; revised July 14, 2019; revised May 8, 2020.

respectively, of their nominal supply voltage.

ii. We develop a model that takes as input selected CPU performance counters and core utilization, and estimates the voltage margin of the workload on the specific CPU part for the base CPU frequency. This estimation can be exploited to safely undervolt CPUs in order to achieve lower power consumption and higher energy efficiency.

iii. To the best of our knowledge we are the first to implement and deploy an Extended Dynamic Voltage Scaling (xDVS) governor, which, guided by such a model, dynamically adjusts the CPU supply voltage to levels below conservative nominal values. Compared with the stock Intel (*P-state*) DVFS governor, our approach achieves energy savings up to 42% for Skylake and 34% for Haswell CPUs with negligible overhead on performance.

iv. We validate the robustness of xDVS with a set of long-running experiments over 23 days without experiencing a single failure. In addition, we show that significant energy gains up to 26% can be achieved, even when taking into account the cost of checkpointing and recovery to address the increased failure probability due to undervolting in large-scale systems with hundreds of nodes.

The rest of the paper is organized as follows. Section 2 outlines the features of the CPUs used in our work. Section 3 presents the offline voltage margins characterization approach and results. Section 4 discusses the V_{min} prediction model. Section 5 focuses on the design and implementation of the xDVS governor and Section 6 presents the experimental evaluation. Section 7 discusses system reliability and fault-tolerance aspects. Section 8 provides an overview of related work. Finally, Section 9 concludes the paper.

2 Background

In this section we explain how we adjust the supply voltage of the Haswell and Skylake CPUs below nominal values. We also briefly discuss their power saving modes, and outline their performance monitoring functionality.

Latest generations of Intel CPUs are powered through the Fully Integrated Voltage Regulator (FIVR) [2] which selects the optimal supply voltage V_{dd} for CPU cores according to the frequency of the CPU cores and the executing workload (Figure 1). It is possible for software to alter the supply voltage, as shown in Equation 1, by writing an offset value into specific Model-Specific Registers (MSRs). Extreme Tuning Utility (XTU) is an Intel tool for Windows, which uses MSRs to perform user commanded undervolting. In this work we use Linux-based systems, so we implemented a similar functionality.

$$V'_{dd} = V_{dd} - MSR_{offset} \quad (1)$$

Although there are separate MSRs for the core and uncore components, we empirically observed that the supplied voltage changes only when both registers are set to the same value. Thus, we operate the core and the uncore components with the same voltage offset (MSR_{offset}). Note that FIVR does not support an independent per-core adjustment of the offset, therefore the offset applies to all CPU cores.

To save energy when idling, a CPU core can be put in a low-power mode. There are several such modes, or so-called *C-states*, which perform clock and power gating to different units inside the core. C-states are numbered starting from *C*0, the normal CPU operating state in which all CPU modules are powered up and clocked. Higher power saving states result in an increasing number of circuits and signals being power- or clock-gated, putting the core into a deeper sleep state. The deeper the core sleep state, the higher the performance and energy penalty to revert back to *C*0.

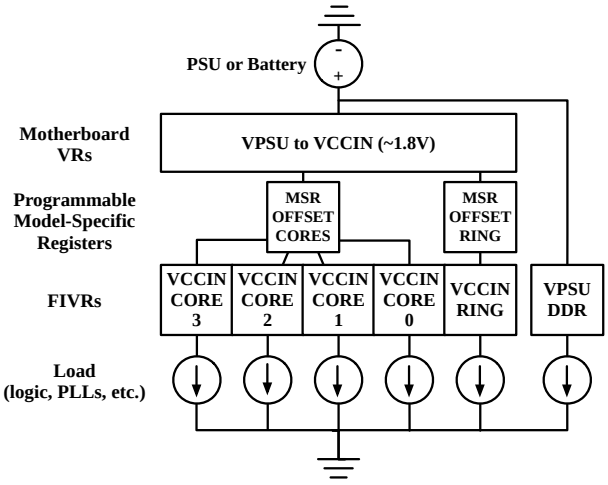


Fig. 1: Overview of Intel FIVR.

Both CPU families have on-chip Performance Monitoring Units (PMUs), used to quantify the interaction of applications with hardware. However, only up to 8 performance counters per core can be monitored simultaneously.

3 Offline Quantification of Voltage Margins

In this section, we quantify the minimum supply voltage V_{min} required to avoid silent data corruption (SDC) errors and CPU/system crashes during application execution. We use both single- and multi-instance/threaded benchmarks included in the SPEC CPU2006 [3] and Parsec [4] benchmark suites, the Linpack [5] benchmark, as well as a number of stress tests (Prime95 [6], Firestarter [7], Stress-NG [8]). For Stress-NG in particular, we include all 68 sub-tests that stress the CPU. Each benchmark is executed in two modes: either occupying a single core, or all cores of the target CPU. Single-instance/threaded experiments are executed once per core, with the running thread pinned on the respective core while the rest of the cores are idle. To fully utilize the CPU, multi-instance/threaded benchmarks are executed with a degree of parallelism equal to the number of cores, whereas in the case of single-instance benchmarks we achieve full utilization by co-executing as many copies of the benchmark as the number of cores. We refer to the combination of benchmark versions and core mappings as *configurations*.

The V_{min} for each benchmark configuration is determined using a binary search algorithm, which looks for the maximum applicable MSR_{offset} within a range $[low, high]$. Initially, $low = 0$mV and $high = 500$mV. In the first step of the search we set MSR_{offset} equal to the middle of this interval. During execution we monitor the system for Machine Check Exceptions (MCEs), application crashes, kernel panics etc. When an experiment terminates, the output is compared against the correct, "golden" output in order to detect any SDCs. To account for potentially non-deterministic behavior, we experiment with every

configuration and MSR_{offset} combination 10 times. If all experiments complete successfully, the region $[low, MSR_{offset}]$ is marked as safe, the low bound is increased to $low = MSR_{offset}$, and MSR_{offset} is adjusted accordingly. If problematic behavior is detected during any experiment, the region $[MSR_{offset}, high]$ is marked as unsafe, and the high bound is decreased to $high = MSR_{offset}$. The algorithm terminates when the interval width becomes less than 5mV. No human intervention is needed, since we reboot after a CPU crash using an external hardware watchdog.

We perform the experimental analysis on six workstations, four featuring an Intel Skylake Xeon CPU called *Skylake 1 - 4*, and two featuring an Intel Haswell i7 CPU, called *Haswell 1, Haswell 2*. All workstations run Ubuntu 16.04LTS with Linux Kernel version 4.10.0-38-generic. Table 1 outlines the characteristics of each workstation as well as the nominal supply voltage for both architectures under maximum utilization.

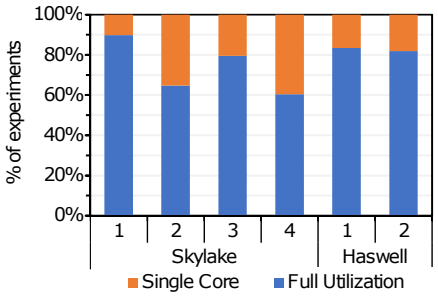
Parameters	Skylake Workstation	Haswell Workstation
CPU	Xeon E3–1220 v5	Core i7–4790
Technology	14nm	22nm
# of Cores	4	4
CPU Base Freq.	3.00GHz	3.60GHz
Supply Voltage (V_{dd})	1.15V	1.07V
CPU TDP	80 W	84 W

TABLE 1: Characteristics of the workstations.

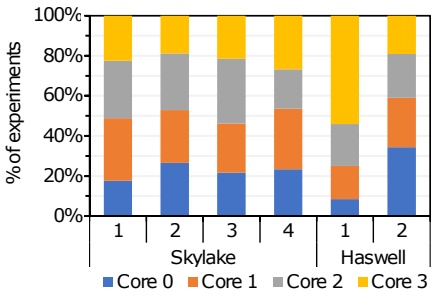
Our experimental evaluation focuses on the base (max) CPU frequency which can also provide ample opportunities for harnessing available voltage margins (as shown later in Figure 4).

Figure 2 illustrates the experimentally identified MSR_{offset} for the four Skylake and two Haswell CPUs, running 34 benchmarks. Since the SPEC CPU2006 benchmarks are

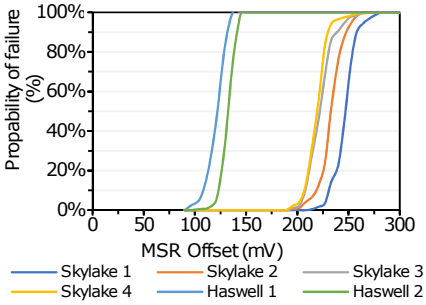
Fig. 2: Evaluation of MSR_{offset} settings for 34 benchmarks (10 runs each) on each workstation; the higher the bar, the wider the exploitable voltage margin. The horizontal lines show the maximum (red) and minimum (black) values of MSR_{offset}.



(a) Percentage of experiments for which single core, or multi-instance/threaded workloads resulted to narrower voltage margins.



(b) Percentage of experiments in which any given core resulted being the weakest during our characterization phase.



(c) Average (across all configurations) failure probability CDF for each CPU, with respect to the applied MSR_{offset}.

Fig. 3: Summary of statistical results from the CPU characterization phase.

single-instance, the respective margins for the fully utilized CPUs are determined by executing simultaneously four instances of the benchmark on each 4-core CPU. Voltage margins span from 17% to 24% and from 9% to 13% of the nominal V_{dd} for the Skylake and Haswell microarchitectures. The difference between min and max voltage margin values (7% and 4% of nominal V_{dd} for Skylake and Haswell, respectively) is the workload dependent margin.

Figure 3a shows that in the large majority of the experiments, multi-instance/threaded benchmark executions have narrower margins than when running the benchmarks in a single-threaded/instance configuration. This observation emphasizes the importance of using both single- and multi-instance/threaded programs to correctly assess V_{min}.

Moreover, unlike previous studies on ARMv8 [9] and Itanium [10] CPUs that revealed intermediate voltage regions of unsafe operation where indications of erratic behavior may be observed, for the architectures investigated in this study the transition to unreliable operation when voltage drops below V_{min} is abrupt and always leads to crashes. Even in the few cases where *SDCs* or *MCE* errors were observed, these errors were accompanied by an immediate system or application crash.

We also observe margin variations across different cores of the same part (difference between margins of the strongest and weakest core of each CPU). The V_{min} variation when executing the same single-instance benchmarks with different cores can reach up to 45mV and 32mV for Skylake and Haswell, respectively. In Figure 3b we present the percentage of times each core was ranked as the weakest in terms of voltage margin when executing a single-instance benchmark. In contrast to the findings of the characterization of ARMv8 and Itanium, the CPUs in this study do not exhibit the pattern of a consistently weakest and a consistently strongest core; in our case the strongest/weakest core varies across chips and even on the same chip across benchmarks (Figure 3b).

Figure 3c shows the cumulative distribution function (CDF) of the average (across all *configurations*) failure probability of each CPU, as a function of the applied MSR_{offset}. Note that lower slope CDF curves indicate a broader range of undervolting opportunities, depending on the characteristics of the workload and the resource pressure exercised. For example, *Skylake 4* offers an MSR_{offset} range between 196mV to 271mV in which different benchmark configurations will run successfully. On the contrary, *Haswell 2* has a narrower dynamic range (105 to 145mV) exhibiting a stepwise behavior. All four parts of the Skylake family have similar margins, with *Skylake 1* being able to operate at lower V_{dd} values than the rest.

Moreover, we identified the voltage margins for a range of operating frequencies, for both CPU architectures. Figure 4 shows that for each frequency there is a workload-independent (green part, static) and a workload-dependent (yellow part, dynamic) voltage margin which allow CPU voltage reduction without any crashes or SDCs. The static margin corresponds to a voltage range where no applications fail. The dynamic margin, in turn, corresponds to a voltage range where some – yet not all – applications fail during the characterization. The workload-dependent part of the margin increases (albeit slightly) at higher frequencies. Specifically, for Skylake 2 at 3.0GHz, the total margin is 268mV (24.5% of the nominal supply voltage), out of which 62mV is attributed to the workload-dependent margin. On the other hand, at 1.0GHz frequency, the margin is 199mV (27.5% of the nominal voltage), but the workload-dependent range is only 4mV (1.9%). Similarly, for the Haswell 2 at 3.6GHz, the total margin is 145mV (14.4% of the nominal supply voltage) and the workload-dependent margin is 52mV. At 1.0GHz frequency, voltage margin is 318mV (39.7% of the nominal voltage), but the workload-dependent range is only 10mV (3.1%).

The Skylake family exhibits wider margins compared with the Haswell family, by 103mV on average. Essentially, CPUs are black boxes and the observed nominal settings and voltage

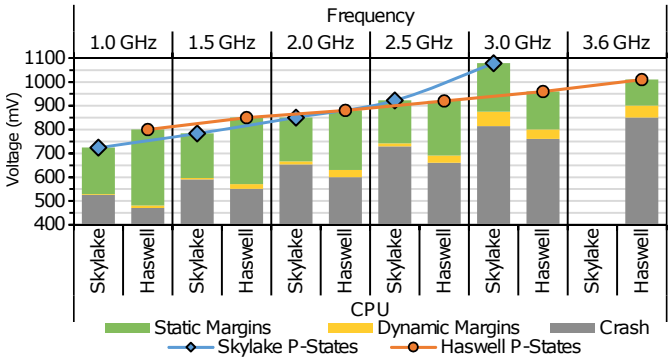


Fig. 4: Voltage margins for a range of CPU operating frequencies for Skylake 2 and Haswell 2.

margins are a result of physical limitations (geometry), architectural limitations, designer decisions, targeted market, etc. More specifically, as shown in Figure 4, at 3.0GHz frequency with nominal voltage settings, Skylake operates at 1078mV and Haswell at 960mV. As a result, the reduction of voltage margins (MSR_{offset}) is lower for the Haswell CPU but in terms of the lowest subnominal voltage, the differences are not that profound.

4 Voltage Margins Modeling and Estimation

Microarchitectural events that disrupt the flow of pipeline execution and influence pressure on CPU resources significantly affect the minimum CPU supply voltage (V_{min}) required for error-free execution [11]. As software dynamically interacts with the underlying hardware, such events appear at different intensities, therefore the effective V_{min} may vary during execution (dynamic voltage margins). This presents opportunities for dynamic, workload-aware undervolting at execution time.

This section introduces the methodology we used to build a model that predicts an operating voltage V'_{dd}, which is lower than the nominal supply voltage V_{dd} but nevertheless safe, i.e., higher than V_{min} for the specific combination of workload characteristics, execution configuration and CPU part. Figure 5 illustrates the four steps of our approach: (1) We determine the V_{min} for different cores (and all cores together) of the target CPU part for different workloads via offline characterization, as discussed in the previous section; (2) We profile the same workloads to quantify their interaction with the CPU using online performance counters; (3) We combine V_{min} characterization and the profiling data to fit two models (based on Random Forest Regression [12]) one for single-instance execution and one for multi-instance/threaded execution that dynamically predict a safe supply voltage V'_{dd}; (4) and finally the models are used by a dynamic voltage scaling governor to dynamically adjust the supply voltage based on the models, predictions as well as the resource pressure of the CPU. Note that this can be relatively time-consuming (in our case, the characterization took two days) and thus may not be affordable during the CPU production stage. However, in HPC and cloud environments it could be performed as part of the deployment process, before a node actually becomes operational in the infrastructure.

4.1 Profiling

During profiling we use a set of performance metrics observable through respective PMU counters to quantify the utilization of and pressure to microarchitectural components. Intel x86-64 PMUs can only track a limited number of performance counters at the same time (in the CPU families we use in this study, up to 8 per core). We consider 85 and 80 performance metrics for Skylake and Haswell, respectively. We include all the metrics used by Intel's Top-down Microarchitecture Analysis Method (TMAM) [13], as well as additional metrics that capture operational aspects of the system, such as the CPU operating temperature, CPU power consumption and DRAM power consumption. The systems, in the context of our work, operate under typical data center conditions [14], at an ambient temperature of 21-23°C. Evaluating the effects of undervolting for ambient temperatures that fall outside typical data center conditions is beyond the scope of this work. Still, during our experiments the CPU temperature varied between 27-64°C and 41-77°C for Skylake and Haswell, respectively.

To collect data for all respective metrics, we perform multiple executions for each benchmark configuration, and in each execution we record a subset of performance counters until all metrics are covered. We sample the counters every 100ms using Linux *Perf* tools [15]. As we discuss later, we reduce the number of metrics used for model training to a maximum of 8, so we can sample all corresponding performance counters in a single execution at xDVS deployment time.

4.2 Model Type

Our goal is to estimate a safe MSR_{offset} according to the resource utilization and pressure quantified by the performance metrics. The values obtained by offline characterization do not always exhibit a simple, monotonic behavior with respect to the obtained performance metrics. Consequently, linear regression models do not adequately capture the MSR_{offset} of the applications. Instead, to predict MSR_{offset} as a combination of the aforementioned metrics, we employ a machine learning ensemble technique, called Random Forest Regression (RFR) [12]. A random forest is a collection of regression decision trees, each used to independently predict a value based on an input vector. The model predicts by averaging over the predictions of all regression trees.

Due to the limitations of Intel PMU, at execution time we are limited to concurrently measuring up to 8 PMU events per core. In order for our model to be applicable in an online manner, the number of performance metrics/features needs to be reduced. This also avoids overfitting and decreases the runtime overhead of the model. To reduce the number of performance metrics, we rank their importance by estimating their mutual information

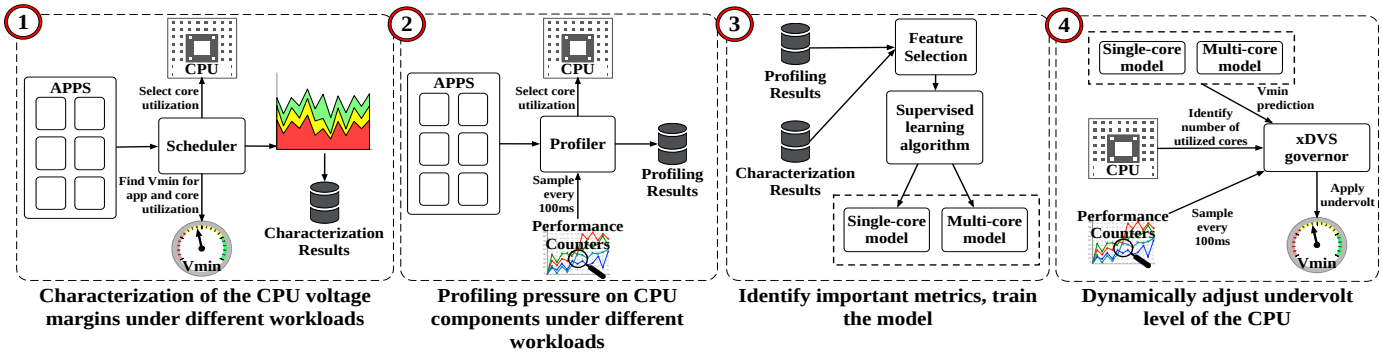


Fig. 5: Overview of our end-to-end approach.

Skylake	Haswell
UOPS_DISPATCHED.PORT_0:	*IDQ.ALL_DSB_CYCLES_4_UOPS:*
Uops dispatched for execution in port 0 (Port 0 is responsible for Int., FP, vector ALU, mult, div and branch operations).	Cycles in which Decode Stream Buffer (DSB) is delivering 4 Uops.
UOPS_DISPATCHED.PORT_4	*UOPS_EXECUTED.PORT_0:*
Uops dispatched for execution in port 4 (Port 4 is responsible for Store operations)	Uops dispatched for execution in port 0 (Port 0 is responsible for Int., FP,vector ALU, mult, div and branch operations).
UOPS_DISPATCHED.PORT_1:	*UOPS_EXECUTED.PORT_1:*
Uops dispatched for execution in port 1 (Port 1 is responsible for Int., FP and vector ALU operations).	Uops dispatched for execution in port 1 (Port 1 is responsible for Int., FP and vector ALU operations).
UOPS_DISPATCHED.PORT_5	*UOPS_EXECUTED.PORT_5:*
Uops dispatched for execution in port 5 (Port 5 is responsible for Int. and vector ALU operations).	Uops dispatched for execution in port 5 (Port 5 is responsible for Int., vector ALU operations).
UOPS_DISPATCHED.PORT_2	*UOPS_EXECUTED.PORT_2:*
Uops dispatched for execution in port 2 (Port 2 is responsible for Load operations).	Uops dispatched for execution in port 2 (Port 2 is responsible for Load operations).
EXE_ACTIVITY.PORTS	*UOPS_EXECUTED.PORT_6*
Cycles for which one uop began execution on any port, and the Reservation Station was not empty.	Uops dispatched for execution in port 6 (Port 6 is responsible for Int., and branch operations).
UOPS_EXECUTED.THREAD	*UOPS_EXECUTED.PORT_3*
Number of Uops executed by this hardware thread.	Uops dispatched for execution in port 3 (Port 3 is responsible for Load operations.)
MEM_UOPS.ALL_STORES:	*MEM_UOPS.ALL_STORES:*
Number of store operations retired.	Number of store operations retired.

TABLE 2: Most influential performance metrics for V_{min}, as ranked by the MI algorithm.

(MI) for the MSR_{offset} target variable. MI is an algorithm commonly used for feature selection in machine learning [16]. It ranks different features by assigning weights so that the higher the weight the more important the feature for modeling. The algorithm assigned the highest importance to the metrics listed in Table 2 in decreasing order of importance. Note that the highest-ranked metrics essentially characterize the instruction mix of the workload. The metrics that capture operational aspects of the system, such as the CPU operating temperature, CPU power consumption and DRAM power consumption, are ranked significantly lower by the MI-driven feature selection, than the ones listed in Table 2.

During the offline profiling phase, the performance metrics are collected through multiple executions for the same configuration of each experiment. After the MI ranking step, we repeat the experiments so that the selected metrics (Table 2) are collected during the same execution. These data are normalized to take values between 0 and 1. As a normalizer, we use the sum of all available slots during the sampling period, which is equal to the number of pipeline slots (4 in our architecture) multiplied by the number of clock cycles (the respective counter does not fall into the limitation of 8 PMU events per core).

Finally, recall that MSR_{offset}, besides being dependent on the workload, also depends on the resilience of the specific cores (inter-core variation) and degree of CPU utilization. Moreover, in a realistic scenario, the operating system may, independently of our mechanisms, modify the topology of active cores via thread migration. Our model should provide a safe setting irrespective of workload mapping to cores. To capture these variations, we construct two models for each CPU part. One model covers the case of single-core execution (single-core model), and is trained using the MSR_{offset} of the weakest core for each benchmark configuration. The other model covers the case where multiple cores are occupied (multi-core model), and is trained using the data of the benchmark configurations that use all cores of the CPU. So, in total, we build 12 different models (6 CPU parts, each having 2 models for single- and multi-core CPU utilization).

4.3 Model Training

Most applications exhibit different execution phases in terms of CPU resource pressure and performance characteristics. For example, Figure 6 shows the number of uops dispatched for execution in port 1 for the first 30 secs of execution for two applications with a single-phase (*swaptions, hmmer*) and one application with multiple phases (*facesim*). However, our offline characterization determines the single worst-case V_{min} across all execution phases of an application. This can negatively affect the effectiveness of training our model as it will overgeneralize, trying to correlate wildly varying performance counter patterns with the same V_{min}.

To provision for such cases, we bias the training input set to include mainly applications with few execution phases such as *hmmer* and *swaptions*. We first rank the 34 benchmarks according to the number of phases they exhibit, normalized to their execution time. Phase change detection is performed by monitoring large changes (more than 10%) on any of the Level-1 performance metrics of TMAM [13]. The smaller the number of phase changes, the higher the ranking of the application. Then, we select the top 90% (most stable) applications for training and validation. 90% of the selected applications are used for training and 10% for validation. The remaining 10% of applications (the ones with the largest number of phase changes) serve as the testing (evaluation) set. The validation set includes *bodytrack, freqmine, gcc*, and the testing set includes *facesim, zeusmp, fluidanimate, stress_ng*.

The training input set, which consists of samples (in 100ms sampling intervals) of the eight performance counters (Table 2) of the applications selected for training, is used to train the Random Forest Regression (RFR) with the goal of minimizing the Root Mean Square Error (RMSE) between the predicted V'_{dd} and the V_{min} determined via the offline characterization. Typically, RFR is defined by a predefined number of different simple estimators (the decision trees) and by the maximum depth of each decision tree. Using a large number of estimators and/or using deep trees for the prediction can both incur high-performance penalties, and result to overfitting. We detect overfitting by cross-validating the

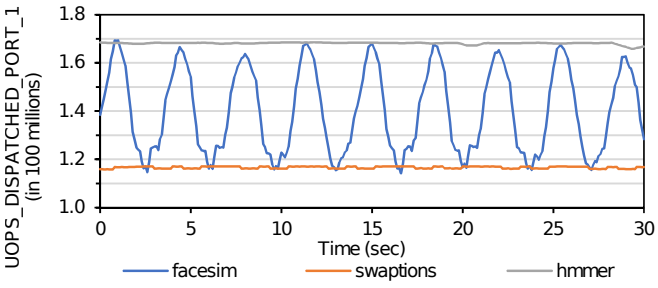


Fig. 6: The number of dispatched uops in port 1 during the execution time of an application.

models. In the end, the models that minimized the RMSE for both the training and validation set consist of only three estimators, with the maximum depth of each estimator being equal to three. The average RMSE is 7.01mV and 5.45mV for the Skylake and Haswell families, respectively.

Over- or under-prediction is a common side-effect of many modeling approaches. In our case, over-predicting the MSR_{offset} would result in reducing the supply voltage below V_{min}, leading to unreliable operation. Therefore, as a last step, we introduce a small *safety margin* to the estimated MSR_{offset} value. The safety margin controls the aggressiveness of our methodology. Using very small values would result to aggressive undervolting at the risk of reduced reliability, whereas too large safety margins would merely decrease the energy gains. For each individual model, the safety margin is set equal to the RMSE between the value that is predicted for the validation data, and the MSR_{offset} value that was observed during the offline characterization for the respective applications in the validation set. Equation 2 provides the final offset that is applied on the MSR registers of the CPU (where $\vec{X}$ denotes the input vector to the model).

$$MSR'_{offset}(\vec{X}) = MSR_{offset}(\vec{X}) - safetyMargin \qquad (2)$$

Figure 7 depicts MSR_{offset} predictions by our model (blue rectangles) at the granularity of 100msec, for the 7 applications in the validation and testing set, versus the static MSR_{offset} quantified by the offline characterization of each entire application. Dynamic MSR_{offset} prediction at a fine granularity enables, in many occasions, more aggressive voltage reduction (blue rectangles over the diagonal black line) compared with the MSR_{offset} identified for the entire application. In other occasions the model is unnecessarily conservative in its predictions (blue rectangles below the diagonal black line), missing opportunities for deeper voltage reduction. The orange diamonds correspond to the adjusted MSR_{offset} predictions after applying the safety margin. The vertical distance between each blue and orange cluster of markers corresponds to the voltage overhead of the safety margin. In Section 6 (Figure 10), we quantify the net effect of dynamic MSR_{offset} prediction (with the addition of the safety margin), against the use of a static MSR_{offset}, which would require a non-realistic characterization of all target applications

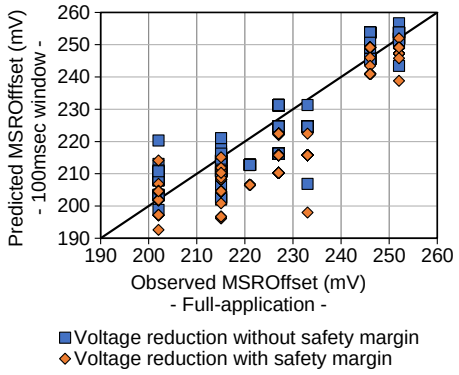


Fig. 7: MSR_{offset} predictions by our model, at the granularity of 100msec, versus the static MSR_{offset} of the characterization process. Each vertical cluster of points corresponds to a single application.

and combinations thereof. Also, in Section 7.2 we evaluate the robustness of system operation using our model.

5 Extended Dynamic Voltage Scaling

The model introduced in the previous section can be used online, to enable fine-grained undervolting at runtime, according to the resource pressure quantified by performance counters samples and the cores utilized by the current workload. To this end, we have implemented an extended dynamic voltage scaling governor (xDVS), which is invoked periodically. Upon invocation it feeds the model with the performance counter measurements collected during the previous interval, and uses the suggested MSR_{offset} to derive a less-than-nominal but still safe supply voltage V'_{dd}.

The xDVS governor is implemented as a Finite State Machine (FSM), depicted in Figure 8. The V'_{dd} selected for the next interval depends on the prediction of the model, the number of active cores and the current state of the governor. Below we describe the states and the logic of xDVS:

Back-Off: In this state, the measurements collected during the previous interval are not considered as representative for the workload during the next interval and, therefore, are not valid input for the model. This is the case, for example, when a core starts executing a new process/thread, or after a long idle period that allowed the operating system to place that core in a higher C-state (lower power mode). In this state, xDVS does not invoke the model and applies the nominal V_{dd}.

Step-Up: This state provides a smooth transition between the *Back-Off* and the *Stable* states. When in this state, the governor invokes the model to perform predictions in a way similar to the *Stable* state, but if the model suggests a large reduction to the supply voltage, this is applied gradually, in smaller steps of 5 mV. This filters abrupt and potentially risky – in terms of reliability, should the behavior of the workload change again – voltage reductions. In the experimental evaluation presented in Section 7.2, we confirmed that both the step-up state and the safety margin (Section 4.3), lead to a robust operation of xDVS and, consequently, achieve system reliability.

Stable: The governor collects performance counter values for each core and invokes the single-core or multi-core model depending on whether only one or more cores are currently active. Then, based on the suggested MSR_{offset}, the governor applies the new V'_{dd}.

The xDVS governor monitors which cores are active, by observing the percentage of time the system was in the $C0$ state ($C0$ residency) of each core. We consider a core as active when

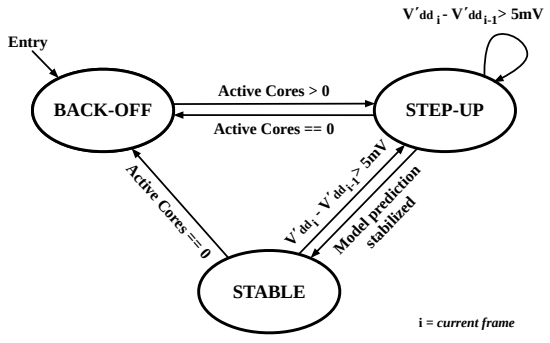


Fig. 8: FSM diagram of the xDVS governor.

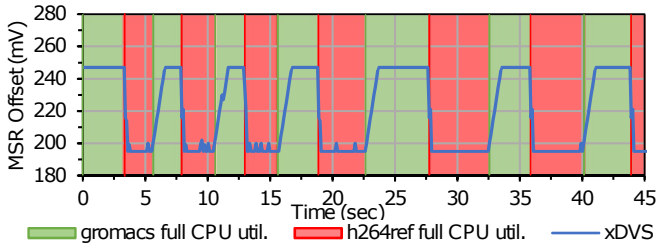


Fig. 9: MSR_{offset} applied by xDVS on Skylake 2, for sudden transitions between workloads with different margins.

$C0$ residency goes above 70%, and a core is classified as inactive when its $C0$ residency drops below 50%. We avoid thresholds close to 100% or 0% as this would result to unnecessarily high transition sensitivity (e.g., a core would be considered as active when merely moving the mouse of a desktop).

When all cores are considered as inactive, the xDVS governor enters/stays in the *Back-Off* state in order to provide a setup time to new workloads. When there is at least one active core across two consecutive sampling intervals, xDVS transitions to the *Step-Up* state. It remains in this state until the supply voltage is gradually reduced to the level suggested by the model, and once this level is reached the governor enters the *Stable* state.

Whenever the model suggests a large increase to the supply voltage, the increase is implemented immediately (without any stepping), irrespective of the state of xDVS as this is considered an emergency and any delay could compromise the reliability of the system. Also, increasing the supply voltage does not introduce any risk if the workload suddenly changes; in the worst case, an energy saving opportunity will be missed.

Figure 9 illustrates the level of undervolting applied by xDVS on Skylake 2 for an indicative scenario with two abruptly alternating workloads with significantly different margins, namely gromacs and h264ref. We execute four instances of each of the applications to fully utilize the CPU. As discussed in Section 3, the CPU supply voltage can be reduced by 257mV and 198mV for gromacs and h264ref respectively. In every transition from the workload with a high tolerance (gromacs) to the workload with a lower tolerance (h264ref), the level of undervolting drops immediately to increase the CPU supply voltage. In the reverse transitions, the undervolting level gradually increases (CPU voltage decreases) until it reaches the level that is suggested by the model.

6 Experimental Evaluation

In this section we evaluate the ability of the xDVS governor to dynamically identify voltage margins based on our prediction model and drive the CPU to a more energy efficient state. We quantify the resulting energy gains using the benchmarks in the test set (*stress_ng, zeusmp, fluidanimate,facesim*), which have not been used during model training and validation and we compare the energy gains of xDVS against the Intel *P-state* DVFS governor.

In addition, we evaluate xDVS with 4 larger-scale applications, namely as Gem5 [17], a CPU miner [18], the compilation of the Linux Kernel [19] and Polybench [20]. More specifically, Gem5 simulates an ARM processor using the system emulation mode. During the simulation we execute a variety of simple micro kernels such as Integer and Floating Point Matrix Multiplication, Sorting algorithms, and Combinatoric problem solving kernels. The CPU miner employs five different hashing algorithms (Bitcore, Sha256d, Xevan, Timetravel and Cryptonight [21]), all used to perform mining for different cryptocurrencies such as Litecoin and Bitcoin. Finally we use multiple solvers and stencils included in the Polybench suite such as Alternating Direction Implicit (ADI), Jacobi, LU factorization, GramSchmidt process, GaussSeidel. Each one of these 4 larger-scale applications is executed for approximately 1 hour. Figure 10 shows the average MSR_{offset} applied by xDVS when applications are executed on all cores, or on the weakest core. For the large-scale applications, in which there is no offline characterization, we present the average dynamic MSR_{offset} across all single core executions.

The MSR_{offset} applied by xDVS includes the extra safety margin, thus we expect it to be on average more pessimistic than the MSR_{offset} identified by offline characterization (as shown in Figure 10). The voltage applied by xDVS (including the safety margin) is on average 9.3mV and 8.2mV higher that the one identified by offline characterization for the Skylake and Haswell microarchitectures respectively. Despite the safety margin, there are cases in which xDVS successfully identifies application phases and adjusts the supply voltage to lower values than those identified in the offline characterization, without compromising system reliability. Note in Figure 10 that xDVS identifies wider static and dynamic (phase-dependent) voltage margins for the Skylake family, which is compatible with the findings of the offline characterization presented in Figure 3c.

The left side of Figure 11 shows the dynamically applied MSR_{offset} when four benchmarks are scheduled for consecutive execution on the same core of Skylake 2. Figure 11 justifies our

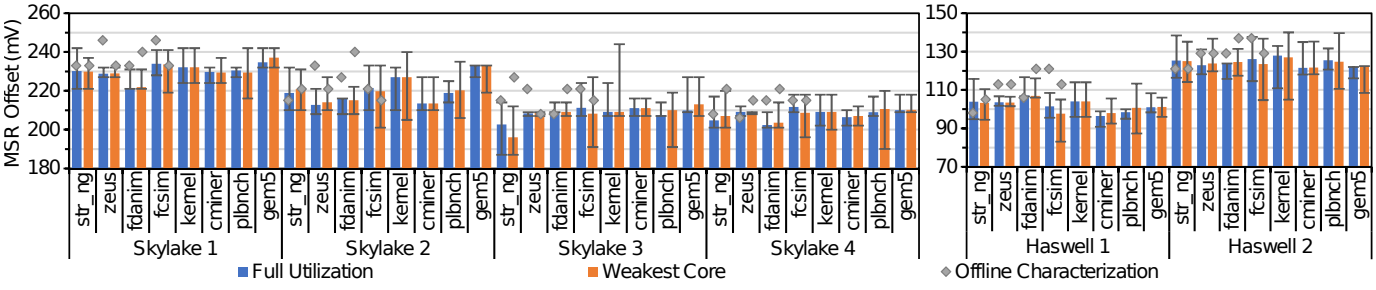


Fig. 10: The bars show the average dynamic MSR_{offset} applied by xDVS, for Skylake (left) and Haswell (right) workstations. The min-max bars represent the minimum and the maximum MSR_{offset} applied by xDVS. The gray diamond represents the MSR_{offset} as identified by offline characterization at the granularity of the whole application.

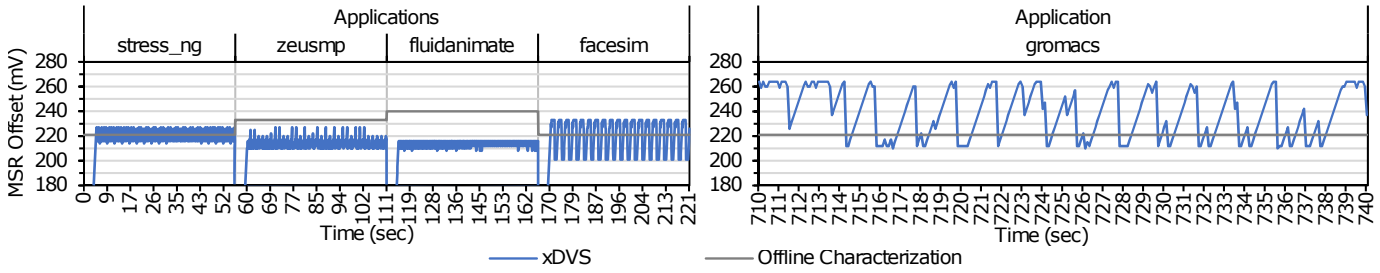


Fig. 11: Timeline showing the MSR_{offset} for consecutive single core executions of four applications on Skylake 2 (left), and a snapshot of single core execution for *gromacs* application on Skylake 4 (right).

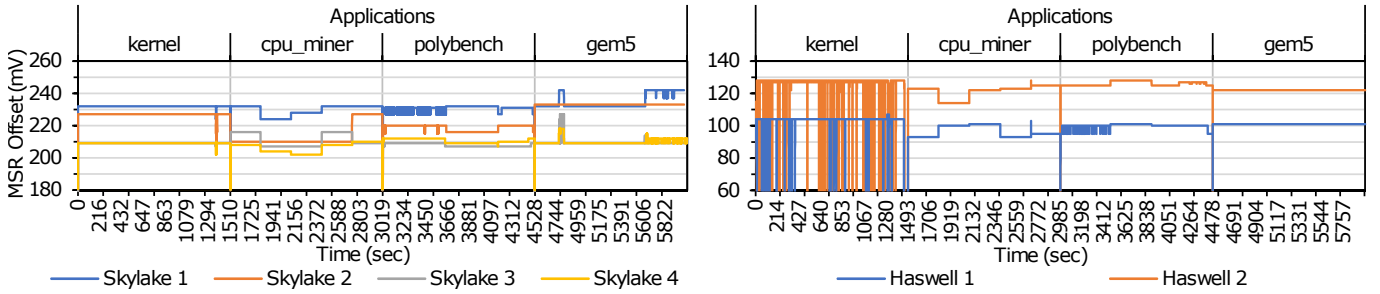


Fig. 12: The timeline showing the MSR_{offset} applied by xDVS, while executing the large applications in full system utilization for Skylake (left) and Haswell (right) workstations.

decision to include mainly applications with few execution phases in the training set of our models (Section 4.3). xDVS is able to capture the dynamic nature of the applications and adjust the CPU operating voltage accordingly. More specifically, in the *facesim* case (an iterative application that executes 3 separate kernels per iteration) the MSR_{offset} varies between 201mV and 233mV while the model captures the phases of the application. A similar behavior is observed for *gromacs* (Figure 11 on the right), in which xDVS captures the periodic phase changes and drives undervolting even more aggressively than what static MSR_{offset} dictates. For brevity, we do not include graphs of the remaining CPUs as they present similar trends.

Figure 12 shows the MSR_{offset} timeline when the four larger-scale applications are scheduled for execution on each workstation. The graphs reveal relatively large intra-family margin variations, as well as variations due to different workload characteristics. Note that the xDVS governor is able to capture the different algorithms consecutively used by the CPU miner application. Although the variation between different predictions is small, as observed in the offline characterization, the margin between correct execution and failure is small (less than 5mV). Therefore, even minimal adjustments to the supply voltage can have an noticeable impact on system reliability.

Note also the steep MSR_{offset} drops when the OS schedules a new application, owing to our xDVS governor to transition to the *Back-Off* state. Interestingly, when running the Linux kernel compilation on the Haswell processors, the governor transitions frequently to the *Back-Off* state. The Haswell PCs are equipped with a slow Hard Drive Disk (HDD). When compiling, the large source files of the kernel and object files are read from and written to the HDD, resulting to the *C*0 residency of each core drop below 50% (since most of the time cores wait for I/O operations). This makes xDVS to transition to the *Back-Off* state. This effect is not observed on the Skylake workstations, as they are equipped with fast Solid State Drive (SSD).

Figure 13 shows that the xDVS governor achieves 29.59% and 21.93% average CPU energy gains and can reach up to 42.68% and 34.37% for the Skylake and Haswell processors respectively, compared with the Intel *P-state* governor. Higher gains are – as expected – obtained when cores of the CPU are highly utilized. For the energy consumption monitoring we used Linux *Perf* tool.

Note that when xDVS is enabled, Intel's Turbo Boost technology is disabled and the operating frequency is pinned to

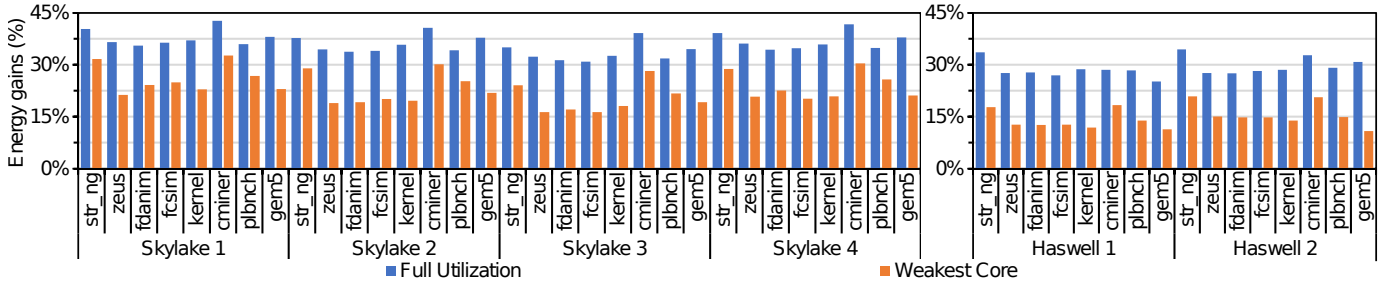


Fig. 13: Energy gains of xDVS when compared with Intel P-state governor for Skylake (left) and Haswell (right) CPUs.

the CPU maximum nominal frequency. This penalty, due to lower operating CPU frequency, translates to application execution slowdown and is on average 8.73% and 5.59% for the Skylake and Haswell, respectively. However, the actual performance overhead of the xDVS governor itself is minimal. When the governor is active the prediction requires only 160*ns*, while changing the new MSR_{offset} requires 155us. On average, the performance overhead is equal to merely 0.04% of execution time when xDVS operates at an 100ms interval. If the interval is set to 10ms (lowest supported sampling interval of performance counters through *Perf* API), the overhead remains at 0.04% for single core utilization, but increases up to 3.5% on average at full CPU utilization. Even in this case, xDVS still provides significant energy gains compared to the Intel P-State CPU governor.

7 System reliability and fault-tolerance mechanisms

Our methodology is designed to predict and apply subnominal CPU operating voltages when executing conventional workloads on multi-core CPUs. Although we have not observed any crashes and SDCs in our experiments, voltage underscaling inevitably comes at the cost of increased probability of system failure compared to nominal CPU operation (where malfunctions occur too, yet with lower probability). In this section, we discuss the prospect of using hardware-level mechanisms to increase the robustness of the system. We also discuss a long-running campaign that was conducted in order to confirm the robustness of xDVS. Finally, we investigate the combination of xDVS with a fault tolerance mechanism, such as checkpointing, in large-scale deployment scenarios.

7.1 Hardware Mechanisms

Our methodology depends on the behavior of the current workload during a sampling period, as this is quantified by the monitored performance counters. Our experimental evaluation demonstrates that, for typical workloads, the performance counters can be successfully used as indicators to reduce the CPU supply voltage. However, the sampling of the performance counters and the supply voltage adjustment can be as frequent as every 10ms, which is too coarse to capture events caused by specific workloads, namely voltage droop viruses, that produce large voltage fluctuations and expose the susceptibilities of the power delivery network of microprocessors [22], [23], [24]. Typically, such workloads consist of a periodic sequence of high activity and low activity instruction regions at a frequency equal to the resonance frequency of the power delivery network ($50-200MHz$ in modern CPUs). This pattern simulates the invocation of high CPU activity workloads immediately after a period of very low activity, which causes large di/dt swings and large voltage droops.

The problem of voltage droop detection and mitigation in modern CPU and GPUs is (and should be) addressed at the hardware level with specialized circuitry [25], [26]. A high-speed droop detection mechanism continuously monitors the power grid causing a rapid charge shunt to the V_{dd} rail to correct a voltage emergency within a few clock cycles. Ideally, these mechanisms could also inform the software stack when voltage emergencies are detected, acting as a trigger towards more conservative undervolting. In general, stronger error protection and error recovery mechanisms to correct timing violations are very helpful as extra protection during aggressive voltage scaling. Note also that voltage droop viruses are difficult to generate and may be different across not only different microarchitectures but also across different CPU parts. Moreover, on top of a realistic software stack, the background operating system activity (jitter) smooths out large voltage swings caused by such viruses [27].

7.2 Robustness of xDVS

To assess the robustness of the proposed approach, we performed a long experimental campaign on 16 identical Skylake nodes. During this campaign, xDVS was enabled on all nodes while executing randomly selected combinations of applications from our full set of benchmarks.

The experiment was concluded after a total of 8832 device-hours, 552 hours (23 days) per system, without any observed failures such as crashes, detected errors or silent data corruptions (SDCs). The lack of errors during this long period of continuous system operation is an indication of the robustness of our approach. Also, we use this result to analytically estimate the expected Mean Time Between Failures ($MTBF$), as discussed next.

7.3 xDVS and Checkpointing

Even when performed in an educated and careful manner, CPU undervolting may affect the resilience of the CPU against errors. This makes the system more prone to failures and necessitates the deployment of a fault tolerance mechanism. In the following we investigate whether the proposed approach remains attractive despite the performance and energy overheads of such mechanisms. All the equations used in our analysis are presented in detail in the Appendix A.

In our study, we pessimistically assume a naive coordinated checkpointing scheme that is performed periodically. More specifically, we use Equation 3 and Equation 4 of the analysis in [28] to estimate the optimal checkpointing period (T_{opt}) and the performance overhead (WASTE) caused by checkpointing. A crucial factor in the estimation of T_{opt} and WASTE is the platform $MTBF_{plat}$ (by platform we mean a system of N nodes that are used to execute a specific workload, assuming that nodes exhibit independent failures). We calculate $MTBF_{plat}$ based on Equation 5 of the analysis in [28]. Moreover, prior studies show that the MTBF of a single node ($MTBF_{node}$) in large-scale facilities can be captured by a Weibull distribution with decreasing hazard rate (shape parameter $\kappa = 0.7$) [28], [29], [30], [31]. Note that Weibull with decreased hazard rate considers infant mortality phenomena, which is consistent with our own observations during the characterization phase that the selection of overly aggressive CPU undervolting (below V_{min}) leads to failures almost instantly.

As we did not observe any failures or SDCs during the long-running experimental campaign discussed in Section 7.2, we pessimistically assume that one of the 16 machines would experience a transient error right after the end of our experimental campaign. Given this assumption, we use Equation 6 and calculate $\text{MTBF}_{node} = 1460$ days. Furthermore, given the limited number of nodes tested (16 machines), we apply the chi-squared distribution to estimate the $MTBF_{node}$ under a specific confidence level, based on Equation 7 of the analysis in [32], [33]. Having not observed any errors during the

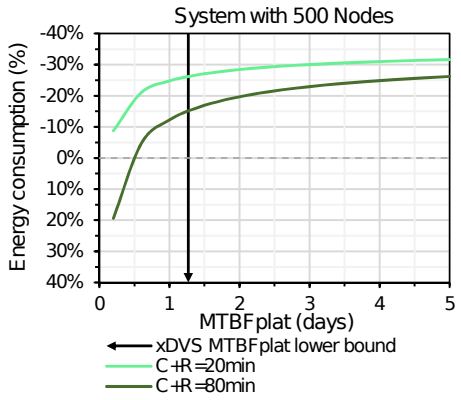


Fig. 14: Energy reduction of xDVS with checkpointing vs. an ideal system at nominal settings without failures and no checkpointing. The vertical black line with the arrow denotes the $MTBF_{plat}$ that was analytically derived using as input the duration of the error-free long running campaign.

testing, we use a chi-squared distribution with 2 degrees of freedom ($n = 0$) [32], [33]. As a result, $MTBF_{node}$ for a confidence level of 90% is estimated to $MTBF(0.9)_{node} \geq 634.79$ days. Note that it is common for industry assessments to provide $MTBF$ at a much lower confidence level of 60% [34].

Figure 14 illustrates the energy benefits, for different $MTBF_{plat}$ values, to be expected at reduced voltage margins with checkpointing vs. an *ideal* execution at nominal CPU settings where we assume that nodes *never* fail and *no* checkpointing is performed *at all* (in reality, nodes do fail and checkpointing schemes are necessary even when the CPU operates at nominal settings). In this case of a node failure (when operating at extended margins), we assume a constant node downtime overhead (D) at 2 minutes and present results for two cases where the checkpointing and recovery overhead ($C+R$) is 20 and 80 minutes, respectively. Based on the $MTBF(0.9)_{node}$ of our analysis, we provide the lowest (most pessimistic) bound of the MTBF_{plat} for a system with 500 nodes. This bound is depicted in Figure 14 as a black vertical line. Interestingly, operation at reduced margins remains beneficial with energy gains at 15% even for costly $C+R$ overhead at 80 minutes (dark green line). For lower $C+R$ overhead at 20 minutes (light green line), energy gains reach up to 26%.

8 RELATED WORK

In contrast to our work, all previous efforts in predicting voltage margins using performance counters only report theoretical energy gains, without deploying their model. They rely only on theoretical results which, if not validated in real hardware, could result in unreliable operation and even system crashes. Moreover, they do not project the potential energy gains, as this work does, in large scale deployments, for which a fault tolerance mechanism – such as checkpointing – is necessary. Our FSM based governor uses performance counter values to reduce the voltage margin of the system at run time on real hardware for unseen workloads which consist of several different applications executing concurrently.

In particular in [9], [35] authors present an automated system-level analysis on multi-core CPUs based on the ARMv8 64-bit architecture, when pushed to operate in scaled voltage conditions. In particular in [9], due to the manifestation of SDCs before system crashes, the authors propose a severity function that can predict safe, SDC-free undervolt levels for each core of the processor. Based on this function and the corresponding core V_{min} that resulted from the offline characterization, they produce a linear regression model that predicts the V_{min} of a core for a single-instance workload. Their model is trained and evaluated using only single core executions. In contrast to their work, we use multi-instance/threaded workloads and demonstrate their importance, as multi-instance/threaded workloads usually result to more conservative V_{min}. Moreover, [9] states that V_{min} prediction is uncorrelated to performance counters and in combination with limited dynamic variation of V_{min} in ARMv8 a naive prediction of using average values is sufficient. In a more recent paper [36], the same authors present a comprehensive statistical analysis for the same platform that improves on prediction on dynamic variations. Note also that we identify margin deviations between off-the-shelf, commercially available parts, and not parts from extreme corner nodes (TTT, TFF and TSS in [9]).

In [37] the authors characterize the voltage margins of two x86-64 microprocessors (Sandy-Bridge-E and Haswell) for a subset of the SPEC CPU2006 benchmark suite. Similar to [9], they do not consider the implications induced by multi-instance/threaded executions. We show that multi-instance/threaded executions significantly limit the depiction of the whole voltage margins spectrum. Also, in our work we quantify the voltage margin deviations between off-the-shelf parts that belong to the same CPU family and eventually deploy a governor that exploits effectively the voltage margins.

The heuristics presented in [10] and [38] that dynamically reduce voltage margins while always preserving safe operation, are based on the error correction ECC hardware built on modern processors such as the server-class Intel Itanium 9560. A key observation there is that as V_{dd} is lowered, ECC correctable errors appear before uncorrectable errors (SDCs and CPU crashes). The rate of ECC correctable errors is used as an indicator on how to adjust the V_{dd} voltage. In our work, errors reported by the ECC mechanism appear very rarely, and they are always accompanied by immediate CPU crashes. We do not rely on ECC, but rather predict a safe supply voltage using a selected set of performance counters as estimators. Eventually, the methodology we propose is generic enough that can be applied to any processor, that provides the ability to manipulate the supply voltage, by measuring performance counters.

Authors in [39] exploit the voltage margins of a server-class 8-core Power7+ processor. In contrast to our work, this approach utilizes specialized hardware mechanisms such as critical path monitor sensors and digital phase locked loops, to detect and exploit the margins. As the number of utilized cores increases by the executing workload, the larger the IR drops are across the chip. The power gains obtained by their adaptive margin scheme begin to diminish as the executing workload utilizes more CPU cores. On the contrary, the CPU families under investigation in our work, exhibit the exact opposite trend, where greater power improvements are observed on multi-threaded/instance workloads. More importantly, our xDVS governor is able to exploit voltage margins regardless the CPU core utilization scheme.

A study of the voltage margins on several Kepler and Fermi GPUs is presented in [40]. The authors first characterize the impact of the process, temperature and voltage variation on V_{min},

and then predict safe values of V_{min} by deploying a linear regression and a neural network model. They show that high energy margins can be achieved by shaving conservative guardbands in modern GPUs. Our work targets CPU architectures which are significantly more complex than GPUs. Moreover, typically CPUs - contrary to GPUs - serve volatile workloads, with diverse characteristics, consisting of mixed user and OS jobs. Our model is able to provide accurate voltage margins predictions for workloads which consist of a mixture of different benchmarks.

Based on microarchitectural events (such as branch mispredictions and cache misses) that flush and stall the CPU pipeline and cause large voltage droops, the authors in [11] they propose a voltage emergency predictor that learns the signatures of such voltage emergencies and triggers a CPU throttling mechanism (e.g. increase voltage or decrease frequency) to prevent their recurrence. This work is based on CPU modeling on the Simplescalar simulator. In a subsequent paper, the authors measure run time voltage emergencies on a Core 2 Duo processor and attribute them to pipeline stalls and flushes [27]. Based on these experimental observations, they propose that a mechanism that uses more aggressive margins and a recovery (check-point based) scheme may be better than a fail-safe static margin. Moreover, they propose a program scheduling mechanism so that the combined voltage droop is canceled out as much as possible. Unlike our work, the previous papers do not resort to undervolting to reduce voltage margins but instead they describe techniques and provide solid design guidelines that could be exploited by the supplementary mechanism we describe in Section 7.

A number of proposed techniques present design approaches at the circuit or micro-architectural level that trade reliability for lower voltage, by attempting to reduce voltage down to the point that produces maximum allowable errors without causing catastrophic failures [41]. Several approaches propose methods which ensure correct operation of caches under undervolted conditions at the microarchitectural level [42], [43], [44]. Architectural techniques are presented to eliminate data corruption, and by extension enable cache operation at scaled voltage settings. In our work we are only interested on the behavior of the whole CPU, and not of any specific component. The Razor processor is designed with builtin support for dynamic detection and correction of timing failures of the critical paths [45]. EVAL is a framework for dynamic adaptation of supply voltage, processor frequency and body bias using a machine learning algorithm [46]. Similar ideas include dynamic pipeline adaptation transferring the time slack of faster pipeline stages to the slower ones (ReCycle) [47], and using variable latency techniques to mitigate the impact of variations on the register file and execution units in a microprocessor [48].

Furthermore, there are many approaches that employ simulations and modeling techniques to provide design guidelines for future hardware. Authors in [49] propose a multi-core processor which can scale its resources and the number of operating cores to lower than the total CPU cores on chip to meet certain power constrains. Several approaches [50], [51], [52] employ regression analysis to map certain performance counters to micro-architectural events and power-delivery estimation. [53] explores ECC protection mechanisms to enable low-power caches through a detailed SRAM failure modeling. [54] introduces a workload dependent technique to identify the paths excited by individual applications on ultra-low-power microprocessors and reduce voltage to a level that meets timing of those paths (instead of all paths).

9 Conclusions

This paper introduces xDVS, a voltage governor which employs a prediction model of the V_{min} for Intel's Skylake and Haswell microarchitectures and is used to continuously adjust the supply voltage and drive a multicore processor to a more energy efficient, yet still safe zone of operation. The model predicts V_{min} using the CPU performance monitoring counters and is based on an offline V_{min} characterization that revealed wide voltage margins, which vary across different microarchitectures, different chip parts of the same microarchitecture, and across different workloads. In this context, we show the importance of using a diverse set of single- and multi-instance/threaded benchmarks for part characterization.

The experimental evaluation of xDVS on real hardware, executing a wide range of benchmarks and larger-scale applications shows significant energy savings, up to 42.68% and 34.37% over the standard Intel *P-state* DVFS governor for parts of the Skylake and Haswell families, respectively.

Our method enables significant power savings for a wide range of conventional workloads, without requiring additional hardware support, or compromising system reliability. However, hardware support for the detection and mitigation of voltage emergencies is useful to safeguard against aggressive (and potentially malicious) codes, such as voltage droop viruses. Voltage emergencies should not be treated differently from other emergencies (such as memory access bound violations, execution of illegal instructions, use of illegal operands etc.) which can undermine the safe operation of a computing system, and are traditionally expected to be identified by hardware. In addition, we provide a risk assessment analysis of CPUs operating at reduced voltage which estimates the MTBF of Xeon E3 CPUs when xDVS is enabled. This analysis indicates that our proposed methodology results to energy gains even at larger scales, when checkpointing and recovery is deployed.

Acknowledgments

This work has been funded by the H2020 Framework Program of the European Union through the UniServer Project (Grant Agreement 688540) http://www.uniserver2020.eu. This work was partially performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DEAC52-07NA2734 (LLNL-JRNL-809714).

References

[1] S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "A Self-Tuning DVS Processor Using Delay-Error Detection and Correction," *IEEE Journal of Solid-State Circuits*, vol. 41, 2006.

[2] E. A. Burton, G. Schrom, F. Paillet, J. Douglas, W. J. Lambert, K. Radhakrishnan, and M. J. Hill, "FIVR—Fully Integrated Voltage Regulators on 4th Generation Intel® Core ™ SoCs," in *Proceedings of Twenty-Ninth Annual international conference on Applied Power Electronics Conference and Exposition (APEC)*. IEEE, 2014.

[3] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Comput. Archit. News*, vol. 34, Sep. 2006.

[4] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques (PACT)*. ACM, 2008.

[5] A. P. Jack J. Dongarra, Piotr Luszczek, "The LINPACK Benchmark: Past, Present and Future," *Journal of Concurrency and Computation Practice and Experience*, vol. 10, August 2003.

[6] G. Woltman and S. Kurowski, "GIMPS, The Great Internet Mersenne Prime Search," 2008. [Online]. Available: https://www.mersenne.org/

[7] D. Hackenberg, R. Oldenburg, D. Molka, and R. Schne, "Introducing FIRESTARTER: A Processor Stress Test Utility," in *Proceedings of the 4th Conference on International Green Computing (IGC)*, 2013.

[8] "Stress-NG," 2017. [Online]. Available: http://kernel.ubuntu.com/~cking/stress-ng/

[9] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, P. Lawthers, and S. Das, "Harnessing Voltage Margins for Energy Efficiency in Multicore CPUs," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017.

[10] A. Bacha and R. Teodorescu, "Using ECC Feedback to Guide Voltage Speculation in Low-Voltage Processors," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2014.

[11] V. J. Reddi, M. S. Gupta, G. Holloway, G.-Y. Wei, M. D. Smith, and D. Brooks, "Voltage Emergency Prediction: Using Signatures to Reduce Operating Margins," in *Proceedings of the 15th International Symposium on High Performance Computer Architecture (HPCA)*, 2009.

[12] A. Liaw and M. Wiener, "Classification and Regression by RandomForest," *Journal of R News*, vol. 2, 2002.

[13] J. Marusarz, S. Cepeda, and A. Yasin, "How to Tune Applications Using a Top-Down Characterization of Microarchitectural Issues," Technical report, Intel, Tech. Rep., 2013.

[14] AT Committee, "Data Center Power Equipment Thermal Guidelines and Best Practices Whitepaper," June 2016, https://tc0909.ashraetcs.org/documents/ASHRAE_TC0909_Power_White_Paper_22_June_2016_REVISED.pdf.

[15] "Perf: Linux Profiling with Performance Counters." 2017. [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page

[16] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating Mutual Information," *Journal of Physical Review E*, vol. 69, 2004.

[17] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 Simulator," *Journal of ACM SIGARCH Computer Architecture News*, vol. 39, 2011.

[18] A. Jeff Garzik, "CPU-Miner." [Online]. Available: https://github.com/tpruvot/cpuminer-multi

[19] "Linux Kernel." 2017. [Online]. Available: https://www.kernel.org/

[20] L.-N. Pouchet, "PolyBench/C 3.2." [Online]. Available: https://sourceforge.net/projects/polybench/

[21] U. Mukhopadhyay, A. Skjellum, O. Hambolu, J. Oakley, L. Yu, and R. Brooks, "A Brief Survey of Cryptocurrency Systems," in *Proceedings of the 14th Annual Conference on Privacy, Security and Trust (PST)*, Dec 2016.

[22] A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management," in *Proceedings of the 26th Security Symposium (USENIX Security)*, 2017.

[23] Y. Kim and L. K. John, "Automated Di/Dt Stressmark Generation for Microprocessor Power Delivery Networks," in *Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and Design (ISLPED)*, 2011.

[24] Y. Kim, L. K. John, S. Pant, S. Manne, M. Schulte, W. L. Bircher, and M. S. S. Govindan, "AUDIT: Stress Testing the Automatic Way," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2012.

[25] T. Singh, A. Schaefer, S. Rangarajan, D. John, C. Henrion, R. Schreiber, M. Rodriguez, S. Kosonocky, S. Naffziger, and A. Novak, "Zen: An Energy-Efficient High-Performance $\times$ 86 Core," *IEEE Journal of Solid-State Circuits*, vol. 53, 2018.

[26] S. T. Kim and Y. C. Shih and K. Mazumdar and R. Jain and J. F. Ryan and C. Tokunaga and C. Augustine and J. P. Kulkarni and K. Ravichandran and J. W. Tschanz and M. M. Khellah and V. De, "Enabling Wide Autonomous DVFS in a 22 nm Graphics Execution Core Using a Digitally Controlled Fully Integrated Voltage Regulator," *IEEE Journal of Solid-State Circuits*, vol. 51, Jan 2016.

[27] V. J. Reddi, S. Kanev, W. Kim, S. Campanoni, M. D. Smith, G. Y. Wei, and D. Brooks, "Voltage Smoothing: Characterizing and Mitigating Voltage Noise in Production Processors via Software-Guided Thread Scheduling," in *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2010.

[28] J. Dongarra, T. Herault, and Y. Robert, "Fault Tolerance Techniques for High-Performance Computing," in *Fault-Tolerance Techniques for High-Performance Computing*. Springer, 2015, pp. 3–85.

[29] T. J. Hacker, F. Romero, and C. D. Carothers, "An Analysis of Clustered Failures on Large Supercomputing Systems," *Journal of Parallel and Distributed Computing*, vol. 69, 2009.

[30] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S. L. Scott, "An Optimal Checkpoint/Restart Model for a Large Scale High Performance Computing System," in *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*. IEEE, 2008.

[31] B. Schroeder and G. A. Gibson, "A Large-Scale Study of Failures in High-Performance Computing Systems," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, June 2006.

[32] MIL-HDBK-338, "Military Handbook. Electronic Reliability Design Handbook," 1998. [Online]. Available: https://www.weibull.com/knowledge/milhdbk.htm

[33] I. Bazovsky, *Reliability Theory and Practice*. Courier Corporation, 2004.

[34] Intel, "Reliability report," 1H 2017, https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/rr/rr.pdf.

[35] G. Papadimitriou, A. Chatzidimitriou, and D. Gizopoulos, "Adaptive Voltage/Frequency Scaling and Core Allocation for Balanced Energy and Performance on Multicore CPUs," in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019.

[36] M. Kaliorakis, A. Chatzidimitriou, G. Papadimitriou, and D. Gizopoulos, "Statistical Analysis of Multicore CPUs Operation in Scaled Voltage Conditions," *Journal of IEEE Computer Architecture Letters*, vol. 17, 2018.

[37] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, C. Magdalinos, and D. Gizopoulos, "Voltage Margins Identification on Commercial x86-64 Multicore Microprocessors," in *Proceedings of the 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2017.

[38] A. Bacha and R. Teodorescu, "Dynamic Reduction of Voltage Margins by Leveraging On-chip ECC in Itanium II Processors," *Journal of SIGARCH Computer Architecture News*, vol. 41.

[39] Y. Zu, C. R. Lefurgy, J. Leng, M. Halpern, M. S. Floyd, and V. J. Reddi, "Adaptive Guardband Scheduling to Improve System-Level Efficiency of the POWER7," in *Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015.

[40] J. Leng, A. Buyuktosunoglu, R. Bertran, P. Bose, and V. J. Reddi, "Safe Limits on Voltage Reduction Efficiency in GPUs: A Direct Measurement Approach," in *Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015.

[41] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Designing a Processor from the Ground up to Allow Voltage/Reliability Tradeoffs," in *Proceedings of the 16th International Conference on High-Performance Computer Architecture (HPCA)*, 2010.

[42] H. Duwe, X. Jian, D. Petrisko, and R. Kumar, "Rescuing Uncorrectable Fault Patterns in On-Chip Memories through Error Pattern Transformation," in *Proceedings of the 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[43] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu, "Improving Cache Lifetime Reliability at Ultra-low Voltages," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009.

[44] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S. L. Lu, "Trading off Cache Capacity for Reliability to Enable Low Voltage Operation," in *Proceedings of the 35th International Symposium on Computer Architecture (ISCA)*, 2008.

[45] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: a Low-Power Pipeline based on Circuit-Level Timing Speculation," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2003.

[46] S. R. Sarangi, B. Greskamp, A. Tiwari, and J. Torrellas, "EVAL: Utilizing Processors with Variation-induced Timing Errors," in *Proceedings of the 41st Annual International Symposium on Microarchitecture (MICRO)*, 2008.

[47] A. Tiwari, S. R. Sarangi, and J. Torrellas, "ReCycle: Pipeline Adaptation to Tolerate Process Variation," in *Proceedings of the 34th International Symposium on Computer Architecture (ISCA)*, 2007.

[48] X. Liang and D. M. Brooks, "Mitigating the Impact of Process Variations on Processor Register Files and Execution Units," in *Proceedings of the*

39th Annual International Symposium on Microarchitecture (MICRO), 2006.
[49] N. Kim, "Resource and Core Scaling for Improving Performance of Power-Constrained Multicore Processors," Mar. 28 2017, US Patent 9,606,842. [Online]. Available: https://www.google.com/patents/US9606842
[50] P. J. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, "Construction and Use of Linear Regression Models for Processor Performance Analysis," in *Proceedings of the 12th International Symposium on High-Performance Computer Architecture (HPCA)*, 2006.
[51] W. Jia, K. A. Shaw, and M. Martonosi, "Stargazer: Automated Regression-Based GPU Design Space Exploration," in *Proceedings of the International Symposium on Performance Analysis of Systems Software (ISPASS)*, April 2012.
[52] B. C. Lee and D. M. Brooks, "Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction," *SIGPLAN Not.*, vol. 41, Oct. 2006.
[53] J. Kim, M. McCartney, K. Mai, and B. Falsafi, "Modeling SRAM Failure Rates to enable Fast, Dense, Low-Power Caches," in *Proceedings of the IEEE Workshop on Silicon Errors in Logic - System Effects (SELSE)*, 2009.
[54] H. Cherupalli, R. Kumar, and J. Sartori, "Exploiting Dynamic Timing Slack for Energy Efficiency in Ultra-Low-Power Embedded Systems," in *Proceedings of the ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016.

APPENDIX

For the estimation of the optimal checkpointing period (T_{opt}) and checkpointing overhead, we use the following equations of the analysis [28]

$$T_{opt} = \sqrt{2(MTBF_{plat} - D - R)C} \tag{3}$$

$$\text{WASTE} = 1 - \left(1 - \frac{C}{T}\right)\left(1 - \frac{1}{MTBF_{plat}}\left(D + R + \frac{T}{2}\right)\right) \tag{4}$$

where C is the checkpointing cost, T is the period of checkpointing, R is the cost of restoring the checkpoint, D is the downtime, and $MTBF_{plat}$ is the platform Mean Time Between Failures (MTBF). All these parameters are measured in seconds.

Furthermore, we calculate the platform $MTBF$ which depends on the $MTBF$ of individual nodes, based on the following equation of the analysis [28]

$$MTBF_{plat} = MTBF_{node}/N \tag{5}$$

where $MTBF_{node}$ is the MTBF of individual nodes, N is the number nodes used to execute a specific workload.

Moreover, we estimate the $MTBF_{node}$ and the corresponding $MTBF(CL)_{node}$ with a confidence level (CL), based on the following equations of the works [32], [33]

$$MTBF_{node} = \frac{1}{\lambda}\Gamma(1 + \frac{1}{\kappa}) \tag{6}$$

where Γ refers to the gamma distribution, λ and κ being the scale parameter and the shape parameter of the Weibull distribution law, respectively, and

$$\text{MTBF(CL)}_{node} = \frac{2\ \text{MTBF}_{node}}{X^2_{1-CL}(2n+2)} \tag{7}$$

where X^2 refers to the chi-square distribution, CL is the target confidence level, MTBF_{node} is the experimentally observed MTBF and n the number of errors. When no errors are observed during the testing ($n = 0$), the respective X^2 distribution has 2 degrees of freedom [32], [33].

Panos Koutsovasilis is a PhD candidate at the ECE Department of the University of Thessaly in Volos, Greece. His research focuses on system level programming (operating systems, schedulers, execution policies), virtualization, energy efficiency and reliability on heterogeneous systems. He is also interested on high performance computing, parallel programming and software optimizations. He received his MSc in computer engineering from the ECE Department of the University of Thessaly, Greece.

Konstantinos Parasyris is a postdoctoral researcher at the Lawrence Livermore National Lab (LLNL) where he works on approximate computing for HPC systems. His research identifies opportunities to disrupt the power / quality of service / performance equilibrium in computing systems, toward more efficient computations, with controlled deterioration of the quality of results. He received the B.Sc., M.Sc., and Ph.D. in Electrical and Computer Engineering at the University of Thessaly (Greece). This work has been carried out while Dr. Parasyris was with the University of Thessaly.

Christos D. Antonopoulos is an Associate Professor at the ECE Department of the University of Thessaly in Volos, Greece. His research interests span the areas of system and applications software for high performance computing, emphasizing on run-time monitoring and adaptivity with performance and power criteria. He also works on improving the programmability of accelerator-based heterogeneous systems, as well as on techniques for automatic, application-driven redefinition of the hardware/software boundary on accelerator-based systems. He earned his PhD, MSc and Diploma from the Computer Engineering and Informatics Department of the University of Patras, Greece.

Nikolaos Bellas is Associate Professor at the Electrical and Computer Engineering Department of the University of Thessaly in Volos, Greece. His research interests include Approximate Computing, Low Power design, CAD tools for architectural synthesis, multimedia and image processing for parallel machines. He received his Diploma in Computer Engineering and Informatics from the University of Patras, Greece in 1992, and MSc and PhD in Electrical and Computer Engineering from the University of Illinois, Urbana-Champaign, in 1995 and 1998, respectively. From 1999 to 2007, he was a Principal Member of Technical Staff at Motorola Labs in the US. He was one of the architects of Falcon, an MPEG4 video decoding chip used by the first Motorola camera phone, and has worked extensively on architecting Systems On Chip for multimedia and imaging applications. He holds 10 issued US patents.

Spyros Lalis is an Associate Professor at the ECE Department of the University of Thessaly in Volos, Greece. His research interests are mainly in programming models, operating systems, distributed systems and ubiquitous computing systems. He has contributed to the development of system-level software, middleware and programming frameworks to support market-based resource allocation, distributed computing, ad-hoc wearable and personal computing, wireless sensor networks and approximate computing. He received a Diploma in Computer Science and a PhD in Technical Sciences from the Swiss Federal Institute of Technology Zurich (ETHZ).