Investigating the Interplay between Energy Efficiency and Resilience in High Performance Computing

Li Tan*, Shuaiwen Leon Song†, Panruo Wu*, Zizhong Chen*, Rong Ge‡, Darren J. Kerbyson†
*University of California, Riverside
{ltan003, pwu011, chen}@cs.ucr.edu
†Pacific Northwest National Laboratory
{Shuaiwen.Song, Darren.Kerbyson}@pnnl.gov
‡Marquette University
rong.ge@marquette.edu

***Abstract*—Energy efficiency and resilience are two crucial challenges for High Performance Computing (HPC) systems to reach exascale. While energy efficiency and resilience issues have been extensively studied individually, little has been done to understand the interplay between energy efficiency and resilience for HPC systems. Decreasing the supply voltage associated with a given operating frequency for processors and other CMOS-based components can significantly reduce power consumption. However, this often raises system failure rates and consequently increases application execution time. In this work, we present an energy saving *undervolting* approach that leverages the mainstream resilience techniques to tolerate the increased failures caused by undervolting. Our strategy is directed by analytic models, which capture the impacts of undervolting and the interplay between energy efficiency and resilience. Experimental results on a power-aware cluster demonstrate that our approach can save up to 12.1% energy compared to the baseline, and conserve up to 9.1% more energy than a state-of-the-art DVFS solution.**

***Keywords*-energy; resilience; failures; undervolting; HPC.**

I. Introduction

Energy efficiency is one of the crucial challenges that must be addressed for High Performance Computing (HPC) systems to achieve ExaFLOPS (10^{18} FLOPS). The average power of the top five supercomputers worldwide has been inevitably growing to 10.1 MW today according to the latest TOP500 list [1], sufficient to power a city with a population of 20,000 people [2]. The 20 MW power-wall, set by the US Department of Energy [3] for exascale computers, indicates the severity of how energy budget constrains the performance improvement required by the ever-growing computational complexity of HPC applications.

Lowering operating frequency and/or supply voltage of hardware components, i.e., DVFS (Dynamic Voltage and Frequency Scaling [4]), is one important approach to reduce power and energy consumption of a computing system for two primary reasons: First, CMOS-based components (e.g., CPU, GPU, and memory) are the dominant power consumers in the system. Second, power costs of these components are proportional to the product of operating frequency and supply voltage squared (shown in Figure 1). In general, supply voltage has a positive correlation with (not strictly proportional/linear to) the operating frequency for DVFS-capable components [5], i.e., scaling up/down frequency results in voltage raise/drop accordingly.

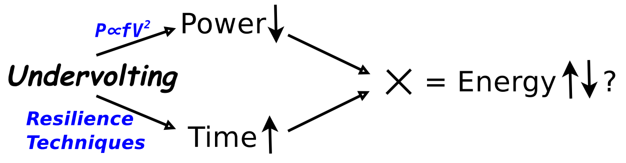


Figure 1. Entangled Effects of Undervolting on Performance, Energy, and Resilience for HPC Systems in General.

Nevertheless, existing DVFS techniques are essentially frequency-directed and fail to fully exploit the potential power reduction and energy savings. With DVFS, voltage is only lowered to comply with the frequency reduction in the presence of "slack" [6], i.e., idle time from hardware components (typically processors) during an HPC run. For a given frequency, cutting-edge hardware components can be supplied with a voltage that is lower than the one paired with the given frequency. The enabling technique, *undervolting* [7] [8] [9] is independent of frequency scaling, i.e., lowering only supply voltage of a chip without reducing its operating frequency. Undervolting is advantageous in the sense that: (a) It can keep the component frequency unchanged such that the computation throughput is well maintained, and (b) it can be uniformly applied to both "slack" and "non-slack" phases of HPC runs for power reduction.

The challenge of employing undervolting as a general power saving technique in HPC lies in efficiently addressing the increasing failure rates caused by it. Both hard and soft errors may occur if components undergo undervolting. Several studies have investigated architectural solutions to support reliable undervolting with simulation [7] [8]. The study by Bacha *et al.* [9] presented an empirical undervolting system on Intel Itanium II processors that resolves the arising Error-Correcting Code (ECC) memory faults yet improves the overall energy savings. While this work aims to maximize the power reduction and energy savings, it relies on pre-production processors that allow such thorough exploration on the undervolting schemes, and also requires additional hardware support for the ECC memory.

In this work, we investigate the interplay (shown in Figure

1) between energy efficiency and resilience for large-scale HPC systems, and demonstrate theoretically and empirically that significant energy savings can be obtained using a combination of undervolting and mainstream software-level resilience techniques on today's HPC systems, without requiring hardware redesign. We aim to explore if the future exascale systems are going towards the direction of low-voltage embedded architectures in order to guarantee energy efficiency, or they can rely on advanced software-level techniques to achieve high system resilience and efficiency. In summary, the contributions of this paper include:

- We propose an energy saving undervolting approach for HPC systems by leveraging resilience techniques;
- Our technique does not require pre-production machines and makes no modification to the hardware;
- We formulate the impacts of undervolting on failure rates and energy savings for mainstream resilience techniques, and model the conditions for energy savings;
- Our approach is experimentally evaluated to save up to 12.1% energy compared to the baseline runs of the 8 HPC benchmarks, and conserve up to 9.1% more energy than a state-of-the-art frequency-directed energy saving solution.

The remainder of the paper is organized as follows: Section II discusses the related work. We theoretically model the problem in Section III, and Section IV presents the details of our experimental methodology. Results and their evaluation are provided in Section V and Section VI concludes.

II. Related Work

To the best of our knowledge, this work is the first of its kind that models and discusses the interplay between energy efficiency and resilience at scale. Most of the related efforts have been conducted in the following areas:

Real-Time/Embedded Processors and Systems-on-Chip: Extensive research has been performed to save energy and preserve system reliability for real-time embedded processors and systems-on-chip. Zhu *et al.* [10] discussed the effects of energy management via frequency and voltage scaling on system failure rates. This work is later extended to reliability-aware energy saving scheduling that allocates slack for multiple real-time tasks [11], and a generalized Standby-Sparing technique for multiprocessor real-time systems, considering both transient and permanent faults [12]. These studies made some assumptions suitable for real-time embedded systems, but not applicable to large-scale HPC systems with complex hardware and various types of faults. Pop *et al.* [13] explored heterogeneity in distributed embedded systems and developed a logic programming solution to identify a reliable scheduling scheme that saves energy. This work ignored runtime process communication, which is an important factor of performance and energy efficiency for HPC systems and applications. The Razor work [14] implemented a prototype 64-bit Alpha processor design that combines circuit and architectural techniques for low-cost speed path error detection/correction from operating at a lower supply voltage. With minor error recovery overhead, substantial energy savings can be achieved while guaranteeing correct operations of the processor. Our work differs from Razor since we consider hard/soft errors in HPC runs due to undervolting. Razor power/energy costs were simulated at circuit level while our results were obtained from real measurements in an HPC environment at cluster level. Similar power-saving and resilient-against-error hardware techniques have been proposed such as Intel's Near-Threshold Voltage (NTV) design [15] on a full x86 microprocessor.

Memory Systems: As ECC memory prevails, numerous studies have explored energy efficient architectures and error-detection techniques in memory systems. Wilkerson *et al.* [7] proposed to trade off cache capacity for reliability to enable low-voltage operations on L1 and L2 caches to reduce power. Their subsequent work [8] investigated an energy saving cache architecture using variable-strength ECC to minimize latency and area overhead. Unlike our work that is evaluated with real HPC systems and physical energy measurements, they used average instructions per cycle and voltage to estimate energy costs. Bacha *et al.* [9] employed a firmware-based voltage scaling technique to save energy for a pre-production multicore architecture, under increasing fault rates that can be tolerated by ECC memory. Although our work similarly scales down voltage with a fixed frequency to save power, it is different in two aspects: Ours targets *general faults* on common *HPC* production machines at scale, while theirs specifically handles *ECC errors* on a *pre-production* architecture. To balance performance, power, and resilience, Li *et al.* [16] proposed an ECC memory system that can adapt memory access granularity and several ECC schemes to support applications with different memory behaviors. Liu *et al.* [17] developed an application-level technique for smartphones to reduce the refresh power in DRAM at the cost of a modest increase in non-critical data corruption, and empirically showed that such errors have few impacts on the final outputs. None of the above approaches analytically model the entangling effects of energy efficiency and resilience at scale like ours.

III. Problem Description and Modeling

A. Failure Rate Modeling with Undervolting

Failures in a computing system can have multiple root causes, including radiation from the cosmic rays and packaging materials, frequency/voltage scaling, and temperature fluctuation. There are two types of induced faults by nature: soft errors and hard errors. The former are transient (e.g., memory bit-flips and logic circuit errors) while the latter are usually permanent (e.g., node crashes from dysfunctional hardware and system abort from power outage). Soft errors are generally hard to detect (e.g., silent data corruption) since applications are typically not interrupted by such errors, while hard errors do not silently occur, causing outputs inevitably partially or completely lost. Note that here we use the terms *failure*, *fault*, and *error* interchangeably. In this work, we aim to theoretically and empirically study if undervolting with a fixed frequency (thus fixed throughput) is able to reduce the overall energy consumption. We study the interplay between the power reduction through undervolting and application performance loss due to the required fault detection and recovery at the raised failure rates from undervolting. Moreover, we consider the overall failure rates from both soft and hard errors, as the failure rates of either type can increase due to undervolting.

Assume that the failures of combinational logic circuits follow a Poisson distribution, and the average failure rate is determined by the operating frequency and supply voltage [18] [10]. We employ an existing exponential model of average failure rate λ in terms of operating frequency f [10] without normalizing supply voltage V_{dd}, where λ_0 is the average failure rate at the maximum frequency f_{max} and voltage V_{max} ($f_{min} \leq f \leq f_{max}$ and $V_{min} \leq V_{dd} \leq V_{max}$):

$$\lambda(f, V_{dd}) = \lambda(f) = \lambda_0 \; e^{\frac{d(f_{max}-f)}{f_{max}-f_{min}}} \tag{1}$$

Exponent d is an architecture-dependent constant, reflecting the sensitivity of failure rate variation with frequency scaling. Previous work [19] modeled the relationship between operating frequency and supply voltage as follows:

$$f = \varphi(V_{dd}, V_{th}) = \beta \; \frac{(V_{dd}-V_{th})^2}{V_{dd}} \tag{2}$$

β is a hardware-related constant, and V_{th} is the threshold voltage. Substituting Equation (2) into Equation (1) yields:

$$\lambda(f, V_{dd}) = \lambda(V_{dd}) = \lambda_0 \; e^{\frac{d(f_{max}-\beta(V_{dd}-2V_{th}+\frac{V_{th}^2}{V_{dd}}))}{f_{max}-f_{min}}} \tag{3}$$

Equation (3) indicates that the average failure rate is a function of supply voltage V_{dd} only, provided that V_{th} and other parameters are fixed. This condition holds true when undervolting is studied in this work. Denote $\sigma(V_{dd}) = \frac{d(f_{max}-\beta(V_{dd}-2V_{th}+\frac{V_{th}^2}{V_{dd}}))}{f_{max}-f_{min}}$. We calculate the first derivative of $\lambda(f, V_{dd})$ (Equation (3)) with respect to V_{dd} and identify values of V_{dd} that make the first derivative zero as follows:

$$\frac{\partial \lambda}{\partial V_{dd}} = \lambda_0 \; e^{\sigma(V_{dd})} \; \frac{-d\beta(1-(V_{th}/V_{dd})^2)}{f_{max}-f_{min}} = 0 \tag{4}$$

$$\Rightarrow V_{dd} = \pm V_{th} \tag{5}$$

Therefore, for $-V_{th} < V_{dd} < V_{th}$, $\lambda(f, V_{dd})$ monotonically strictly increases as V_{dd} increases; for $V_{dd} \leq -V_{th}$ and $V_{dd} \geq V_{th}$, $\lambda(f, V_{dd})$ monotonically strictly decreases as V_{dd} increases. Given that empirically $V_{dd} \geq V_{th}$, we conclude that $\lambda(f, V_{dd})$ is a monotonically strictly decreasing function for all valid V_{dd} values. $\lambda(f, V_{dd})$ maximizes at $V_{dd} = V_{th}$.

EXAMPLE. Figure 2 shows the comparison between the experimentally observed and theoretically calculated failure rates with regard to supply voltage for a pre-production processor, where the observed data is from [9] and the calculated data is via Equation (3). As shown in the figure, the calculated values are very close to the observed ones, which demonstrates that Equation (3) can be used to model failure rate. Based on the voltage parameters from the vendor [20], we denote several significant voltage levels in Figure 2, where V_h and V_l refer to the the maximum and minimum supply voltages for a production processor. They also pair with the maximum and minimum operating frequencies respectively. In many cases, undervolting is disabled on production processors by vendors based on the assumption that there is no adequate fault tolerance support at the software stack, and thus production processors often shut down when their supply voltage is scaled below V_l. For some pre-production processors [9], supply voltage can be further scaled down to V_{safe_min}, which refers to the theoretical lowest safe supply voltage under which the system can operate without crashing. But even for these pre-production processors, when V_{dd} is actually reduced below V_{safe_min}, they will no longer operate reliably and shut down [9].

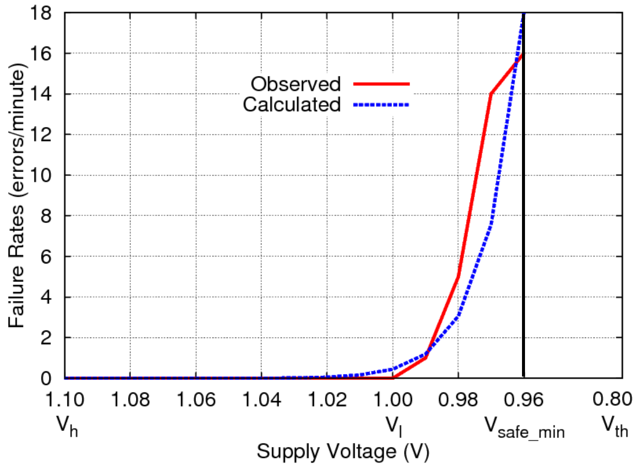


Figure 2. Observed and Calculated Failure Rates λ as a Function of Supply Voltage V_{dd} with a Fixed Frequency for a Pre-production Intel Itanium II 9560 8-Core Processor (Note that the observed failures are ECC memory correctable errors for one core. V_h: the maximum voltage paired with the maximum frequency, V_l: the minimum voltage paired with the minimum frequency, V_{safe_min}: the lowest safe voltage for one core of the pre-production processor, and V_{th}: threshold voltage).

Note that although there are no observed faults in Figure 2 for the voltage range from 1.10 V to 1.00 V, the calculated failure rates are not zero, ranging from 10^{-6} to 10^{-1}. This difference suggests that failures with a small probability do not often occur in real runs. For the voltage range from 0.99 V to 0.96 V, the calculated failure rates match the observation with acceptable statistical errors. The calculated failure rates for voltage levels that are lower than V_{safe_min} are not presented here due to the lack of observational data for comparison. Unless some major circuit-level redesign on the hardware for enabling NTV, we commonly consider the voltage range between V_{safe_min} and V_{th} inoperable in HPC even with sophisticated software-level fault tolerant techniques. Thus, modeling this voltage range is out of the scope of this work.

B. Performance Modeling under Resilience Techniques

Resilience techniques like Checkpoint/Restart (C/R) and Algorithm-Based Fault Tolerance (ABFT) have been widely employed in HPC environments for fault tolerance. State-of-the-art C/R and ABFT techniques are lightweight [21] [22], scalable [23] [22], and sound [24]. C/R is a general resilience technique that is often used to handle hard errors (it can also recover soft errors if errors can be successfully detected). ABFT is more cost-efficient than C/R to detect and correct soft errors. But it is not as general as C/R because it leverages algorithmic knowledge of target programs and thus only works for specific applications. Next we briefly illustrate how they function, and present the formulated performance models of both techniques.

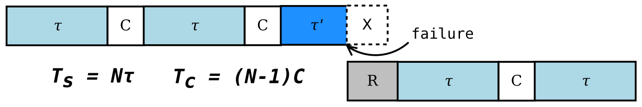


Figure 3. Checkpoint/Restart Execution Model for a Single Process.

A checkpoint is a snapshot of a system state, i.e., a copy of the contents of application process address space, including all values in the stack, heap, global variables, and registers. Classic C/R techniques save checkpoints to disks [25], memory [21], and both disks and memory via multi-level checkpointing [23]. Figure 3 shows how a typical C/R scheme recovers a single-process failure, where we denote checkpoint overhead as C, restart overhead as R, and compute time between checkpoints as τ respectively. An interrupting failure can arise at any given point of an execution. C/R can capture the failure and restart the application from the nearest saved checkpoint with the interrupted compute period re-executed.

Next we present the issue of determining the optimal checkpoint interval, i.e., τ_{opt} that can minimize the total checkpoint and restart overhead, given a failure rate of λ. This is significant since in the scenario of undervolting, failure rates may vary dramatically as shown in Figure 2, which could affect τ_{opt} considerably. Without considering the impacts of undervolting, several efforts on estimating τ_{opt} have been proposed. Young [26] approximated $\tau_{opt} = \sqrt{\frac{2C}{\lambda}}$ as a first-order derivation. Taking restart overhead R into account, Daly [27] refined Young's model into $\tau_{opt} = \sqrt{2C(\frac{1}{\lambda} + R)}$ for $\tau + C \ll \frac{1}{\lambda}$. Using a higher order model, the case that checkpoint overhead C becomes large compared to MTTF (Mean Time To Failure), $\frac{1}{\lambda}$ was further discussed for a perturbation solution in [27]:

$$\tau_{opt} = \begin{cases} \sqrt{\frac{2C}{\lambda}} - C & \text{for } C < \frac{1}{2\lambda} \\ \frac{1}{\lambda} & \text{for } C \geq \frac{1}{2\lambda} \end{cases} \tag{6}$$

Note that R has no contributions in Equation (6) . Since Equation (6) includes the failure rate λ discussed in Equation (3), it is suitable to be used to calculate τ_{opt} in the scenario of undervolting.

Consider the basic time cost function of C/R modeled in [27]: $T_{cr} = T_s + T_c + T_w + T_r$, where T_s refers to the solve time of running an application with no interrupts from failures. The total checkpoint overhead is denoted as T_c. T_w is the amount of time spent on an interrupted compute period before a failure occurs, and T_r is defined as the total time on restarting from failures. Figure 3 shows the case of a single failure, and how the time fractions are accumulated. Next we generalize the time model to accommodate the case of multiple failures in a run as follows:

$$T_{cr} = N\tau + (N-1)C + \phi(\tau + C)n + Rn \tag{7}$$

where N is the number of compute periods and n is the number of failures within a run ($N-1$ is because there is no need for one more checkpoint if the last compute period is completed). ϕ is the percentage of elapsed time in a segment of compute when an interrupt occurs and the process restarts. And we adopt a common assumption that interrupts never occur during a restart. As a constant, $N\tau$ can be denoted as T_s and Equation (7) is reformed as:

$$T_{cr} = T_s + (\frac{T_s}{\tau} - 1)C + \phi(\tau + C)n + Rn \tag{8}$$

We adopt Equation (8) as the C/R time model henceforth. In the absence of failures, this model is simplified with the last two terms omitted. Next we introduce how ABFT works and present its performance model formally.

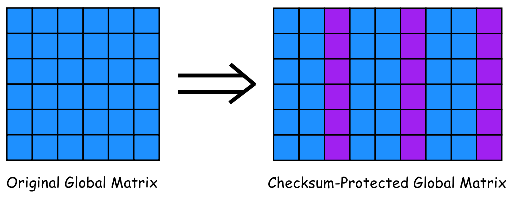


Figure 4. Algorithm-Based Fault Tolerance Model for Matrix Operations.

Figure 4 shows how ABFT protects the matrix operations from soft errors using the row-checksum mechanism. The checksum blocks are periodically maintained in between the original matrix elements by adding an extra column to the process grid. The redundant checksum blocks contain sums of local matrix elements in the same row, which can be used for recovery if one original matrix element within the protection of a checksum block is corrupted. In C/R, checkpoints are periodically saved. Likewise, checksum blocks are periodically updated, generally together with the matrix operations. The difference is that the interval of updating the checksum in ABFT is fixed, i.e., not affected by the variation of failure rates, while in C/R, the optimal checkpoint interval that minimizes C/R overhead highly depends on failure rate λ.

Our previous work [22] has modeled ABFT overhead for the dense matrix factorizations, including Cholesky, LU, and QR factorizations. Here we present the overall performance model for ABFT-enabled dense matrix factorizations, taking Cholesky factorization for example due to algorithmic similarity.

$$T_{abft} = \frac{\mu C_f \mathbb{N}^3}{\mathbb{P}} t_f + \frac{\mu C_v \mathbb{N}^2}{\sqrt{\mathbb{P}}} t_v + \frac{C_m \mathbb{N}}{nb} t_m + T_d + T_l + T_c \quad (9)$$

where $\mathbb{N}$ represents the dimensions of the global matrix, $\mathbb{P}$ is the total number of processes, and nb is the block size for data distribution. $\mu = 1 + \frac{4}{nb}$ is the factor of introducing checksum rows/columns to the global matrix (the actual factorized global matrix size is $\mu\mathbb{N}$). Cholesky-specific constants $C_f = \frac{1}{3}$, $C_v = 2 + \frac{1}{2}\log\mathbb{P}$, and $C_m = 4 + \log\mathbb{P}$. t_f is the time required per FLoating-point OPeration (FLOP), t_v is the time required per data item communicated, and t_m is the time required per message prepared for transmission. Error detection overhead is denoted as $T_d = \frac{\mathbb{N}^2}{\mathbb{P}} t_f$, and error localization overhead is denoted as $T_l = \frac{nb\mathbb{P}}{\mathbb{N}^3} t_f$. Since one error correction operation only requires one FLOP, the error correction overhead can be described as $T_c = nt_f$. Similarly as in the C/R performance model, the error-related overhead is only valid in the presence of failures. Otherwise the last three terms are omitted and the sum of the first three terms represents the performance of running Cholesky with ABFT.

Due to the conceptual similarity and space limitation, we only model the performance of these two resilience techniques in this work.

C. Power and Energy Modeling under Resilience Techniques and Undervolting

In this subsection, we present general power and energy models under a combination of undervolting and resilience techniques. We use C/R as an example for model construction, which can also be applied to other resilience techniques used in this work. Since undervolting will affect the processor power most directly and the processor power has the most potential to be conserved in an HPC system, we assume all the power/energy savings come from processors. Therefore, we focus on modeling processor-level energy. Using the energy models, we can explore in theory what factors will likely affect the overall energy savings through undervolting.

First, we adopt the nodal power model in [28] [29] as follows:

$$\begin{aligned} P &= P^{CPU}_{dynamic} + P^{CPU}_{leakage} + P^{other}_{leakage} \\ &= AC' f V_{dd}^2 + I_{sub} V_{dd} + I'_{sub} V'_{dd} \end{aligned} \quad (10)$$

A and C' are the percentage of active gates and the total capacitive load in a CMOS-based processor; I_{sub} and I'_{sub} are subthreshold leakage current of CPU and non-CPU components, and V_{dd} and V'_{dd} are their supply voltages. We denote $I'_{sub}V'_{dd}$ as a constant P_c since this term is independent of undervolting. Previous work [30] indicates that the power draw of a node during C/R is very close to its idle mode. Therefore, by leveraging this power characteristic for C/R phases and different levels of voltages in Figure 2, we propose the following power formulae for a given node:

$$\begin{cases} P_h &= AC' f_h V_h^2 + I_{sub} V_h + P_c \\ P_m &= AC' f_h V_{safe\_min}^2 + I_{sub} V_{safe\_min} + P_c \\ P_l &= AC' f_l V_{safe\_min}^2 + I_{sub} V_{safe\_min} + P_c \end{cases} \quad (11)$$

In this Equation, both P_h and P_m use the highest frequency, while P_l scales frequency to the minimum; both P_m and P_l exploit V_{safe_min} to save energy. Here are the scenarios where P_h, P_m, and P_l are mapped into: For the baseline case, we employ P_h to all execution phases of an HPC run. For the energy-optimized case, we apply undervolting to different phases based on their characteristics using P_m and P_l, in order to achieve the optimal energy savings through leveraging resilience techniques. Specifically, without harming the overall performance, we apply P_l to the frequency-insensitive phases including *non-computation* (e.g., *communication*, *memory* and *disk accesses*) and *C/R* phases, while P_m is applied in all the *computation* phases. Using Equations (8) and (11), we can model the energy costs of a baseline run (E_{base}), a run with undervolting but in the absence of failures ($E_{uv}^{\overline{err}}$), and a run with undervolting in the presence of failures (E_{uv}) as:

$$\begin{cases} E_{base} &= P_h T_s \\ E_{uv}^{\overline{err}} &= P_m T_s + P_l(\frac{T_s}{\tau} - 1)C \\ E_{uv} &= P_m(T_s + \phi\tau n) + P_l\left(\left(\frac{T_s - \tau}{\tau} + \phi n\right)C + Rn\right) \end{cases} \quad (12)$$

For processor architectures equipped with a range of operational frequencies, frequency-directed DVFS techniques [4] [6] have been widely applied in HPC for energy saving purposes. Commonly, they predict and apply appropriate frequencies for different computational phases based on workload characteristics. Meanwhile, for the selected frequency (or two frequencies in the case of split frequencies [6]) f_m ($f_l < f_m < f_h$), a paired voltage V_m ($V_l < V_m < V_h$) will also be applied accordingly. One important question emerges: can we further save energy beyond these DVFS techniques by continuing undervolting V_m? To answer this question, we compare our approach with a state-of-the-art DVFS technique named Adagio [6] as an example to demonstrate the potential energy savings from undervolting beyond DVFS. Basically, Adagio runs aside with HPC applications, identifies computation and communication slack, and then apply appropriately reduced frequencies accordingly to save energy without sacrificing the overall performance. We will use the frequencies predicted by Adagio for each phase but further reduce V_m. Therefore, we have:

$$\begin{cases} P^{slack}_{Adagio} &= AC' f_m V_m^2 + I_{sub} V_m + P_c \\ P^{non-slack}_{Adagio} &= P_h \\ P^{slack}_{uv} &= AC' f_m V_{safe\_min}^2 + I_{sub} V_{safe\_min} + P_c \\ P^{non-slack}_{uv} &= P_m \end{cases} \quad (13)$$

$$E_{Adagio} = P^{slack}_{Adagio} T_{slack} \oplus P^{non-slack}_{Adagio} T_s \quad (14)$$

P_{slack} and $P_{non-slack}$ denote the average power during the slack and non-slack phases respectively; T_{slack} is the slack duration due to task dependencies and generally overlaps with T_s across processes [28] (thus we use $\oplus$ instead of $+$). Assume there exists a percentage η ($0 < \eta < 1$) of the total slack that overlaps with the computation. Therefore we define $\oplus$ by explicitly summing up different energy costs:

$$\begin{aligned} &P_1 T_{slack} \oplus P_2 T_s = \\ &\quad P_1 T_{slack}(1-\eta) + P_{hybrid} T_{slack}\eta + P_2 T_s \end{aligned} \tag{15}$$

where P_1 and P_2 denote the nodal power during the slack and computation respectively, and P_{hybrid} ($P_1 < P_{hybrid} < P_2$) is the average nodal power when slack overlaps the computation. Using Equations (12), (13), (14), and (15), we can model the energy consumption E'_{uv}[1] which integrates ($\uplus$) the advantages of both DVFS techniques (Adagio) and the undervolting beyond DVFS, in the presence of slack:

$$E'_{uv} = E_{Adagio} \uplus E_{uv} = P_{uv}^{slack} T_{slack} \oplus E_{uv} \tag{16}$$

Potential energy savings through appropriate undervolting over a baseline run and an Adagio-enabled run can then be calculated as follows:

$$\begin{aligned} \Delta E_1 &= E_{base} - E'_{uv} \\ &= (P_h - P_m)T_s \oplus (P_h - P_{uv}^{slack})T_{slack} - \\ &\quad \left(P_m \phi \tau n + P_l\left(\left(\frac{T_s-\tau}{\tau} + \phi n\right)C + Rn\right)\right) \end{aligned} \tag{17}$$

$$\begin{aligned} \Delta E_2 &= E_{Adagio} - E'_{uv} \\ &= (P_h - P_m)T_s \oplus (P_{Adagio}^{slack} - P_{uv}^{slack})T_{slack} - \\ &\quad \left(P_m \phi \tau n + P_l\left(\left(\frac{T_s-\tau}{\tau} + \phi n\right)C + Rn\right)\right) \end{aligned} \tag{18}$$

DISCUSSION. From Equations (17) and (18), we can observe that the potential energy savings from undervolting is essentially the difference between two terms: Energy savings gained from undervolting (denoted as E_+) and energy overhead from fault detection and recovery due to the increasing errors caused by reduced voltage (denoted as E_-). In other words, the trade-off between the power savings through undervolting and performance overhead for coping with the higher failure rates determines if and how much energy can be saved overall. In E_+, two factors are significant: V_{safe_min} and T_{slack}. They impact the maximum energy savings and generally depend on chip technology and application characteristics. In E_-, three factors are essential: n, C, and R, where n, the number of failures in a run, is affected by undervolting directly. C and R rely on the resilience techniques being employed. Some state-of-the-art resilience techniques (e.g., ABFT) have relatively low values of C and R [24], which can be beneficial to more energy savings.

[1]Since here we treat all cores uniformly during undervolting, i.e., all cores undervolt simultaneously to the same supply voltage, we do not have number of cores as a parameter in our model. But it can be modeled for more complex scenarios.

Here we showcase how to use the above models to explore the energy saving capability of undervolting. We apply the parameters from our real system setup shown in Table I into Equation (17). In order to investigate the relationship among C, R, λ, and the overall energy savings, we let Equation (17) > 0 and solve the inequation with appropriate assumptions.

Using the HPC setup in Table I, we have $f_h = 2.5$, $f_l = 0.8$ (we assume $f_m = 1.8$ for Adagio), $V_h = 1.3$, $V_l = 1.1$, and $V_{safe\_min} = 1.025$. We solve $AC' = 20$, $I_{sub} = 5$, and $P_c = 107$, estimate $\phi = \frac{1}{2}$, and adopt $n = \lambda T_s$, $\tau_{opt} = \frac{1}{\lambda}$ (Equation (6)) using the methodology in Section IV. For simplicity, we assume slack does not overlap any computation and $T_{slack} = 0.2T_s$. So $\oplus$ in Equation (17) > 0 becomes $+$. First we consider the case of $C \geq \frac{1}{2\lambda}$. According to Equation (6), let $\tau = \tau_{opt}$ and now we have:

$$\begin{aligned} &33.34375\ T_s + 9.6105\ T_s - 164.65625 \times \frac{1}{2} T_s - \\ &\quad 128.935\left(\left(\lambda T_s - 1 + \frac{1}{2}\lambda T_s\right)C + R\lambda T_s\right) > 0 \end{aligned} \tag{19}$$

It is clear that the sum of the first three terms is negative and the fourth term is positive. Thus in this case no energy savings can be achieved using undervolting. The conclusion continues to stand even if we let $T_{slack} = T_s$ to increase the second term. It is due to the high failure rate that causes the third term overlarge. Next we consider the other case ($\tau_{opt} = \sqrt{\frac{2C}{\lambda}} - C$) in Equation (6). Thus the inequation becomes:

$$\begin{aligned} &33.34375\ T_s + 9.6105\ T_s - 164.65625 \times \frac{1}{2}(\sqrt{2\lambda C} - \lambda C)T_s - \\ &128.935\left(\left(\frac{\lambda T_s}{\sqrt{2\lambda C} - \lambda C} - 1 + \frac{1}{2}\lambda T_s\right)C + R\lambda T_s\right) > 0 \end{aligned} \tag{20}$$

$$\Rightarrow T_s > \frac{c_3 C}{c_3\left(\frac{\lambda C}{\sqrt{2\lambda C} - \lambda C} + \frac{1}{2}\lambda C + R\lambda\right) - c_1 + c_2(\sqrt{2\lambda C} - \lambda C)} \tag{21}$$

where $c_1 = 42.95425$, $c_2 = 82.328125$, and $c_3 = 128.935$

If the above condition (21) is satisfied, energy savings can be achieved. The condition of T_s can be also reformed into an inequation of C or R in terms of T_s and λ, without losing generality. Using Equation (3), we can also substitute λ with an expression of V_{dd}. Recall that it needs to meet $C < \frac{1}{2\lambda}$ to yield the case $\tau_{opt} = \sqrt{\frac{2C}{\lambda}} - C$.

MODEL RELAXATION. We notice that the relationship among C, R, and λ can be obtained without T_s, if the C/R performance model (see Equation (7)) is relaxed. In the scenario of undervolting, τ_{opt} is comparatively small and thus the number of checkpoints $N = \frac{T_s}{\tau_{opt}}$ is large, due to the high failure rates λ. Therefore, we consider $N - 1 \approx N$ in Equation (7), and the term -1 in Inequation (20) can be ignored, by which T_s in the inequation can be eliminated. Consequently, Inequation (20) can be relaxed into:

Table I
HARDWARE CONFIGURATION FOR ALL EXPERIMENTS.

Cluster	HPCL
System Size	64 Cores, 8 Compute Nodes
Processor	AMD Opteron 2380 (Quad-core)
CPU Frequency	0.8, 1.3, 1.8, 2.5 GHz
CPU Voltage	1.300, 1.100, 1.025, 0.850 V ($V_h/V_l/V_{safe\_min}/V_{th}$)
Memory	8 GB RAM
Cache	128 KB L1, 512 KB L2, 6 MB L3
Network	1 GB/s Ethernet
OS	CentOS 6.2, 64-bit Linux kernel 2.6.32
Power Meter	PowerPack

$$33.34375+9.6105-164.65625\times\frac{1}{2}(\sqrt{2\lambda C}-\lambda C)-128.935\left(\left(\frac{\lambda}{\sqrt{2\lambda C}-\lambda C}+\frac{1}{2}\lambda\right)C+R\lambda\right)>0 \qquad (22)$$

$$\Rightarrow R<\frac{c_1}{c_3\lambda}-\frac{c_2}{c_3}\left(\sqrt{\frac{2C}{\lambda}}-C\right)-\left(\frac{1}{\sqrt{2\lambda C}-\lambda C}+\frac{1}{2}\right)C \qquad (23)$$

In the relaxed performance and energy models, energy savings can be fulfilled as long as the above relationship (23) holds. Again it is required that $C < \frac{1}{2\lambda}$ in order to yield $\tau_{opt} = \sqrt{\frac{2C}{\lambda}} - C$, likewise in the non-relaxed models.

IV. EXPERIMENTAL METHODOLOGY

A. Experimental Setup and Benchmarks

Table I lists the hardware configuration of the power-aware cluster used for all experiments. Although the size of the cluster is comparatively small, it is sufficient for the proof of concept of our techniques. For power/energy measurement, PowerPack [31], a framework for profiling and analysis of power/energy consumption of HPC systems, was deployed on a separate meter node to collect power/energy profiling data of the hardware components (e.g. CPUs and Memory) on this cluster. This data was recorded in a log file on the meter node and used for post-processing.

Table II shows several mainstream resilience techniques and their applicable failures. Benchmarks used in this paper are selected from NPB [32] benchmark suite, LULESH [33], AMG [34], and our fault tolerant ScaLAPACK [22]. In the next section, we will show the performance and energy results of running these applications under various resilience techniques and demonstrate if energy savings can actually be obtained using a combination of undervolting and resilience techniques on a conventional HPC system.

B. Failure Rate Calculation

Recall that the limitation of applying undervolting in HPC is that production machines used in the HPC environment do not allow further voltage reduction beyond the point of V_l, shown in Figure 2. To estimate failure rates between V_l and V_{safe_min}, we use Equation (3) to calculate the failure rates used in our experiments. As demonstrated in Figure 2, our calculated data matches well with the observed data from [9]. Thus the calculated failure rates are appropriate for empirical evaluation, although no real failures can be observed beyond V_l on our platform.

Table II
EMPIRICAL RESILIENCE TECHNIQUES AND APPLICABLE FAILURES.

Resilience Technique	Recovery Model	Failure Type
Disk-Based Checkpoint/Restart (DBCR)	Backward	Hard Errors
Diskless Checkpointing (DC)		
Triple Modular Redundancy (TMR)	Retry	Soft Errors
Algorithm-Based Fault Tolerance (ABFT)	Local/Global	

C. Undervolting Production Processors

Unlike the undervolting approach used in [9] through software/firmware control on a pre-production processor, we conduct undervolting by directly modifying the northbridge/CPU FID and VID control register [35], where FID and VID refer to frequency and voltage ID numbers respectively. The register values can be altered using the Model Specific Register (MSR) interface [36]. This process needs careful detection of the upper and lower bounds of processors' supply voltage. Otherwise overheat and hardware-damaging issues may arise.

D. Error Injection and Energy Cost Estimation

As previously discussed in Section III-A, production machines commonly disable the further voltage reduction below V_l. Thus, we need to emulate the errors that may appear for the voltage range between V_l and V_{safe_min}, based on the failure rates calculated by Equation (3). For instance, for emulating hard errors occurring at V_{safe_min}, our fault injector injects errors by arbitrarily killing parallel processes. For emulating soft errors, our injector randomly modifies (e.g. bit-flips) the values of matrix elements likewise as in [22]. Although, due to the hardware constraints, energy costs cannot be experimentally measured when undervolting to V_{safe_min}, we apply the following *emulated scaling* method to estimate the energy costs at V_{safe_min}.

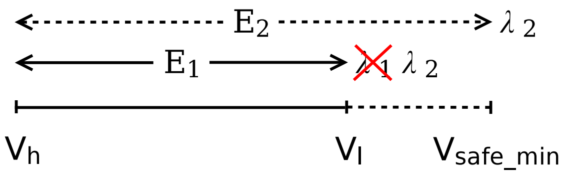


Figure 5. Estimating Energy Costs with Undervolting at V_{safe_min} for Production Processors via Emulated Scaling.

Figure 5 demonstrates this emulated scaling mechanism, where E_1 and E_2 refer to the energy costs at V_l and V_{safe_min} respectively. λ_1 and λ_2 are the calculated failure rates at the two voltage levels via Equation (3). For estimating the energy cost E_2 at V_{safe_min}, we inject the errors at the rate of λ_2 at V_l instead of using the failure rate λ_1. Moreover, in Equation (11), replacing V_{safe_min} with V_l, we can solve AC', I_{sub}, and P_c by measuring the system power consumption at V_l by applying P_h, P_m, and P_l to different phases (see Section III-C). Finally, with AC', I_{sub}, and P_c, we can calculate the values of P_h, P_m, and P_l at V_{safe_min} using Equation (11), and apply these values into Equation (12) to calculate the energy costs at V_{safe_min}.

V. Experimental Results

In this section, we present comprehensive evaluation on performance and energy efficiency of several HPC applications with undervolting and various resilience techniques. We experimentally show under what circumstances energy savings using this combinational technique can be achieved. The benchmarks under test include NPB MG (MG), NPB CG (CG), NPB FT (FT), matrix multiplication (MatMul), Cholesky factorization (Chol), LU factorization (LU), QR factorization (QR), LULESH, and AMG. All the experiments are conducted on our 8-node (64 cores) power-aware cluster shown in Table I. All the cases are under the ideal scenario that once an error occurs it will be detected.

Figure 6 shows the normalized execution time and energy costs of various benchmarks running under undervolting and four different resilience techniques. Figure 7 demonstrates the comparison of the normalized performance and energy efficiency of Adagio, a state-of-the-art DVFS technique for HPC, and our undervolting approach based on Adagio with ABFT. The test scenarios of the two figures are explained as follows: For C/R based resilience techniques (e.g. disk-based and diskless C/R), OneCkpt means checkpoint/restart is only performed once; OptCkpt@V_x refers to checkpoint/restart is performed with the optimal checkpoint interval at V_x; and OptCkpt@V_x+uv is to integrate the impacts of undervolting into OptCkpt@V_x to see time and energy changes. The nature of the Triple Modular Redundancy (TMR) and ABFT determines that the frequency of performing fault-tolerance actions is not affected by failure rates. Therefore, we simply apply undervolting to them during program execution. For the comparison against Adagio, we first run applications with Adagio, and then with Adagio and our combinational techniques together.

We use the Berkeley Lab Checkpoint/Restart (BLCR) version 0.8.5 [25] as the implementation for the Disk-Based C/R, and our own implementation of fault tolerant ScaLAPACK [22] for ABFT. We also implemented an advanced version of TMR and Diskless C/R for the selected benchmarks.

In this section, we do not include the case studies that have a mix of hard and soft errors, although it is a more realistic scenario of undervolting. The reason is that the performance and energy impacts of hard and soft error detection and recovery are accumulative, based on a straightforward hypothesis that at any point of execution only one type of failure (either a hard error or a soft error) may occur for a given process. Thus the current single-type-of-error evaluation is sufficient to reflect the trend of the impacts (see Figure 6).

A. Disk-Based Checkpoint/Restart (DBCR)

As discussed earlier, DBCR needs to save checkpoints into local disks, which can cause high I/O overhead. The first two subgraphs in Figure 6 show that for some benchmarks (MatMul, Chol, LU, QR, and AMG) single checkpoint overhead is as high as the original execution time without C/R. For matrix benchmarks, DBCR will save the large global matrix with double-precision matrix elements as well as other necessary C/R content, which explains the excessive C/R overhead. Using the scaling technique presented in Section IV-D, the failure rate at V_{safe_min} is around 10^{-1} while the one at V_l is of the order of magnitude of 10^{-3}. Based on the relationship between the checkpoint overhead and failure rates shown in Equation (6), the optimal numbers of checkpoints at these two voltages differ. Due to the high C/R overhead and a larger number of checkpoints required, undervolting to V_{safe_min} using DBCR cannot save energy at all. With less checkpoints required, undervolting to V_l still does not save energy for DBCR.

B. Diskless Checkpointing (DC)

As a backward recovery technique, compared to DBCR, DC saves checkpoints in memory at the cost of memory overhead, which is much more lightweight in terms of C/R overhead due to low I/O requirement. From the third and fourth subgraphs in Figure 6, we can observe that the C/R overhead significantly drops by an average of 44.8% for undervolting to V_l. Consequently, energy savings are obtained by an average of 8.0% and 7.0% from undervolting to V_{safe_min} and V_l respectively. Note that the energy savings from undervolting to the two voltages are similar, since the extra power saving from undervolting to a lower voltage level is offset by the higher C/R overhead (e.g. more checkpoints are required). This also indicates that the overhead from a resilience technique greatly impacts the potential energy saving from undervolting.

C. Triple Modular Redundancy (TMR)

As another effective retry-based resilience technique, Triple Modular Redundancy (TMR) is able to detect and correct error once in three runs, assuming there exists only one error arising within the three runs. Instead of using the naïve version of TMR that runs an application three times, we implemented a cost-efficient TMR that performs triple computation and then saves the results for the recovery purposes, while reducing the communication to only once to lessen overhead. This requires the assumption that we have a reliable network link for communication. Subgraphs in Figure 6 shows that we can successfully reduce TMR overhead except for LULESH, for which we had to run three times to ensure resilience. Although similar to DBCR, bearing the high overhead from the resilience technique, undervolting the system to both voltage levels by leveraging TMR generally cannot save energy. This once again demonstrates that the resilience techniques with lower overhead will benefit energy savings from undervolting.

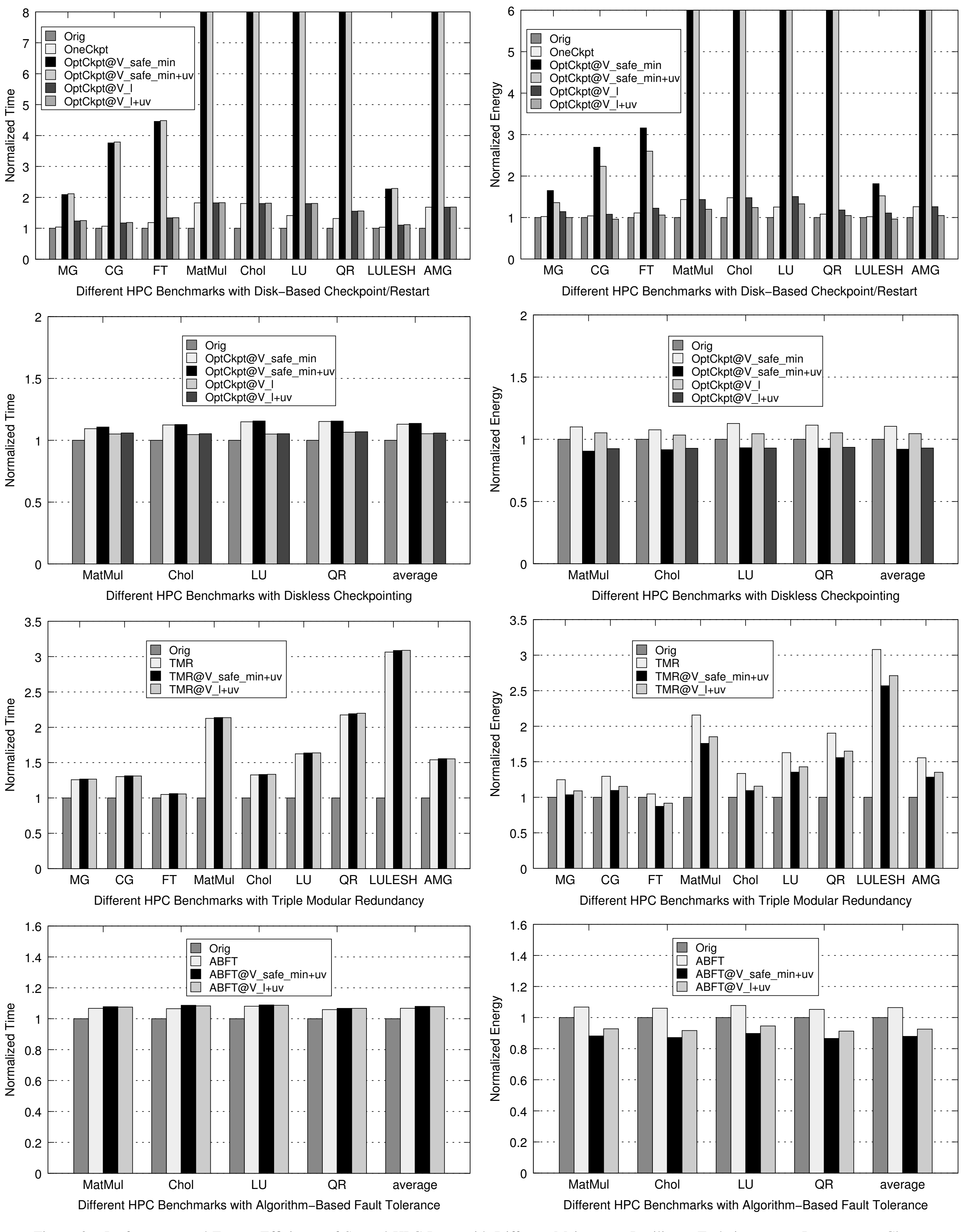
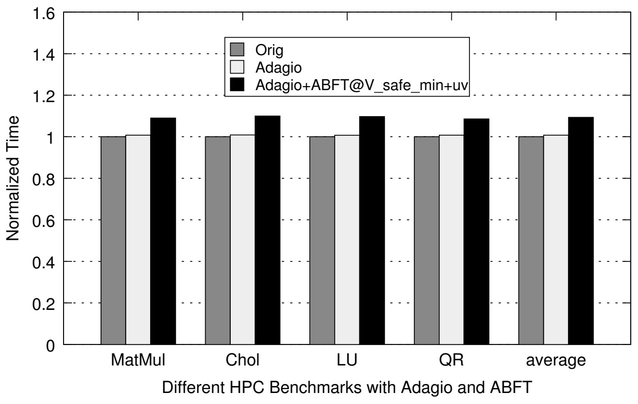


Figure 6. Performance and Energy Efficiency of Several HPC Runs with Different Mainstream Resilience Techniques on a Power-aware Cluster.



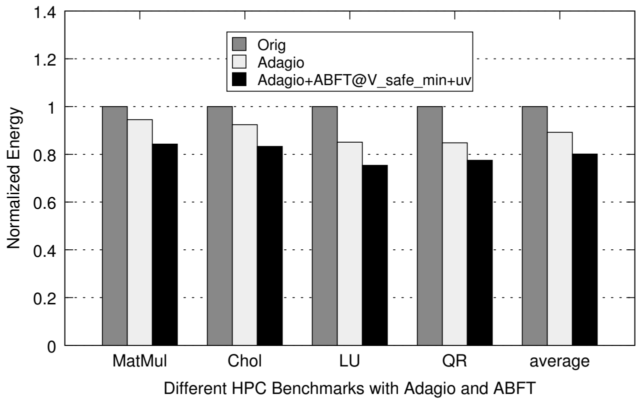


Figure 7. Performance and Energy Efficiency of the HPC Runs with an Energy Saving Solution Adagio and a Lightweight Resilience Technique ABFT.

D. Algorithm-Based Fault Tolerance (ABFT)

We also evaluate the performance counterpart of TMR that handles soft errors, based on local/global recovery using arithmetic checksums for matrix elements, i.e., ABFT. Well known for its low overhead compared to other fault tolerant techniques, ABFT is thus an ideal candidate to pair with undervolting for energy savings for some applications where ABFT can be applied. As shown in the last two subgraphs of Figure 6, without undervolting, ABFT only adds less than 6.8% overhead on average to the original runs. Similarly as in the cases of the other three resilience techniques, undervolting only incurs negligible overhead (around 1%) in ABFT. Therefore, the combined usage of ABFT and undervolting only causes minor overhead (on average 8.0% for undervolting to V_{safe_min} and 7.8% for undervolting to V_l). Another advantage of using ABFT is that ABFT is essentially based on the checksum algorithms. Checksum blocks periodically update with matrix operations and do not require to update more frequently when the failure rates increase, which means the overhead of ABFT is constant regardless of the failure rates. Consequently, the energy savings using ABFT and undervolting can be up to 12.1% compared to the original runs in our experiments. One disadvantage for using ABFT is that it is only applicable to certain applications such as matrix operations, which is not general enough to cover the entire spectrum of HPC applications.

E. Energy Savings over Adagio

As discussed in Section III-C, we aim to see if further reducing voltages for the selected frequencies of various phases by Adagio will save more energy overall. Figure 7 confirms that we are able to further save energy on top of Adagio through undervolting by effectively leveraging lightweight resilience techniques. In this experiment, we adopt the most lightweight resilience technique evaluated above, i.e., ABFT, to maximize the potential energy savings from undervolting. Undervolting was conducted on top of Adagio without modifying the runtime frequency-selecting decisions issued by Adagio, so the energy savings from Adagio are retained. Here we only present the data of undervolting to V_{safe_min} because we already know from Section V-D that undervolting to V_{safe_min} gains the most energy savings for the case of ABFT. On average, combining undervolting and Adagio can further save 9.1% more energy than just Adagio, with less than 8.5% extra performance loss. Note that the majority of the performance loss is from ABFT itself for guaranteeing resilience.

VI. Conclusions

Future large-scale HPC systems require both high energy efficiency and resilience to achieve ExaFLOPS computational power and beyond. While undervolting processors with a given frequency could decrease the power consumption of an HPC system, it often increases failure rates of the system as well, hence, increases application's execution time. Therefore, it is uncertain that applying undervolting to processors is a viable energy saving technique for production HPC systems. In this paper, we investigate the interplay between energy efficiency and resilience at scale. We build analytical models to study the impacts of undervolting on both application's execution time and energy costs. By leveraging software-level cost-efficient resilience techniques, we found that undervolting can be used as a very effective energy saving technique for the HPC field.

Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful comments and valuable suggestions. This research is partly supported by US National Science Foundation under grants #CCF-1305622 and #ACI-1305624, and DOE/ASCR Beyond Standard Model Project #62855.

References

[1] *TOP500 Supercomputer Lists*. http://www.top500.org/.

[2] *Renewable Energy and Energy Efficiency for Tribal Community and Project Development, US Department of Energy*. http://apps1.eere.energy.gov/tribalenergy/pdfs/energy04_terms.pdf.

[3] *Exascale Computing Initiative Update 2012, US Department of Energy*. http://science.energy.gov/~/media/ascr/ascac/pdf/meetings/aug12/2012-ECI-ASCAC-v4.pdf.

[4] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proc. OSDI*, 1994, p. 2.

[5] A. Miyoshi, C. Lefurgy, E. V. Hensbergen, R. Rajamony, and R. Rajkumar, "Critical power slope: Understanding the runtime effects of frequency scaling," in *Proc. ICS*, 2002, pp. 35–44.

[6] B. Rountree, D. K. Lowenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: Making DVS practical for complex HPC applications," in *Proc. ICS*, 2009, pp. 460–469.

[7] 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 *Proc. ISCA*, 2008, pp. 203–214.

[8] A. R. Alameldeen, I. Wagner, Z. Chishti, W. Wu, C. Wilkerson, and S.-L. Lu, "Energy-efficient cache design using variable-strength error-correcting codes," in *Proc. ISCA*, 2011, pp. 461–472.

[9] A. Bacha and R. Teodorescu, "Dynamic reduction of voltage margins by leveraging on-chip ECC in Itanium II processors," in *Proc. ISCA*, 2013, pp. 297–307.

[10] D. Zhu, R. Melhem, and D. Mossé, "The effects of energy management on reliability in real-time embedded systems," in *Proc. ICCAD*, 2004, pp. 35–40.

[11] D. Zhu and H. Aydin, "Energy management for real-time embedded systems with reliability requirements," in *Proc. ICCAD*, 2006, pp. 528–534.

[12] Y. Guo, D. Zhu, and H. Aydin, "Generalized standby-sparing techniques for energy-efficient fault tolerance in multiprocessor real-time systems," in *Proc. RTCSA*, 2013, pp. 62–71.

[13] P. Pop, K. H. Poulsen, V. Izosimov, and P. Eles, "Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems," in *Proc. CODES+ISSS*, 2007, pp. 233–238.

[14] 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 *Proc. MICRO*, 2003, pp. 7–18.

[15] H. Kaul, M. Anders, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar, "Near-threshold voltage (NTV) design – opportunities and challenges," in *Proc. DAC*, 2012, pp. 1153–1158.

[16] S. Li, D. H. Yoon, K. Chen, J. Zhao, J. H. Ahn, J. B. Brockman, Y. Xie, and N. P. Jouppi, "MAGE: Adaptive granularity and ECC for resilient and power efficient memory systems," in *Proc. SC*, 2012, p. 33.

[17] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving DRAM refresh-power through critical data partitioning," in *Proc. ASPLOS*, 2011, pp. 213–224.

[18] S. M. Shatz and J.-P. Wang, "Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems," *IEEE Trans. Reliability*, vol. 38, no. 1, pp. 16–27, Apr. 1989.

[19] Y. Zhang, K. Chakrabarty, and V. Swaminathan, "Energy-aware fault tolerance in fixed-priority real-time embedded systems," in *Proc. ICCAD*, 2003, pp. 209–213.

[20] *Intel® Itanium® Processor 9560 Specifications*. http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/itanium-9500-brief.pdf.

[21] J. S. Plank, K. Li, and M. A. Puening, "Diskless checkpointing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 10, pp. 972–986, Oct. 1998.

[22] P. Wu and Z. Chen, "FT-ScaLAPACK: Correcting soft errors on-line for ScaLAPACK Cholesky, QR, and LU factorization routines," in *Proc. HPDC*, 2014, pp. 49–60.

[23] A. Moody, G. Bronevetsky, K. Mohror, and B. R. de Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *Proc. SC*, 2010, pp. 1–11.

[24] P. Du, A. Bouteiller, G. Bosilca, T. Herault, and J. Dongarra, "Algorithm-based fault tolerance for dense matrix factorizations," in *Proc. PPoPP*, 2012, pp. 225–234.

[25] J. Duell, "The design and implementation of Berkeley lab's Linux Checkpoint/Restart," Lawrence Berkeley National Laboratory, Tech. Rep., 2003.

[26] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Communications of the ACM*, vol. 17, no. 9, pp. 530–531, Sep. 1974.

[27] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Generation Computer Systems*, vol. 22, no. 3, pp. 303–312, Feb. 2006.

[28] L. Tan and Z. Chen, "Slow down or halt: Saving the optimal energy for scalable HPC systems," in *Proc. ICPE*, 2015, pp. 241–244.

[29] S. Song, C.-Y. Su, R. Ge, A. Vishnu, and K. W. Cameron, "Iso-Energy-Efficiency: An approach to power-constrained parallel computation," in *Proc. IPDPS*, 2011, pp. 128–139.

[30] E. Meneses, O. Sarood, and L. V. Kalé, "Assessing energy efficiency of fault eolerance protocols for HPC systems," in *Proc. SBACPAD*, 2012, pp. 35–42.

[31] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron, "PowerPack: Energy profiling and analysis of high-performance systems and applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 5, pp. 658–671, May 2010.

[32] *NAS Parallel Benchmarks (NPB)*. http://www.nas.nasa.gov/publications/npb.html.

[33] *Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH)*. https://codesign.llnl.gov/lulesh.php.

[34] *A Parallel Algebraic Multigrid (AMG) Solver for Linear Systems*. https://www.nersc.gov/users/computational-systems/cori/nersc-8-procurement/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks/amg/.

[35] *BIOS and Kernel Developers Guide (BKDG) For AMD Family 10h Processors*. http://developer.amd.com/wordpress/media/2012/10/31116.pdf.

[36] *Model-Specific Register (MSR) Tools Project*. https://01.org/msr-tools.