Parallel Computing 73 (2018) 3–15

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Using virtualization to quantify power conservation via near-threshold voltage reduction for inherently resilient applications☆

Li Tan*, Nathan DeBardeleben, Qiang Guan, Sean Blanchard, Michael Lang

Ultrascale Systems Research Center, Los Alamos National Laboratory, USA

ARTICLE INFO

Article history:
Received 25 October 2016
Revised 22 May 2017
Accepted 26 July 2017
Available online 27 July 2017

MSC:
68M14
68M20
68N25
68W10
68W15

Keywords:
Inherently resilient applications
Power savings
Fault injection
Failure rates
Soft errors
Virtualization
Near-threshold voltage reduction

ABSTRACT

Power efficiency nowadays is a mainstream pressing issue in High Performance Computing (HPC), due to limited power supply capability of current and projected supercomputers. As a promising solution, leveraging inherent application resilience to relax power requirements of HPC runs can effectively save power with minor/acceptable loss of output quality. However, the challenges of this approach lie in: (a) how to reduce power usage of HPC runs online within allowable maximum extent such that quality metrics of applications can be satisfied, and (b) how to identify potential intrinsic nature of fault tolerance in general for an application. Existing efforts to date fail to address both challenges systematically and efficiently. In this work, based on virtualization and near-threshold voltage reduction techniques, we propose an empirical framework named `V-Power` to save the most power for inherently resilient applications. As an integrated empirical system, our approach effectively addresses the two above challenges using quantitative and fine-grained application inherent resilience analysis and frequency-independent near-threshold voltage reduction. Experimental results for a wide spectrum of scientific applications running on a 40-core power-aware server demonstrate that `V-Power` is capable of saving power up to 12.3%, resulting in a failure rate with acceptable program outputs.

1. Introduction

Considering ever-growing power bills and limited power supply capability of electrical facilities, power efficiency has already become a first-class citizen in scalable and cost-efficient High Performance Computing (HPC) systems. Numerous software/hardware solutions have been proposed to achieve power savings at different levels of system abstraction, with little performance degradation of HPC runs. Provided a growing number of specialized-purpose scientific applications in different domains nowadays, a promising solution is to leverage specifically *inherent resilience* of an application for operating in the low-power mode, while the resulting errors are *masked* or *tolerated* by the application. Note that the inherent resilience can be specific to certain error types, or be specific to certain portions of the application. Generally, HPC runs with masked

☆ This work was performed at the Ultrascale Systems Research Center (USRC) at Los Alamos National Laboratory (LANL), assigned the LANL identifier LA-UR-16-23584.

* Corresponding author.
E-mail address: darkwhite29@gmail.com (L. Tan).

http://dx.doi.org/10.1016/j.parco.2017.07.009
0167-8191/

errors result in successful completion, which highly depends on the application characteristics. For instance, the faulty values caused by the errors are not reused in the rest of a run [1]. On the other hand, errors in some applications (e.g., from the domains of image/video processing, recognition, and data mining) can be tolerated to some extent. Incorrect results in a run of such applications within a given range are acceptable [2] (e.g., minor image errors can be unperceivable by users). These applications can benefit from being operated in the low-power mode for power savings, with little loss of output quality.

Strategies at various layers of abstraction can be applied to leverage the inherent resilience of applications, including using inexact hardware [3,4], voltage scaling [5,6], load value approximation [7,8], and task skipping [9,10]. However, little work has been done to investigate the quantitative correlation between resilience and power efficiency of HPC runs, e.g., the integrated modeling between resilience factors such as failure rates, with power saving factors such as supply voltage. In this work, we propose to achieve power efficiency for scientific applications with inherent resilience to different types of soft errors including crashes, hangs, and silent data corruption, using near-threshold voltage reduction techniques [11,12]. There exist several efforts on employing voltage scaling techniques for power savings with minor loss of output quality at different levels of abstraction [5] and at GPU level [6], but neither of them discuss the quantitative modeling of resilience and power savings, nor experimentally evaluate the trade-offs between output quality loss and power efficiency.

CMOS-based hardware components such as CPU, GPU, and memory are generally the dominant power consumers in a computing system. Lowering supply voltage of them can effectively reduce power costs of the system, given that power consumption of these components is proportional to the product of operating frequency and supply voltage squared [13], which means voltage reduction has a greater impact on power savings than frequency reduction. Existing efforts extensively demonstrate that Dynamic Voltage and Frequency Scaling (DVFS) techniques can effectively save power/energy when the peak performance of hardware components (CPU [14,15], GPU [16,17], and memory [18,19]) is not necessary. Nevertheless, a critical limitation of existing DVFS work lies in the fact that the employed DVFS techniques are essentially *frequency-directed*: Voltage and frequency are scaled down together in the presence of hardware idle time [20], which fails to fully exploit potential power saving opportunities during HPC runs. As a remedy solution, *undervolting* [12,20,21] has been employed to further reduce power costs, which reduces voltage independent of frequency scaling. Using undervolting, hardware components are supplied with a voltage that is lower than the one paired with a given frequency used in DVFS.

As a generic approach applicable during any stages of HPC runs, undervolting is capable of saving extra power in addition to using DVFS during hardware idle time. Nevertheless, the trade-off of using undervolting is increased failure rates [11] of the components with lowered voltage. For inherently resilient applications, the growing number of errors can be tolerated to some extent, and thereby incur negligible impacts on performance and output quality. Therefore, significant power efficiency can be achieved with minor (or acceptable, depending on application characteristics) loss of resilience. In this work, targeting inherently resilient applications, we leverage the resilient nature of specific types of applications and frequency-independent near-threshold voltage reduction, proposing a framework of Virtualization-based `Power`-efficiency (`V-Power` for short) to save the most power with minor/acceptable loss of output accuracy. We aim to explore the fact that, theoretically and experimentally, a wide scope of (and extensively used) scientific applications running on future exascale systems can benefit from power efficiency, if resilience is provided to some extent as the inherent application characteristics. In summary, the contributions of this paper include:

- We leverage inherent resilience of applications for considerable power savings with minor/acceptable loss of program output quality, using frequency-independent near-threshold voltage reduction;
- We develop an integrated virtualization-based framework named `V-Power` to analyze potential resilience of applications, and achieve power efficiency for a runtime-dominant kernel/function of the application, if it is validated resilient by nature;
- Our approach is experimentally evaluated on a power-aware platform for a wide scope of scientific applications to save up to 12.3% in power, with an error rate for acceptable program outputs.

The remainder of the paper is organized as follows: Background knowledge is introduced in Section 2. Details of the proposed `V-Power` framework are presented in Section 3. Experimental results are provided in Section 4. Section 5 discusses related work and Section 6 concludes.

2. Background

2.1. Inherently resilient applications

Regardless of hardware relaxation techniques [22,23], some applications are inherently resilient to errors, i.e., for provided inputs, when errors in runs occur, they produce *acceptable* outputs with (partially) incorrect or inaccurate results. For instance, pixel errors in an image or a video can exist to some extent without the perception from users (or with user-acceptable quality degradation). For some approximate computing applications, a range of output values are considered acceptable, since there is no unique golden output for a golden run. Due to precision requirements of some iterative scientific applications, computation errors occurring in one iteration can get cancelled by truncated numbers during the computation in later iterations [24]. This property of self-tolerance/self-healing to faults is referred to as *inherent application resilience*, arising in some categories of scientific applications. Specifically, applications with such a property span from domains of image/video processing, recognition, and data mining. Notably, the acceptance of program outputs is defined

Table 1
Quality metrics for applications and Kernels of different domains.

Quality Metric	Domain	Representative [Benchmarks] Applications/Kernels
Output/Runtime Difference (between runs or with the golden run)	General	[PARSEC] blackscholes, swaptions, vips; [NPB] EP, CG, FT [Lonestar] Barnes-Hut n-body simulation
Numerical Error	Linear Algebra	[ScaLAPACK] Cholesky/LU/QR factorizations, matrix multiplication; [Misc] partial differential equations
Pixel Error (e.g., image error ratio PSNR/MSE)	Image Processing	[PARSEC] bodytrack, facesim, x264, raytrace [MediaBench] jpg2000/mpeg encoder and decoder
Classification Accuracy	Machine Learning	[PARSEC] ferret, streamcluster; [Misc] k-means clustering neural networks, support vector machines
Ranking Accuracy	Searching	[RankNet] web/text search; [Misc] supervised semantic indexing document search
Decision Correctness	Data Mining	[MATLAB] fitctree, fitrtree; [Misc] barcode recognizer image binarization, decision trees

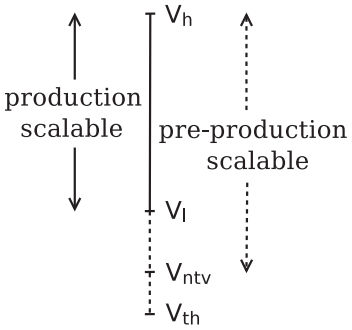

Fig. 1. Voltage levels for empirical voltage reduction.

by specific *quality metrics* that highly depend on application characteristics. Table 1 summarizes various typical quality metrics for applications/kernels of different domains. Note that not all listed are inherently resilient, provided specific quality metrics. Intuitively, inherently resilient applications can run in the low-power mode for power saving purposes, with resulting errors tolerated by the applications themselves.

2.2. Near-threshold voltage reduction

As introduced, undervolting is a voltage reduction technique regardless of frequency scaling for greater power savings. Undervolting differs from traditional DVFS techniques in the sense that the frequency of hardware components is kept the same during undervolting, with computation throughput maintained. Ideally, if the resulting errors can be tolerated by inherent resilient applications, no performance impacts will be incurred together with the maximized power savings. Using undervolting, supply voltage can be lowered as close as threshold voltage of the components, the lowest voltage level that guarantees the minimum electronic activities in the circuit. Note that undervolting can be conducted together with DVFS, i.e., as an appropriate frequency is scaled to by DVFS, instead of using the voltage paired with the selected frequency, voltage can be further reduced as close as threshold voltage. In this work, we assume that DVFS techniques are employed when necessary (e.g., race-to-halt at OS level [25] or library level [26], and critical path aware slack reclamation at OS level [15] or library level [27]), and near-threshold voltage reduction is conducted on top of the selected frequency by DVFS techniques, if employed.

Nowadays, hardware vendors have launched cutting-edge nano-technology that enables processors to be supplied with a significantly low voltage, e.g., using Intel's Near-Threshold Voltage (NTV) design [11], the 32 nm low-leakage Intel Claremont processor can operate from 280 mV at 3 MHz and scale up to 1.2 V at 915 MHz, with the minimum power of 2 mW. Note that this NTV technique requires low supply voltage accompanied by low operating frequency as well, different from our frequency-independent near-threshold voltage reduction (referred to as FiNTV for short henceforth). Empirically, protected by the OS, production processors are generally locked and will typically shut down or crash when near-threshold voltage reduction without frequency reduction is performed. Experimentally, for a customized pre-production Intel Itanium II 9560 processor [12], undervolting to near-threshold level has been validated to save power effectively at the cost of increased ECC memory errors, as the first practical undervolting effort on real machines. Tan et al. [20] relaxed the hardware requirements of undervolting to general production machines by emulated scaling, evaluated on an HPC cluster with production AMD Opteron 2380 processors. Fig. 1 shows different voltage levels state-of-the-art voltage reduction techniques are capable of scaling, where V_h/V_l, V_{ntv}, V_{th} refer to the maximum/minimum voltage (paired with the maximum/minimum frequency), a near-threshold voltage, and the threshold voltage individually. General production processors can be scaled within the

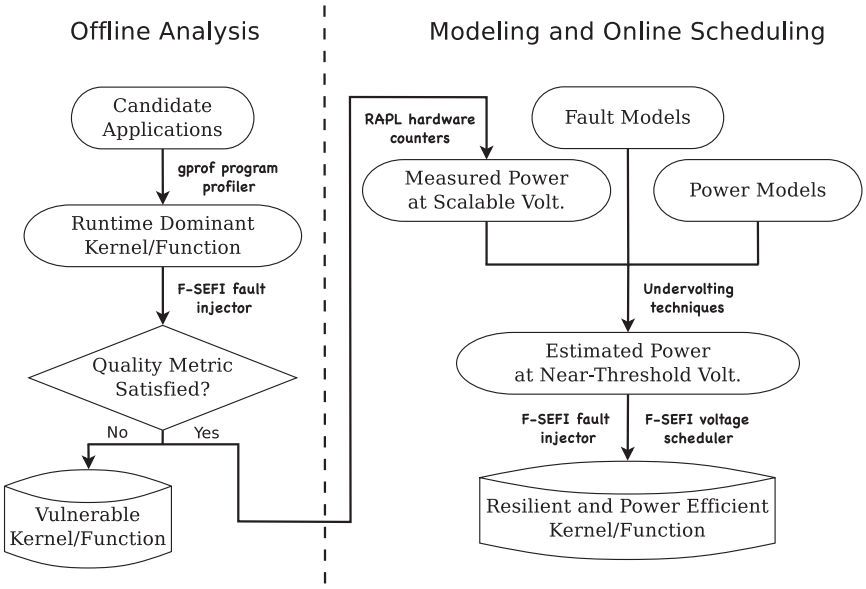

Fig. 2. Overview of the `V-Power` famework.

voltage range $[V_l, V_h]$, while for customized pre-production processors, they can be supplied with a near-threshold voltage V_{ntv}. Our goal is to enable production HPC systems to achieve the maximum power savings using FiNTV and application inherent resilience, although empirically they cannot be supplied with V_{ntv} without frequency reduction.

Targeting general production HPC clusters, in this work we adopt a similar undervolting strategy as employed in [20], and extend it to near-threshold voltage level with quantitative models. We elaborate our quantitative near-threshold voltage reduction with fine-grained fault and power models used in `V-Power` in Section 3.

2.3. Virtualization-based fault injection

As a technique creating the infrastructure-independent virtual execution environment, i.e., Virtual Machines (VM), virtualization enables software running on VM to be separated from the underlying hardware resources, which provides improved scalability, autonomicity, and efficient utilization of hardware resources. Virtualization is capable of evaluating different hardware and new architectures with minimal modification to a computing system or an application. For instance, using virtualization, several guest OS can be run in parallel on a single CPU without interference. Fault injection within this work is accomplished using a fault injector F-SEFI [28] based on the virtualization technique QEMU [29], which supports both CPU emulation for architectures like x86, PowerPC, ARM and Sparc, and full system emulation, using portable dynamic instruction translation. Specifically, when an application is launched at the guest OS, instruction sets are translated and delivered from the guest OS to the host OS, using virtualization. Soft errors are emulated by intercepting instructions and corrupt register data during instruction translation. Using a binary function symbol table, faults can be injected into a chosen instruction in a specific function of an application.

3. V-Power: design and implementation

In this section, we present design and implementation details of our power saving `V-Power` approach for inherent resilient applications. Fig. 2 shows the `V-Power` framework, consisting of two functional components: *offline* application inherent resilience analysis, and *online* voltage scheduling with fine-grained power and fault modeling. Specifically, we utilize the program profilers gprof [30] and Intel RAPL interface [31], and extend the fault injector F-SEFI [28] to implement `V-Power` as an integrated system.

The goal of `V-Power` is to leverage the resilience nature of specific applications for power efficiency, with little impacts on program output quality, i.e., minor loss of resilience. As shown in Fig. 2, the workflow of `V-Power` proceeds as follows: Given a candidate application with a runtime dominant kernel/function (profiled using gprof for runtime information), we utilize F-SEFI to inject faults to validate if the kernel/function is vulnerable or resilient to concerned error types (discussed in Section 3.2), using application-specific quality metrics (discussed in Section 3.1). If the kernel/function is validated resilient by the offline resilience analysis, we pass it to the online module of `V-Power`. Based on measured power data using RAPL and established power models regarding frequency/voltage, we can estimate the power costs at near-threshold voltage V_{ntv}, and we inject errors to emulate the faulty runs at V_{ntv}. We demonstrate that the selected kernel/function is resilient and power-efficient using our `V-Power` framework.

3.1. Application inherent resilience analysis

We next discuss strategies of analyzing inherent resilience of applications. We investigate application inherent resilience at function level for two primary reasons: (a) In general, an inherently resilient application consists of resilient functions and non-resilient functions (we assume the studied application is comprised of functional units, e.g., functions). Our power saving approach should be applied to the resilient functions within an application, and circumvent the non-resilient ones; (b) large-scale scientific applications are usually by nature iterative, i.e., the primary operations (e.g., computation in terms of floating-point calculation, and communication in terms of data movement) are conducted within a loop or multiple nested loops, which can be the most time-consuming part of the application. If the loop-based function/kernel is identified resilient, it is selected as an ideal candidate for our solution. Note that program runs are profiled using gprof dynamically, but the critical application inherent resilience analysis is conducted offline based on program traces and outputs.

As shown in the left part of Fig. 2, firstly, we obtain the most time-consuming kernel/function during an HPC run from the call graph of the run generated by gprof, with the provided inputs. This step is important, since saving power for a runtime dominant kernel/function achieves more energy efficiency. Secondly, a separate run is performed while faults of different instruction types are injected using our F-SEFI fault injector. Per application characteristics, a post-checking step is done offline to verify if the outputs satisfy specific program quality metrics (see Table 1). If not, the selected kernel/function is declared a vulnerable one, and we move on to verify the second most time-consuming kernel/function (not shown in Fig. 2). Otherwise, the resilient and time-dominant kernel/function is considered to save power using near-threshold voltage reduction strategies with fine-grained fault and power models, i.e., the modeling module Section 3.2) and the online voltage scheduling module (Section 3.3) of `V-Power`.

Discussion. Our approach needs to profile program runs to identify appropriate power saving candidates, which has typical weaknesses of all profiling-based approaches. The applications evaluated may have qualified candidates but for some inputs, the normal observation for other inputs may not manifest well in program runs. We select representative inputs for profiling an application, but also make sure the selected inputs are not within a limited scope of all possible inputs for the application. We use the same selected inputs when finding the time-dominant and resilient kernel/function in an application, with fault injection to emulate the faulty runs at V_{ntv}.

3.2. Fault and power modeling

Next we elaborate the characterization and formulation of failures and power consumption for the investigated HPC systems.

3.2.1. Fault characterization

Faults incurred in a computing system can be categorized into *hard* errors and *soft* errors, where the soft errors are of great concern nowadays in HPC, since they are *transient* and thus can contaminate HPC runs silently (i.e., difficult to detect and correct), including memory bit-flips and logic circuit miscalculation [21]. On the other hand, hard errors refer to *permanent* component-wise or system-wide failures, including node crashes from dysfunctional hardware and system abort from accidental power outage [20]. Due to the severity of soft errors as HPC systems nowadays grow dramatically in size and usage, we focus on soft errors in this work.

Per error occurrence locations, soft errors can be triggered generally in memory (both SRAM and DRAM by bit-flips) [32], computing units such as CPU/GPU/FPGA (radiation-induced logic circuit errors) [33], and in disk (sector intermittent errors from thermal asperity) [34]. We assume that all errors eventually propagate to the computational logic units, and thus register-level fault injection by F-SEFI is sufficient to reflect errors in either instructions or data. On the other hand, per the mechanism of detection and correction, soft errors can be categorized as *Detected and Corrected Errors* (DCE), *Detected but Uncorrectable Errors* (DUE), and *Silent Errors* (SE) [35]. Any unmasked SE are referred to as *Silent Data Corruption* (SDC), i.e., incorrect program outputs. DCE generally occur in ECC-protected SRAM/DRAM, and examples of DUE include *crashes* and *hangs* of program execution. Specifically, crashes refer to an exception raised during execution that causes its termination, and hangs refer to the cases that the execution lasts for a significantly longer time than normal. There also exist *benign* errors, i.e., the cases that the application finishes with a valid output. In this work, we consider crashes, hangs, SDC, and benign errors occurring in HPC runs.

3.2.2. Voltage-directed failure rate modeling

Existing studies indicate failures of combinational logic circuits comply with a Poisson distribution, determined by both operating frequency and supply voltage [36]. Our previous work summarized a handy formula of average failure rates in terms of supply voltage only [20], by substituting frequency with voltage (frequency has a non-linear positive correlation with voltage [37]) in the general equation stated in [38]:

$$\lambda(f, V_{dd}) = \lambda(V_{dd}) = \lambda_0 \, e^{\frac{d\left(f_{\max} - \beta\left(V_{dd} - 2V_{th} + \frac{V_{th}^2}{V_{dd}}\right)\right)}{f_{\max} - f_{\min}}} \tag{1}$$

Constants d and β are architecture-dependent, reflecting the sensitivity of failure rate variation with frequency/voltage scaling. V_{th} and V_{dd} are threshold voltage and supply voltage respectively. f_{max} and f_{min} are the maximum frequency and

the minimum frequency within the scope of DVFS individually. As mentioned in Section 2, near-threshold voltage reduction is disabled by the OS for production machines as hardware protection mechanism. Therefore real errors at near-threshold voltage level cannot be observed experimentally, since our target is production machines used in real HPC clusters.

Model validation. This failure model has been cross-checked by experimentally measured failure rates on a recent HPC processor: Based on the observed ECC correctable errors reported in [12] for an Intel pre-production processor, Eq. (1) was demonstrated to be accurate to model failure rates at different voltage levels [20], and is thus considered applicable for HPC architectures nowadays, and adopted in this work for failure modeling during near-threshold voltage reduction and fault injection. As concluded in [20], $\lambda(f, V_{dd})$ monotonically strictly decreases as V_{dd} increases, for all valid V_{dd} values. Given that frequency has a positive correlation with voltage, we know that $\lambda(f, V_{dd})$ is a monotonically strictly decreasing function of frequency as well. In Eq. (1), if the constants f_{max} and f_{min} are different in another DVFS setting, $\lambda(f, V_{dd})$ changes accordingly following the equation as the baseline changes, which is observably true since in a real setup, processors with different DVFS settings have different failure rates empirically.

3.2.3. Power metering in virtualization-based systems

Although virtualization has advantages in efficiently utilizing various resources in a non-intrusive manner, power measurement for virtualization-based systems is a great challenge. Obviously, power meters cannot be physically attached to VM for direct power measurement. Only the total power costs of the server where VM runs are measurable. Power consumption of virtualized environment purely is difficult to obtain in hardware. Several efforts have studied potential practical solutions of measuring VM power. VM power models were built to infer power costs from resource usage at runtime, with instrumentation in hardware and hypervisors [39]. A middleware-based fine-grained monitoring approach was proposed to collect process usage and thus infer power costs of applications without power meters [40]. Existing VM power estimation solutions were evaluated on real platforms to be accurate with low runtime overhead. Experimental results in [40] also show that power costs collected using Intel RAPL hardware counters follow the trend of real usage, with minor overestimation. In order to focus on our power saving approach, modeling power costs of HPC runs in virtualization-based systems is out of the scope of this work, and we adopt power costs reported by the RAPL counters.

3.2.4. Measurement/estimation-based power modeling

For an HPC system, the following power model is used to calculate the nodal power consumption [41], assuming the processor is DVFS-enabled:

$$\begin{aligned} P &= P_{dynamic}^{processor} + P_{leakage}^{processor} + P_{leakage}^{other} \\ &= ACfV_{dd}^2 + I_{sub}V_{dd} + I'_{sub}V'_{dd} \end{aligned} \tag{2}$$

where A and C are the percentage of active gates and the total capacitive load in a CMOS-based processor respectively. I_{sub}/I'_{sub} and V_{dd}/V'_{dd} are subthreshold leakage current and supply voltage of processors and all other nodal components, individually. When voltage scaling techniques (DVFS and undervolting) are applied to processors, $I'_{sub}V'_{dd}$ can be denoted as a constant P_c. With measured power data, the simplified power equation is employed to estimate the nodal power costs at near-threshold voltage, as shown in Algorithm 1 (V_{dd} is denoted as V for short). P_{sav} refers to the percentage of

Algorithm 1: Emulation of the power costs at NTV.

Input: A candidate application app with its quality metric qm, frequency/voltage pairs used in DVFS: $\{f_h/V_h, f_m/V_m, f_l/V_l\}$, near-threshold voltage V_{ntv}, and calculated failure rate λ_{ntv} at V_{ntv}.
Output: Power costs P_{ntv} and failure data at V_{ntv}.

```
1  begin
2      make the system idle, operating at nominal f/V
3      for f/V ∈ {f_h/V_h, f_m/V_m, f_l/V_l} do
4          scale to the selected f/V and run app
5          measure power P = ACfV^2 + I_sub V + P_c
6      solve AC, I_sub, and P_c, given P_h, P_m, and P_l
7      scale to f_h/V_h and run app
8      calculate P_ntv = AC f_h V_ntv^2 + I_sub V_ntv + P_c
9      P_sav ← (P_h − P_ntv)/P_h × 100%
10     InjectFault(app, λ_ntv)
11     GetFailureData(app, qm)
```

power savings achieved by near-threshold voltage reduction. Since undervolting to near-threshold voltage level is generally disabled for production machines, we inject errors to emulate the faulty runs at the NTV low-power mode. Functions *InjectFault*() and *GetFailureData*() in Algorithm 1 abstract fault injection and failure analysis operations, respectively. Faults

are injected at the calculated failure rate of λ_{ntv} (using Eq. (1)) during HPC runs. We can obtain failure data after the execution, provided application-specific quality metrics shown in Table 1.

Note that the output power P_{ntv} is obtained using the measurement/estimation-based method: All power constants used in Eq. (2) are solved from measured power data, while power consumption at the unscalable V_{ntv} is calculated from the same power equation with f_h and V_{ntv} in place. Leveraging measured data partly and fine-grained power formula, this method is more accurate than purely estimation-based approaches, and is capable of emulating power costs at unscalable voltage levels for production machines.

3.3. Virtualization-based near-threshold voltage reduction

Given the resilient and time-dominant kernel/function of an application identified by the offline module of `V-Power`, the virtualization techniques used in fault injection can also facilitate near-threshold voltage reduction for the candidate kernel/function, by profiling program execution dynamically. When the candidate kernel/function is executed (located using the pre-generated function symbol table extracted from the application binary), voltage reduction is performed until the end of its execution, i.e., the application runs in the low-power model during the function starts and ends, with the expectation that the most power can be saved (due to its time-dominance) at the cost of minor quality loss (due to its resilience).

Next we present the details of selecting the near-threshold voltage. As shown in Fig. 1, V_{ntv} refers to a voltage level that lies in between V_l and V_{th}, both of which can be found in processor specifications provided by vendors. It is critical to pinpoint the potential lowest NTV level, since it directly determines the ultimate power savings. To this end, two requirements need to be considered in the selection: Firstly, as shown in Fig. 1, production processors can be safely scaled within the voltage range $[V_l, V_h]$. There exists a lowest safe core voltage below V_l that does not crash the system. Experimentally, we choose such a voltage at the lowest frequency f_l, referred to as $V_{safe\_min\min}$. Secondly, given an application, there exists a lowest voltage level under which the quality metric of the application can be satisfied. Scaling to any voltage levels lower than it will incur unacceptable output quality degradation, i.e., more errors than the pre-defined threshold value per application characteristics. This voltage is referred to as $V_{qm\_min\min}$. The optimal lowest near-threshold voltage is the higher one of the two above voltage levels, i.e., $max(V_{safe\_min\min}, V_{qm\_min\min})$, such that both of the two requirements are guaranteed. Algorithm 2 shows the selection process in pseudo code.

Algorithm 2: Selection of the lowest NTV.

Input: A candidate application app with its quality metric qm, the lowest frequency/voltage f_l/V_l, and threshold voltage V_{th}.
Output: The lowest near-threshold voltage V_{ntv}.

```
1  begin
2      make the system idle, operating at nominal f/V
3      scale to f_l
4      while system runs reliably && V_th < V ≤ V_l do
5          undervolt to V and run app
6          V_safe_min ← V
7      scale to f_h
8      while qm is satisfied && V_th < V ≤ V_l do
9          undervolt to V and run app
10         V_qm_min ← V
11     V_ntv ← max(V_safe_min, V_qm_min)
```

4. Evaluation

In this section, we present details of experimental evaluation on our `V-Power` approach for a wide scope of mainstream inherently resilient scientific applications running on a power-aware production HPC server. Overall, the empirical study aims to showcase that: (a) `V-Power` is capable of identifying inherently resilient runtime-dominant function/kernel of the evaluated applications, and (b) with little loss of output quality, significant quantitative power savings for the selected function/kernel can be achieved by `V-Power`, using online frequency-independent near-threshold voltage reduction on a virtualization-based platform, with fault injection enabled to emulate faulty HPC runs in the low-power mode.

4.1. Experimental setup

Experiments were conducted on a wide scope of nine mainstream scientific applications to benchmark our approach, from domains of machine learning, image/video processing, and scientific computing, summarized in Table 3. The

Table 2
Hardware configuration for all experiments.

System size	40 logical cores
Processor	Intel Xeon E5-2660 (10-core)
CPU Frequency	1.2 to 2.6 GHz incremented by 0.1 GHz
CPU Voltage	1.05, 0.65, 0.49, 0.40 V ($V_h/V_l/V_{safe\_min}/V_{th}$)
Memory	128 GB RAM
Cache	640 KB L1, 2560 KB L2, 25600 KB L3
OS	Ubuntu 14.10, 64-bit Linux kernel 3.16.0
Power Meter	Intel RAPL

Table 3
Benchmark details. From left to right: benchmark name, benchmark suite, benchmark description and test case used, problem domain, target function, execution time percentage of the function relative to the total, and type of soft error incurred and tainted instruction from fault injection.

Benchmark	Suite	Description and Test Case	Domain	Function	Runtime (in %)	Error (Tainted Inst.) from Fault Injection
ferret	PARSEC	Perform content-based similarity search in 34,973 images.	Machine Learning	do_query	99.1%	crash (CMP)
streamcluster	PARSEC	Perform an online clustering algorithm on 16,384 points.	Machine Learning	pkmedian	99.8%	SDC and crash (FADD/FMUL/CMP)
vips	PARSEC	Perform image transformation pipeline on an image with 1600 × 1200 pixels.	Image Processing	im_LabQ2disp	99.2%	crash (FMUL/CMP)
raytrace	PARSEC	Perform real-time raytracing of a Buddha statue with 1 million polygons.	Video Rendering	renderFrame	78.4%	crash (FADD/FMUL/CMP)
imgcmp	MediaBench	Calculate the mean squared error of two 1.21 MB PGM images.	Image Processing	msen	96.9%	SDC and crash (FADD/CMP)
jasper	MediaBench	Convert a 257.1 KB image from BMP format to compressed JPEG format.	Image Processing	jpg_encode	92.9%	SDC and crash (CMP)
h264dec	MediaBench	Decode a 537.4 KB compressed video stream using the H.264 standard.	Video Decoding	decode_one_frame	97.7%	SDC (CMP)
EP	NPB	Generate independent Gaussian random variates by Marsaglia polar method.	Probability Theory and Statistics	main	100.0%	SDC, crash, and hang (FADD/FMUL/XOR/CMP)
CG	NPB	Estimate eigenvalue of a sparse matrix with conjugate gradient method.	Numerical Linear Algebra	conj_grad	96.4%	SDC, crash, and hang (FADD/FMUL/XOR/CMP)

benchmarks were selected from PARSEC [42], MediaBench [43], and NPB [44] benchmark suites. Table 2 lists hardware configuration of the experimental power-aware server with two Intel Xeon E5-2660 processors based on the Haswell microarchitecture [45]. Power consumption of HPC runs on the server was collected by Intel RAPL hardware counters [31] that reports in-band power costs of the total processor package and DRAM, which can be used to estimate the total system power costs during HPC runs (power savings reported henceforth are system-wide). Using RAPL and its API, it is convenient and lightweight to emulate power consumed during HPC runs without physical power meters attached. Before presenting experimental results, we detail the implementation of techniques used in our approach.

4.2. Empirical implementation

As discussed and shown in Fig. 1, production machines used in HPC clusters are protected by the OS from voltage levels lower than V_l. Keeping the processor frequency constant, we manage to implement Frequency-independent Near-Threshold Voltage reduction (FiNTV) below V_l on the experimental Intel processors by directly modifying the core voltage Model-Specific Register (MSR, specifically MSR_PERF_STATUS), defined in [31]. The high 16-bit (i.e., VID) of the 48 bit register value determines the P-state core voltage, which can be calculated as VID/2^{13}. The lowest near-threshold voltage was selected using Algorithm 2, given an application and its quality metric. We scaled to this selected voltage by altering the value of the high 16-bit of the core voltage MSR.

As stated, although power efficient, FiNTV on production processors incurs a growing number of failures from scaling to voltage levels lower than V_l. We calculated the failure rates at corresponding voltage levels using Eq. (1), and injected errors at the calculated failure rates using our fault injector F-SEFI to emulate faulty HPC runs at near-threshold voltage levels. Note that in Eq. (1), there is a reference failure rate λ_0 at $V_h/f_{\max}$, which can be eliminated by dividing an unknown failure rate with a known one. We used the failure rates reported in [12] as known data as the baseline for our experimental platform. Although different ISA is employed by the two evaluated architectures, both have considerable similarity in power configuration [46]. Moreover, from [12] and [47], we observed that failure rates of different cores on the same die differ but only have slight variation. Thus we adopted the failure rates reported in [12] as an average value across cores, and consider it is sufficient to reflect the trend under the circumstance of experimental hardware limitation. Table 4 lists the calculated failure rates at voltage levels ranging from V_h to V_{ntv} (we consider the documented highest voltage level 1.30 V

Table 4
Calculated failure rates at different voltage levels (Unit: Voltage (V) and Failure Rate (errors/minute)).

Core voltage	Calculated failure rate
1.05	1.33×10^{-5}
0.95	1.62×10^{-4}
0.85	1.77×10^{-3}
0.75	1.70×10^{-2}
0.65	0.14
0.55	1.06
0.49	2.79
0.45	6.72

Table 5
Quality metrics with specific threshold values and lowest safe voltage levels for different benchmarks (Unit: Voltage (V)).

Benchmark	Quality Metric w/ Threshold Value	Lowest Safe Voltage	
		V_{qm_min}	V_{safe_min}
ferret	$\Delta_{\text{runtime}} < 20\%$	0.45	
streamcluster	$\Delta_{\text{output}} < 10\%$; output or not	0.51	
vips	output or not	0.47	
raytrace	output or not	0.49	
imgcmp	$\Delta_{\text{MAE}} < 10\%$; output or not	0.51	0.49
jasper	$\Delta_{\text{MAE}} < 10\%$; output or not	0.49	
h264dec	$\Delta_{\text{SNR}} < 10\%$	0.49	
EP	numerically verified or not	0.51	
CG	numerically verified or not	0.55	

as corresponded to the Turbo frequency 3.30 GHz which is not used in our experiments). We can see that for voltage levels higher than V_l (including V_l), the rates is low so that no fault injection is needed given that a single run of each application takes less than one minutes in our experiments. Specifically, we performed 3000 runs with fault injection for each benchmark. For near-threshold voltage levels that incur more than one error in an HPC run, e.g., 0.49 V and 0.45 V shown in Table 4, we injected multiple number of errors into each run per calculated failure rates.

Faults were injected at register level for different types of instructions of applications running on guest OS. We consider soft errors occurring in functional units of processors (e.g., ALU and FPU). We corrupted guest program instructions by intercepting and contaminating values of registers during instruction translation at runtime. Crashes occur when a tainted register value holds an address. SDC is incurred if only data corruption of registers is involved in fault injection. Rarely, there exist some hangs in HPC runs, given mutually blocking shared resources. Per application characteristics, the last column of Table 3 lists different types of instructions we corrupted for each benchmark, including `FADD`, `FMUL`, `CMP`, and `XOR` instructions.

Fault injection mechanism representativeness. In general, only the hardware (i.e., silicon) level fault injection method can best represent all possible errors in the reality for a computing system. However, the expenses in time and money on specially designed hardware are typically unaffordable. The benefits of using fault injection at hypervisor level are to improve the cost-efficiency. Although our fault injection cannot cover all types of faults, it still should be considered as a useful tool if it is much needed to emulate the errors in CPU logics/registers where the ECC is not applied. More fault models can be implemented in our approach if we have the data and information to build the fault models. For example, one of our previous work discussed the fault model of cache errors [48].

4.3. Results

In this section, we present detailed experimental results for all benchmarks on evaluating resilience characterization and power saving capabilities of our `V-Power` approach with elaborated failure and power data, given specific quality metrics of the evaluated applications. In our experiments, for all applications, the most time-consuming function, i.e., the target function, is resilient provided application-specific quality metrics. For evaluating the maximum benefits from our approach, we did not include the cases that the most time-consuming function is not resilient and thus the second most time-consuming function needs to be considered.

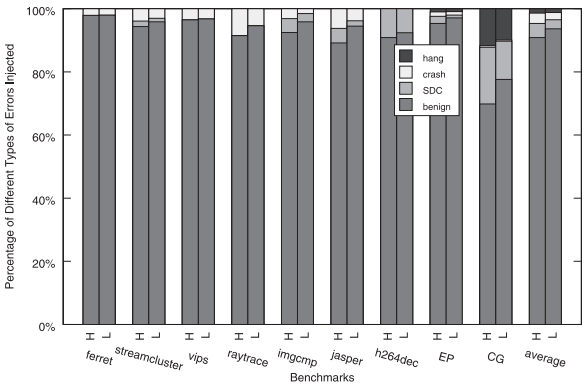

Fig. 3. Breakdown of types of errors injected.

4.3.1. Application inherent resilience with quality metrics at near-threshold voltage levels

Table 5 elaborates quality metrics with application-specific threshold values for the evaluated applications. In general, determining the type and range of quality metrics varies from application to application, which highly depends on the following factors: (a) application-specific characteristics including application domains and algorithmic patterns, (b) output accuracy requirements that rely on both applications and users, (c) execution scenarios (e.g., a chosen quality metric can be acceptable in one scenario but not in another), and (d) what trade-offs are needed (e.g., quality can be traded-off against execution time and power costs). In our experiments, we intend to evaluate that given a specific quality metric of applications, our approach is effective to achieve the power saving goals. The actual values of the selected quality metrics should not affect the nature of our evaluation, but definitely will change the power savings achieved. We carefully determine what a typical quality metric is for the evaluated benchmarks, based on the above factors discussed, and if they can be evaluated or not in the experiments.

We denote the difference between program outputs before and after fault injection as Δ. Generally, for image/video processing applications, we adopt the threshold value of quality metric to be 10%, since standard output errors (e.g., (Peak) Signal-to-Noise Ratio and Mean Absolute Error) less than 10% are considered acceptable in this domain [2]. For HPC runs where crashes occur, a normal output cannot be produced, and thus the fact that if there exists a successful output is selected as an additional threshold value. Specifically, for applications *vips* and *raytrace*, we did not observe output errors but only crashes, but for applications *imgcmp* and *jasper*, we observed both. For numerical linear algebra kernels *EP* and *CG*, their own numerical verification modules conduct quality checking of program outputs. With the commonly shared system level lowest safe voltage V_{safe_min} 0.49 V, Table 5 also lists the lowest safe voltage levels for various application-specific quality metrics, ranging from 0.45 V to 0.55 V. For HPC runs with more than one type of injected errors (i.e., with more than one quality metric and possibly more than one V_{qm_min}), the higher value of V_{qm_min} is shown in the table to guarantee all quality metric satisfied. We can see that applications with higher resilience (e.g., *ferret* and *vips*) have a lower V_{qm_min} value. As stated in Algorithm 2, the higher one between V_{qm_min} and V_{safe_min} is selected as the optimal near-threshold voltage level in our `V-Power` approach. Before showing power savings of our approach, we next present more failure analysis of the emulated faulty HPC runs using fault injection.

4.3.2. Types of faults injected breakdown

Fig. 3 gives a detailed breakdown of different types of errors in our fault injection campaign, where `H`/`L` columns represent fault injection at higher/lower failure rates from greater/smaller voltage reduction (as provided in Table 4), individually. Specifically, the `H` columns refer to fault injection at V_{qm_min}, while the `L` columns correspond to fault injection at a voltage level V ($V_l < V < V_{qm\_min}$). We can see that in general the evaluated applications are inherently resilient, except for *CG*, which is vulnerable to hangs (10.8% on average) and SDC (15.0% on average). Although inherently resilient, HPC runs with fault injection at lower failure rates suffer from less number of errors. Overall, the average failure rate of all types of possible errors for all benchmarks is 9.1% for `H` and 6.3% for `L` (7.7% on average). Moreover, there exists another interesting observation: The most resilient benchmarks (e.g., *ferret* and *vips*) as shown in Fig. 3 have lower V_{qm_min} as shown in Table 5. However, the less resilient benchmarks (e.g., image/video processing applications *raytrace, jasper*, and *h264dec*) do not necessarily have higher V_{qm_min}. This is due to the fact that higher number of output errors for image/video processing applications are generally acceptable with more relaxed quality metrics compared to other applications. The most common soft errors occurring in the experiments are SDC and crashes, while hangs are much more less incurred in HPC runs.

4.3.3. Power savings

With application inherent resilience data discussed, Fig. 4 depicts the normalized power savings of our `V-Power` approach for all benchmarks. Compared to baseline runs without `V-Power`, our approach is capable of saving power costs of HPC runs by 12.3% on average, up to 15.9% (*ferret*), and at least 9.0% (*CG*), which comes from core voltage reduction

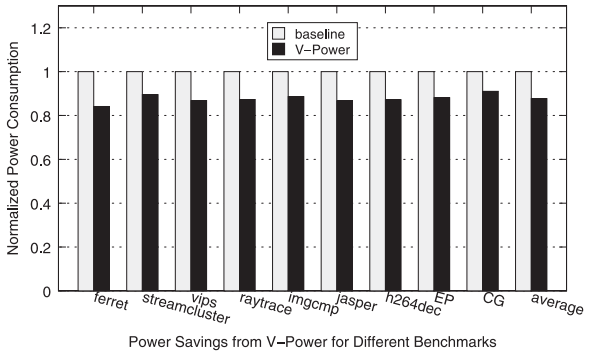

Fig. 4. Power efficiency of the `V-Power` approach.

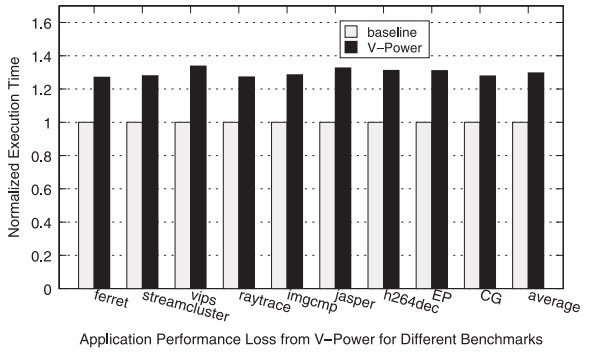

Fig. 5. Application performance degradation of the `V-Power` approach.

from the nominal V_h/V_l (0.65–1.05 V) to greatly reduced V_{ntv} (0.49–0.55 V). The power savings achieved match well with the selected lowest near-threshold voltage shown in Table 5, since from the adopted power model in Eq. (1), we know that power consumption is positively correlated with supply voltage. In `V-Power`, near-threshold voltage reduction is performed independent of frequency scaling, i.e., FiNTV as stated earlier. Therefore, power costs of HPC runs monotonically strictly decrease when FiNTV is conducted. Due to the constraints of doing FiNTV on production machines, our fine-grained power model is leveraged to emulate power costs at the selected lowest near-threshold voltage using the measurement and estimation combined method, as introduced in Algorithm 1.

4.3.4. V-Power system and application performance degradation

As demonstrated, our `V-Power` approach has been evaluated effective and efficient to save power, provided inherently resilient applications. With a lightweight modeling and online voltage scheduling module attached to a soft error fault injector, the primary performance degradation of `V-Power` bringing to applications comes from fault injection activities, which averages 29.7% and can be up to 33.8% (*vips*) as shown in Fig. 5. This is due to the fact that injecting faults in our experiments requires dynamic instruction-to-instruction translation, without any software bypass or hardware acceleration to speed up instruction analysis and fault injection at register level. Moreover, at system level, the fundamental virtualization techniques based on QEMU VM and its hypervisor employed as the basis of `V-Power` incur significant performance overhead system-wide (e.g., up to 200%) [28,29], due to the heavyweight processor and system emulation, i.e., instructions executed on the host processors are mostly QEMU instructions for the emulation, rather than application instructions. However, improving performance bottleneck of virtualization-based techniques used in our approach is out of the scope of this work. We treat this part of efforts as potential future work.

As elaborated in Section 4.2, the voltage scaling technique used in our approach is fulfilled using FiNTV, by directly modifying the core voltage model-specific register, in which the frequency of processors remains the same and thus it has no performance penalty for HPC runs. Moreover, this FiNTV approach is conducted offline, not during HPC runs, and thus it has no runtime overhead. In our experiments and previous studies [12,20], the offline overhead of FiNTV is negligible within the application performance loss shown in Fig. 5. Although the overhead of fault injection will go away on a real setup where errors actually occur and no fault injection is needed, the evaluation is also helpful to discuss: We propose our approach `V-Power` as a prototype virtualization-based framework for saving the maximum power for inherently resilient applications, and this framework could be reused/rewritten by researchers who are interested in similar research directions, given that aggressive voltage scaling is practically not enabled for production processors nowadays.

5. Related work

In general, resilience and power/energy efficiency are mutually constrained trading-off dimensions in HPC. It has been widely studied that there exists entangled interplay between the two dimensions if either is improved/degraded. Existing efforts explore different strategies at various layers of abstraction to leverage inherent resilience of applications for performance efficiency and power/energy savings, Examples include the use of inexact hardware [3,4], voltage scaling [5,6], load value approximation [7,8], and task skipping [9,10]. Next we detail the efforts that focus on improving system-wide power/energy efficiency, where power/energy savings are not completely and necessarily due to performance efficiency.

Inexact hardware: Targeting image filtering and compression applications, Kulkarni et al. [3] proposed a multiplier with a modified inaccurate 2×2 building block, which reduces the area by half and achieves up to 45.4% power savings with an average error rate of 2.36%. They also demonstrated the strength of this circuit-centric approach for power-quality trade-offs with a software-level approach, and enhanced their design for non error-resilient applications. Ganapathy et al. [4] exploited tolerable imprecision of inherently resilient applications by developing a fault-mitigation scheme for unreliable data memories. A bit-shuffling mechanism was employed to isolate errors into bit locations with lower significance, and tolerate limited number of faults. Compared to ECC memory, the proposed technique saved up to 83% in read power, 77% in read access time, and 89% in area for data mining applications. Our virtualization-based approach differs from these hardware solutions, since our non-intrusive approach requires no modification to applications, OS, or hardware, running alongside as a daemon process with no interference to other applications as well. Moreover, power savings from these hardware approaches primarily come from area reduction, not from frequency/voltage scaling.

Voltage scaling: Generally, frequency/voltage reduction of hardware components decreases their power/energy consumption, but increases failure rates of the components as a trade-off. A few studies investigated the trade-off while leveraging the inherent resilience of applications. Chippa et al. [5] proposed a cross-layer scalable hardware design at different levels of abstraction, i.e., circuit, architecture, and algorithm. Different scaling techniques including voltage scaling were utilized for energy savings while maintaining an acceptable output quality. The implemented processor based on the proposed approach was capable of saving energy up to 5x with negligible quality loss, and up to $50\times$ with moderate quality loss. Rahimi et al. [6] devised an innovative architectural solution named approximate associative memristive memory that is able to tolerate timing errors from undervolting for GPU multimedia applications. Experimental results on an AMD GPU demonstrated 32% average energy savings, while delivering an acceptable output quality. Previous work [49] proposed a combination of compilation and micro-architectural design that integrates a module of associative memristive memory with floating point units. The simulated hardware enabled 39% energy savings with enhanced resilience against timing errors, compared to undervolting. Similarly leveraging application inherent resilience, our undervolting approach is however applied to virtualization-based platforms, and is application/OS/architecture transparent.

Interplay benchmarking and modeling: Bacha et al. [12] conducted the first undervolting work based on firmware on real machines protected by ECC memory. ECC errors resulting from dynamic voltage reduction were tolerated by the ECC mechanism. Using pre-production processors running CPU intensive workloads, voltage was scaled to V_{safe_min} level per core individually, and power savings achieved ranged from 18% to 23% with negligible performance loss. Leveraging mainstream resilience techniques, Tan et al. [20] proposed an emulated scaling method for undervolting production machines in HPC, where increased number of soft and hard errors were tolerated by the resilience techniques. This approach was evaluated to save up to 12.1% energy and save 9.1% more energy than an advanced DVFS solution. Subsequent work [50] discussed the impacts of HPC parameters in energy efficiency and resilience at scale, and built theoretical energy-resilience models to investigate the interplay, based on the Amdahl's Law and the Karp–Flatt Metric. Results on two power-aware clusters showed that the proposed models were accurate and effective to find the balanced HPC configuration for the optimal scalable energy efficiency with resilience. Targeting virtualization-based environment, our work focuses on exploit inherent resilience of applications for power efficiency without hardware/software fault tolerance techniques applied, where near-threshold voltage reduction is employed online with quality metrics of applications satisfied.

6. Conclusions

System-wide resilience and power efficiency are two crucial and demanding requirements for current and future supercomputers. Although trading off resilience for power efficiency has been extensively studied in HPC community, there exists a lack of empirical systems that are capable of analyzing intrinsic nature of resilience of applications efficiently, and leveraging the inherent application resilience for quantitative power savings to the maximum extent. Based on virtualization and fault injection techniques, we propose and develop a power saving framework for inherently resilient applications, using near-threshold voltage reduction independent of frequency scaling. With fine-grained fault and power models employed, our systematic approach is evaluated on a power-aware production server to save power significantly with negligible loss of output quality for a wide scope of mainstream scientific applications, provided application-specific quality metrics.

References

[1] N. Wang, M. Fertig, S. Patel, Y-branches: When you come to a fork in the road, take it, in: Proceedings PACT, 2003, pp. 56–66.
[2] S. Mittal, A survey of techniques for approximate computing, ACM Comput Surv 48 (4) (2016) 62.

[3] P. Kulkarni, P. Gupta, M. Ercegovac, Trading accuracy for power with an underdesigned multiplier architecture, in: Proceedings Internatioal Conference on VLSI Design, 2011, pp. 346–351.
[4] S. Ganapathy, G. Karakonstantis, A. Teman, A. Burg, Mitigating the impact of faults in unreliable memories for error-resilient applications, in: Proceedings DAC, 2015, p. 102.
[5] V.K. Chippa, D. Mohapatra, K. Roy, S.T. Chakradhar, A. Raghunathan, Scalable effort hardware design, IEEE Trans. Very Large Scale Integ. Syst. 22 (9) (2014) 2004–2016.
[6] A. Rahimi, A. Ghofrani, K.-T. Cheng, L. Benini, R.K. Gupta, Approximate associative memristive memory for energy-efficient GPUs, in: Proceedings DATE, 2015, pp. 1497–1502.
[7] J.S. Miguel, M. Badr, N.E. Jerger, Load value approximation, in: Proceedings MICRO, 2014, pp. 127–139.
[8] M. Sutherland, J.S. Miguel, N.E. Jerger, Texture cache approximation on GPUs, in: Proceedings Workshop on Approximate Computing Across the Stack, 2015.
[9] M. Samadi, D.A. Jamshidi, J. Lee, S. Mahlke, Paraprox: pattern-based approximation for data parallel applications, in: Proceedings ASPLOS, 2014, pp. 35–50.
[10] Í. Goiri, R. Bianchini, S. Nagarakatte, T.D. Nguyen, Approxhadoop: bringing approximations to MapReduce frameworks, in: Proceedings ASPLOS, 2015, pp. 383–397.
[11] H. Kaul, M. Anders, S. Hsu, A. Agarwal, R. Krishnamurthy, S. Borkar, Near-threshold voltage (NTV) design – opportunities and challenges, in: Proceedings DAC, 2012, pp. 1153–1158.
[12] A. Bacha, R. Teodorescu, Dynamic reduction of voltage margins by leveraging on-chip ECC in Itanium II processors, in: Proceedings ISCA, 2013, pp. 297–307.
[13] A. Miyoshi, C. Lefurgy, E.V. Hensbergen, R. Rajamony, R. Rajkumar, Critical power slope: understanding the runtime effects of frequency scaling, in: Proceedings ICS, 2002, pp. 35–44.
[14] B. Rountree, et al., Bounding energy consumption in large-scale MPI programs, in: Proceedings SC, 2007, pp. 1–9.
[15] B. Rountree, et al., Adagio: Making DVS practical for complex HPC applications, in: Proceedings ICS, 2009, pp. 460–469.
[16] C. Liu, J. Li, W. Huang, J. Rubio, E. Speight, X. Lin, Power-efficient time-sensitive mapping in heterogeneous systems, in: Proceedings PACT, 2012, pp. 23–32.
[17] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, Z. Zong, Effects of dynamic voltage and frequency scaling on a K20 GPU, in: Proceedings PASA, 2013, pp. 826–833.
[18] H. David, C. Fallin, E. Gorbatov, U.R. Hanebutte, O. Mutlu, Memory power mamagement via dynamic voltage/frequency scaling, in: Proceedings ICAC, 2011, pp. 31–40.
[19] Q. Deng, D. Meisner, L. Ramos, T.F. Wenisch, R. Bianchini, Memscale: active low-power modes for main memory, in: Proceedings ASPLOS, 2011, pp. 225–238.
[20] L. Tan, S.L. Song, P. Wu, Z. Chen, R. Ge, D.J. Kerbyson, Investigating the interplay between energy efficiency and resilience in high performance computing, in: Proceedings IPDPS, 2015, pp. 786–796.
[21] C. Wilkerson, H. Gao, A.R. Alameldeen, Z. Chishti, M. Khellah, S.-L. Lu, Trading off cache capacity for reliability to enable low voltage operation, in: Proceedings ISCA, 2008, pp. 203–214.
[22] C.-H. Ho, Mechanisms and evaluation of cross-layer fault-tolerance for supercomputing, in: Proceedings ICPP, 2012, pp. 510–519.
[23] A. Yazdanbakhsh, Axilog: language support for approximate hardware design, in: Proceedings DATE, 2015, pp. 812–817.
[24] L. Tan, S.R. Kothapalli, L. Chen, O. Hussaini, R. Bissiri, Z. Chen, A survey of power and energy efficient techniques for high performance numerical linear algebra operations, Parallel Comput. 40 (10) (2014) 559–573.
[25] CPUFreq - CPU frequency scaling, https://wiki.archlinux.org/index.php/CPU_Frequency_Scaling.
[26] L. Tan, Z. Chen, Slow down or halt: Saving the optimal energy for scalable HPC systems, in: Proceedings ICPE, 2015, pp. 241–244.
[27] J. Chen, L. Tan, P. Wu, D. Tao, H. Li, X. Liang, S. Li, R. Ge, L.N. Bhuyan, Z. Chen, GreenLA: green linear algebra software for GPU-accelerated heterogeneous computing, in: Proceedings SC, 2016, p. 57.
[28] Q. Guan, N. DeBardeleben, S. Blanchard, S. Fu, F-SEFI: A fine-grained soft error fault injection tool for profiling application vulnerability, in: Proceedings IPDPS, 2014, pp. 1245–1254.
[29] F. Bellard, Qemu, a fast and portable dynamic translator, in: Proceedings USENIX ATC, 2005, p. 41.
[30] S.L. Graham, P.B. Kessler, M.K. Mckusick, Gprof: a call graph execution profiler, in: Proceedings CC, 1982, pp. 120–126.
[31] Intel® 64 and IA-32 architectures software developer manuals, http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html, 2016.
[32] F. Ayatolahi, B. Sangchoolie, R. Johansson, J. Karlsson, A study of the impact of single bit-flip and double bit-flip errors on program execution, in: Proceedings SafeComp, 2013, pp. 265–276.
[33] N. DeBardeleben, S. Blanchard, Q. Guan, Z. Zhang, S. Fu, Experimental framework for injecting logic errors in a virtual machine to profile applications for soft error resilience, in: Proceedings Euro-Par, 2011, pp. 282–291.
[34] H. Song, J.R. Cruz, Reduced-complexity decoding of Q-ary LDPC codes for magnetic recording, IEEE Trans. Magn. 39 (2) (2003) 1081–1087.
[35] M. Snir, R.W. Wisniewski, J.A. Abraham, S.V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A.A. Chien, P. Coteus, N.A. Debardeleben, P.C. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, E.V. Hensbergen, Addressing failures in exascale computing, Int. J. High Perform. Comput. Appl. 28 (2) (2014) 129–173.
[36] S.M. Shatz, J.-P. Wang, Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems, IEEE Trans. Reliab. 38 (1) (1989) 16–27.
[37] Y. Zhang, K. Chakrabarty, V. Swaminathan, Energy-aware fault tolerance in fixed-priority real-time embedded systems, in: Proceedings ICCAD, 2003, pp. 209–213.
[38] D. Zhu, R. Melhem, D. Mossé, The effects of energy management on reliability in real-time embedded systems, in: Proceedings ICCAD, 2004, pp. 35–40.
[39] A. Kansal, F. Zhao, J. Liu, N. Kothari, A.A. Bhattacharya, Virtual machine power metering and provisioning, in: Proceedings SoCC, 2010, pp. 39–50.
[40] M. Colmant, M. Kurpicz, P. Felber, L. Huertas, R. Rouvoy, A. Sobe, Process-level power estimation in VM-based systems, in: Proceedings EuroSys, 2015, p. 14.
[41] C.-H. Hsu, W.-C. Feng, A power-aware run-time system for high-performance computing, in: Proceedings SC, 2005, p. 1.
[42] The Princeton application repository for shared-memory computers (PARSEC), http://parsec.cs.princeton.edu/index.htm.
[43] MediaBench II Benchmark, http://euler.slu.edu/~fritts/mediabench/.
[44] NASA advanced supercomputing parallel benchmarks (NPB), http://www.nas.nasa.gov/publications/npb.html.
[45] Intel® Xeon® processor E5-2660 v3 specifications, http://ark.intel.com/products/81706/Intel-Xeon-Processor-E5-2660-v3-25M-Cache-2_60-GHz, 2014.
[46] Intel Itanium 2 9560 vs Xeon E5-2699 v3, http://www.cpu-world.com/Compare/458/Intel_Itanium_2_9560_vs_Intel_Xeon_E5-2699_v3.html.
[47] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, G. Favor, K. Sankaran, S. Das, A system-level voltage/frequency scaling characterization framework for multicore CPUs, in: Proceedings SELSE, 2017.
[48] Q. Guan, N. DeBardeleben, P. Wu, S. Eidenbenz, S. Blanchard, L. Monroe, E. Baseman, L. Tan, Design, use and evaluation of P-FSEFI: a parallel soft error fault injection framework for emulating soft errors in parallel applications, in: Proceedings SIMUTOOLS, 2016, pp. 9–17.
[49] A. Rahimi, A. Ghofrani, M. Angel, K.-T. Cheng, L. Benini, R.K. Gupta, Energy-efficient GPGPU architectures via collaborative compilation and memristive memory-based computing, in: Proceedings DAC, 2014, pp. 1–6.
[50] L. Tan, Z. Chen, S.L. Song, Scalable energy efficiency with resilience for high performance computing systems: a quantitative methodology, ACM Trans. Archit. Code Optim. 12 (4) (2016) 35.