A

Scalable Energy Efficiency with Resilience for High Performance Computing Systems: A Quantitative Methodology

LI TAN, University of California, Riverside
ZIZHONG CHEN, University of California, Riverside
SHUAIWEN LEON SONG, Pacific Northwest National Laboratory

Ever-growing performance of supercomputers nowadays brings demanding requirements of energy efficiency and resilience, due to rapidly expanding size and duration in use of the large-scale computing systems. Many application/architecture-dependent parameters that determine energy efficiency and resilience individually have causal effects with each other, which directly affect the trade-offs among performance, energy efficiency and resilience at scale. To enable high-efficiency management for large-scale High Performance Computing (HPC) systems nowadays, quantitatively understanding the entangled effects among performance, energy efficiency, and resilience is thus required. While previous work focuses on exploring energy saving and resilience enhancing opportunities separately, little has been done to theoretically and empirically investigate the interplay between energy efficiency and resilience at scale. In this paper, by extending the Amdahl's Law and the Karp-Flatt Metric, taking resilience into consideration, we quantitatively model the integrated energy efficiency in terms of performance per Watt, and showcase the trade-offs among typical HPC parameters, such as number of cores, frequency/voltage, and failure rates. Experimental results for a wide spectrum of HPC benchmarks on two HPC systems show that the proposed models are accurate in extrapolating resilience-aware performance and energy efficiency, and capable of capturing the interplay among various energy saving and resilience factors. Moreover, the models can help find the optimal HPC configuration for the highest integrated energy efficiency, in the presence of failures and applied resilience techniques.

CCS Concepts:•**Computer systems organization** → **Distributed architectures;**•**Hardware** → *Power estimation and optimization;* Fault tolerance;

Additional Key Words and Phrases: energy, power, DVFS, undervolting, performance, resilience, failures, checkpoint and restart, scalability, HPC

1. INTRODUCTION

As the exascale supercomputers are expected to embark around 2020 [Esmaeilzadeh et al. 2011], High Performance Computing (HPC) systems nowadays expand rapidly in size and duration in use, which brings demanding requirements of energy efficiency and resilience at scale, along with the ever-growing performance boost. These requirements are becoming prevalent and challenging, considering two crucial facts that: (a) The costs of powering an HPC system grow greatly with its expanding scale, and (b) the failure rates of an HPC system are dramatically increased due to a larger amount of interconnected computing nodes. Therefore, it is desirable to consider both dimensions of energy efficiency and resilience, when building scalable, cost-efficient, and robust

New Paper, Not an Extension of a Conference Paper. This work is partially supported by the NSF grants CCF-1305622, ACI-1305624, and CCF-1513201, by the DOE/ASCR Beyond Standard Model Project 62855 from Pacific Northwest National Laboratory, and also by the SZSTI basic research program JCYJ20150630114942313.
Author's addresses: L. Tan and Z. Chen, Department of Computer Science and Engineering, University of California, 900 University Avenue, Riverside, California 92521; S. L. Song, High Performance Computing Group, Pacific Northwest National Laboratory, 902 Battelle Boulevard, Richland, Washington 99354.

DOI: 0000001.0000001

large-scale HPC systems. Specifically, for a given HPC system, our ultimate goal is to achieve the optimal performance-power-failure ratio while exploiting parallelism.

Nevertheless, for the concerns of energy efficiency and resilience in scalable HPC systems, alleviating one dimension does not necessarily improve the other. Energy efficiency and resilience are essentially mutually-constrained during the efforts of finding the balanced HPC configuration for the integrated optimal performance-power-failure ratio. Despite the straightforward fact that both of energy efficiency and resilience are correlated with execution time of HPC runs, altering some HPC parameters that are closely related to both dimensions, such as supply voltage of hardware components and number of cores used, can be beneficial to one dimension but harmful to the other.

For instance, energy savings can be achieved via Dynamic Frequency and Voltage Scaling (DVFS) techniques [Weiser et al. 1994] [Rountree et al. 2009] [Tan et al. 2014] [Tan and Chen 2015], for CMOS-based processing components including CPU, GPU, and memory. In general practice, DVFS is often frequency-oriented towards idle time of the components, which means the voltage will be changed if the paired frequency is altered but will be kept the same otherwise. Nowadays state-of-the-art processors with cutting-edge nano-technology are allowed to be supplied with a significantly low voltage, close to the transistor's threshold voltage, e.g., Intel's Near-Threshold Voltage (NTV) design [Kaul et al. 2012]. Further energy savings can be achieved through a fixed-frequency scheme with further reduced voltage, named *undervolting* [Wilkerson et al. 2008] [Alameldeen et al. 2011] [Bacha and Teodorescu 2013] [Tan et al. 2015], at the cost of increased failures of the components. However, it is not clear that which variation from undervolting is more dominant for high energy efficiency: power savings from further voltage reduction or performance loss from the overhead on error detection and recovery. It is thus desirable to investigate the potential of achieving high energy efficiency in HPC by undervolting, with hardware/software-level resilience techniques applied meanwhile to guarantee the correct execution of HPC runs.

There exist only a few efforts investigating this issue in the context of HPC with well-grounded modeling and experimental validation [Bacha and Teodorescu 2013] [Tan et al. 2015]. However, the proposed approach in [Bacha and Teodorescu 2013] worked specifically for a customized pre-production multi-core processor with ECC (Error-Correcting Code) memory, and thus their solution considered a single type of potential failures, i.e., ECC errors only. On the other hand, the authors in [Tan et al. 2015] did not theoretically consider the effects of scalability of HPC systems and discuss the interplay among mutually-constrained HPC parameters at scale, nor empirically evaluated the trade-offs among the HPC parameters towards scalable energy efficiency and resilience. Therefore, in this work, we propose to quantitatively model the entangled effects of energy efficiency and resilience in the scalable HPC environment and investigate the trade-offs among typical HPC parameters for the optimal energy efficiency with resilience. In summary, the contributions of this work include:

— We quantitatively model the entangled effects of energy efficiency and resilience at scale, by extending the Amdahl's Law and the Karp-Flatt Metric;
— We showcase the interplay among typical HPC parameters with internal causal effects, and demonstrate how the trade-offs impact the energy efficiency at scale;
— We provide experimental results using ten HPC benchmarks on two power-aware clusters, showing that our models are accurate and effective to find the balanced HPC configuration for the optimal scalable energy efficiency under resilience constraints.

The remainder of this paper is organized as follows. Section 2 discusses related work and Section 3 introduces background. We present our modeling of scalable energy efficiency with resilience, and trade-offs among HPC parameters in Section 4. Implementation details and experimental results are provided in Section 5. Section 6 concludes.

2. RELATED WORK

There exist few efforts investigating the joint relationship among performance, energy efficiency, and resilience for HPC systems. Rafiev *et al.* [Rafiev et al. 2014] studied the interplay among time, energy costs, and reliability for a single-core and a multicore system respectively, while they focused on concurrency and did not quantitatively elaborate the impacts of frequency/voltage on performance and reliability. Yetim *et al.* [Yetim et al. 2012] presented an energy optimization framework using mixed-integer linear programming while meeting performance and reliability constraints. Targeting the application domain of multimedia, this work exploited the workload characteristics that limited error tolerance can be traded off for energy reduction. Nevertheless, there exist a large body of studies that quantify only energy costs and performance, or resilience and performance, at single-node level, at scale, or based on simulation.

ENERGY EFFICIENCY QUANTIFICATION. Woo and Lee [Woo and Lee 2008] built an analytical model that extends the Amdahl's Law for energy efficiency in scalable many-core processor design. They considered three many-core design styles of processors only without communication, while we focus on networked symmetric multicore processors, and communication time and power costs for processors across nodes are involved in our models. By augmenting the Amdahl's Law, Cassidy and Andreou [Cassidy and Andreou 2012] derived a general objective function linking performance gain with energy-delay costs in microarchitecture and applied it to design the optimal chip multiprocessor (CMP) architecture, while our work aims to achieve the optimal performance/energy efficiency in terms of FLOPS per watt. Song *et al.* [Song et al. 2011] developed an energy model to evaluate and predict energy-performance trade-offs for various HPC applications at large scale. Ge and Cameron [Ge and Cameron 2007] devised a power-aware speedup metric that quantify the interacting effects between parallelism and frequency, and used it to predict performance and power-aware speedup for scientific applications. Their models were accurate and scalable, but did not incorporate the effects of energy saving DVFS and undervolting techniques as in our work. Moreover, they did not evaluate the interplay among typical HPC parameters, where we conduct an extensive theoretical and empirical study in this work.

RESILIENCE QUANTIFICATION. Zheng and Lan [Zheng and Lan 2009] modeled the impacts of failures and the effects of resilience techniques in HPC, and used their models to predict scalability for HPC runs with possibility of failures. Wang *et al.* [Wang 2009] proposed a unified speedup metric that incorporates checkpointing overhead into classic speedup metrics in the presence of failures. Yu *et al.* [Yu et al. 2014] created a novel resilience metric named data vulnerability factor to holistically integrate application and hardware knowledge into resilience analysis. Again, all of these approaches however did not consider energy saving DVFS and undervolting techniques, both of which can significantly affect energy efficiency and resilience, with negligible performance loss. Moreover, analytical models in [Wang 2009] are not as fine-grained as ours, since relationship among important HPC parameters was simplified such that it is not clear how they interact and by what extent they affect speedup (and energy efficiency).

SIMULATION-BASED QUANTIFICATION. Li and Martínez [Li and Martínez 2005] investigated power-performance correlation of parallel applications running on a CMP, aiming to optimize power costs under a performance budget and vice versa. They conducted simulation-based experiments to evaluate the proposed power-performance optimization for one single parallel application. Suleman *et al.* [Suleman et al. 2009] proposed a technique that accelerates the execution of critical sections, a code fragment of shared data accessed by only one thread at a given time, which differs from the sequential code in the Amdahl's Law. Experimental results using lock-based multithreaded workloads on three different CMP architectures indicated significant performance and scalability improvement. Bois *et al.* [Bois et al. 2011] developed a framework of generating synthetic workloads to evaluate energy efficiency for multicore power-aware systems. The proposed framework effectively showed the energy and performance trade-off for generated workloads. All these simulation approaches are at single-node level, which may need considerate adaptation to work for large-scale HPC systems.

3. BACKGROUND: ENERGY SAVINGS, UNDERVOLTING, AND FAILURES

Numerous efforts have been made to address the demanding requirements of energy efficiency in HPC nowadays. In general, processor-based energy saving techniques can be categorized into two types: *frequency-directed* and *voltage-directed*. Next we present the details of each as the background knowledge for later modeling and discussion.

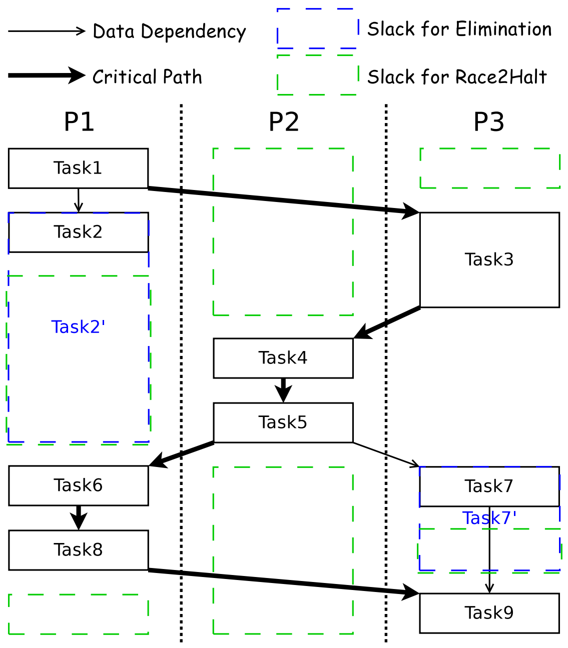

Fig. 1. DAG Notation of Two DVFS Solutions for a 3-Process HPC Run.

3.1. Frequency-Directed DVFS Techniques

Generally for an HPC run, *slack* refers to a time period when one hardware component waits for another due to imbalanced throughput and utilization [Tan and Chen 2015]. There exist many slack opportunities during an HPC run. For instance, if network components are busy, other components (e.g., CPU, GPU, memory, and disk) are often alternatingly idle when message-passing communication (specifically, message copy among buffers of different compute nodes) is performed. Moreover, if the application is memory and disk intensive, CPU usually waits for the data from memory and disk, due to the performance bottleneck at memory and disk accesses. An ideal method to save energy is to reduce the power of non-busy components when such slack occurs, while keep the peak performance of busy components when otherwise. By exploiting DVFS, existing solutions decrease processor power during communication [Rountree et al. 2007] [Tan et al. 2014], halt/idle processors when no workloads are available [Alonso et al. 2012] [Tan and Chen 2015], or work based on *Critical Path* (CP) analysis to reclaim slack arising among computation [Rountree et al. 2009] [Tan and Chen 2015]. Energy can be saved with negligible performance loss using the above solutions.

For energy saving purposes, without degrading performance, slack arising in HPC runs can either be eliminated by decreasing processor frequency to extend computation time of program execution fragments (e.g., tasks) with slack appropriately, or be utilized to make non-working hardware components stay in the halt/idle state, as shown in Figure 1 [Tan and Chen 2015], which depicts the Directed Acyclic Graph (DAG) representation of applying two classic DVFS approaches on a task-parallel HPC run. By respecting the CP, slack is not over-reclaimed or over-utilized for halting, and thus the DVFS solutions incur negligible performance loss from DVFS itself in practice. In this work, we adopt state-of-the-art DVFS approaches that outperform other DVFS solutions in different scenarios by completely eliminating potential slack. Similarity among these DVFS solutions is that they are all *frequency-directed*: Processor voltage is only lowered together with processor frequency reduction, in the presence of slack, and is fixed otherwise. In other words, a voltage is always paired with a chosen frequency, and no further voltage reduction is conducted for the given frequency.

3.2. Fixed-Frequency Undervolting Technique

Although effective, frequency-directed DVFS approaches may fail to fully exploit energy saving opportunities. Hardware components such as processors nowadays are allowed to be supplied with a voltage that is lower than the one paired with a given frequency, which is referred to as *undervolting* for more power savings beyond DVFS. This *voltage-directed* technique is independent of frequency scaling (i.e., during undervolting, the frequency is fixed after it is chosen), but requires hardware support for empirical deployment. Unlike traditional simulation-based undervolting approaches [Wilkerson et al. 2008] [Alameldeen et al. 2011], Bacha *et al.* [Bacha and Teodorescu 2013] first implemented an empirical undervolting system on Intel Itanium II processors via software/firmware control, which was intended to reduce voltage margins and thus save power, with ECC memory correcting arising ECC errors. This work maximized potential power savings since it used pre-production processors that allows the maximum extent of undervolting: They were able to reduce voltage until the levels lower than the lowest voltage corresponding to the lowest frequency supported. In general, production processors are locked for reliability purposes by the OS, and will typically shut down when voltage is lowered below the one paired with the lowest frequency. For *generality* purposes, Tan *et al.* [Tan et al. 2015] proposed an *emulated scaling* undervolting scheme that works for general production processors, which was deployed on a power-aware HPC cluster as the first attempt of its kind to demonstrate more energy savings compared to state-of-the-art DVFS solutions. They implemented undervolting using the Model Specific Register interface, which does not require the support of pre-production machines and makes no modification to the hardware.

The further power savings from undervolting are however achieved at the cost of higher failure rates λ [Zhu et al. 2004] [Bacha and Teodorescu 2013]. As shown in Equation (1) below, the average failure rates are quantified in terms of supply voltage only with other parameters known [Tan et al. 2015] (λ_0: the average failure rate at f_{max} (and V_{max}), d and β: hardware-dependent constants, f_{max}/f_{min}: the highest/lowest operating frequency, V_{th}: threshold voltage). Therefore while undervolting is beneficial to saving power, hardware/software-based fault tolerant techniques are also required to guarantee correct program execution. Bacha *et al.* [Bacha and Teodorescu 2013] employed ECC memory to correct memory bit failures (single-bit flips). Tan *et al.* [Tan et al. 2015] adopted lightweight resilience techniques such as diskless checkpointing and algorithm-based fault tolerance to correct both hard and soft errors. The difference lies in: Pre-production machines are required in the work of Bacha *et al.*, where real errors can be observed at the lowest safe voltage V_{safe_min}, $V_{th} < V_{safe\_min} < V_l$, while the solution proposed by Tan *et al.* needs production machines only, and thus errors from undervolting cannot be empirically observed, due to close-to-zero failure rates at V_l per Equation (1), i.e., the lowest voltage can be scaled to by undervolting for production machines without crashing. The emulated scaling scheme [Tan et al. 2015] was able to estimate the energy costs at V_{safe_min}, based on real measured power/energy data at V_l and power/energy models under resilience techniques and undervolting.

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

Since this work is intended for general production HPC systems, we leverage the software-based undervolting approach [Tan et al. 2015] to save more energy beyond DVFS solutions at the cost of raised failure rates. Thus as in [Tan et al. 2015], no errors were experimentally observed, but were likewise successfully emulated (see section 5.2). In this work, the Checkpoint/Restart technique is employed to recover from errors.

3.3. Checkpoint/Restart Failure Model

Computing systems in general suffer from various sources of failures, ranging from computation errors on logic circuits, to memory bit-flips due to frequency and voltage fluctuation [Bacha and Teodorescu 2013] [Wu and Chen 2014] [Tan et al. 2015]. Without loss of generality, in this work we discuss how to detect and recover from a failure in an HPC run using a general-purpose widely used resilience technique Checkpoint/Restart (C/R) [Daly 2006], and build and evaluate our performance and power models based on C/R. Our methodology of theoretical modeling and experimental evaluation also applies to other resilience techniques such as Algorithm-Based Fault Tolerance (ABFT), with minor changes in the proposed models accordingly. Note that we use the terms *errors*, *faults*, and *failures* interchangeably henceforth in the later text.

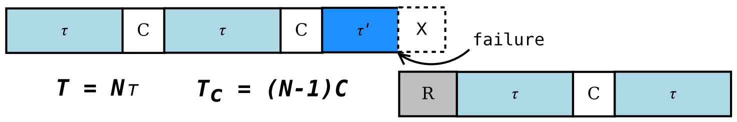

Fig. 2. Fault Tolerance using the Checkpoint/Restart Technique.

Figure 2 demonstrates the scenario of fault tolerance using C/R. A program with the execution time T can be *checkpointed* by making a snapshot of a system state at the end of fragments of an evenly divided program run so that $T = N\tau$, where τ is the length of fragments of the divided program run, and $N - 1$ is the number of checkpoints added into the run. Specifically, a system state is a copy of current application process address space, including the contents of values of heap, stack, global variables, program text and data, and registers. The total checkpoint overhead is thus modeled as $T_C = (N - 1)C$, where C is the time required for making one checkpoint. At any points of time within a program run fragment, a failure can arise and interrupt the program run. We denote the time that a failure occurs as τ'. For continuing the run without re-executing the whole program, we reinstate the last saved checkpoint and restart from the saved information in the checkpoint. The time overhead on restarting the run is denoted as R. The total restart overhead depends on the number of failures during the run.

4. MODELING SCALABLE ENERGY EFFICIENCY WITH RESILIENCE

4.1. Problem Description

We aim to achieve the optimal energy efficiency with resilience in a scalable HPC environment as in Figure 3: an HPC system with a number of compute nodes, each of which consists of multiple symmetric cores, interconnected by networks. Note that we

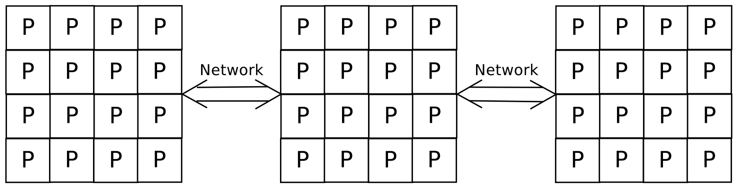

Fig. 3. Investigated Architecture – Symmetric Multicore Processors Interconnected by Networks.

assume if there exist multiple multicore processors in a node, the cores across processors are also symmetric. Although at the initial stage, applicable architectures are homogeneous HPC systems without accelerators, with minor changes, the methodology proposed and the empirical studies conducted in this work also apply to other architectures, such as emerging heterogeneous GPU/coprocessor-accelerated HPC systems.

4.2. Amdahl's Law and Karp-Flatt Metric

Amdahl's Law [Amdahl 1967] and Karp-Flatt Metric [Karp and Flatt 1990] are two classic metrics that quantify the performance of parallel programs. Amdahl's Law stresses performance impacts from the parallelized code within a parallel program, without considering communication. Karp-Flatt speedup formula takes communication (including data transmission and synchronization in HPC runs) into account, likewise as the consideration of this work. Specifically, Amdahl's Law is written as:

$$\texttt{Speedup}_{\texttt{a}} = \frac{T_s + T_p}{T_s + \frac{T_p}{P}} = \frac{(1-\alpha)T + \alpha T}{(1-\alpha)T + \frac{\alpha T}{P}} = \frac{1}{1 - \alpha + \frac{\alpha}{P}} \tag{2}$$

where $T = T_s + T_p$ is the total execution time of the program, consisting of the runtime of the sequential code T_s and the runtime of the parallelized code T_p individually, α is the percentage of code that can be parallelized within the program ($0 \leq \alpha \leq 1$), P is the total number of cores used in the HPC system where the program runs, and N is the problem size. Considering $\kappa(N, P)$ as the communication time, determined by N and P jointly, Karp-Flatt speedup formula can be written in the following form:

$$\begin{aligned}\texttt{Speedup}_{\texttt{kf}} &= \frac{T_s + T_p}{T_s + \frac{T_p}{P} + T_{comm}} = \frac{(1-\alpha)T + \alpha T}{(1-\alpha)T + \frac{\alpha T}{P} + \kappa(N,P)} \\ &= \frac{1}{1 - \alpha + \frac{\alpha}{P} + \frac{\kappa(N,P)}{T}}\end{aligned} \tag{3}$$

4.3. Extended Amdahl's Law for Power Efficiency

The original Amdahl's Law considers performance of HPC systems only. Woo *et al.* [Woo and Lee 2008] incorporated power and energy efficiency into the Amdahl's Law, without considering communication as in the Karp-Flatt speedup formula. In this work, we take into account both the power/energy efficiency and the communication during HPC runs by extending the classic Amdahl's Law for scalable HPC systems.

4.3.1. A General Extension. We first formulate the total power consumption of all hardware components in an HPC system (specifically, including multicore processors across nodes with P cores in total and other components) during an original error-free run, without considering energy saving DVFS and undervolting solutions applied:

$$\begin{aligned}
\texttt{Power} &= \frac{Energy}{Time} \\
&= \frac{(Q+(P-1)\mu Q)(1-\alpha)T + PQ\frac{\alpha T}{P} + P\mu Q\kappa(N,P) + \mathbb{C}((1-\alpha)T + \frac{\alpha T}{P} + \kappa(N,P))}{(1-\alpha)T + \frac{\alpha T}{P} + \kappa(N,P)} \\
&= Q \times \frac{(1+\mu(P-1))(1-\alpha) + \alpha + \mu P\frac{\kappa(N,P)}{T} + \frac{\mathbb{C}((1-\alpha)+\frac{\alpha}{P}+\frac{\kappa(N,P)}{T})}{Q}}{(1-\alpha)+\frac{\alpha}{P}+\frac{\kappa(N,P)}{T}} \qquad (4)
\end{aligned}$$

where Q is the power consumption of a single core at the peak performance, and μ is the fraction of the power the core consumes in the idle state with regard to that at the peak performance ($0 < \mu < 1$). We assume that the core power consumption during communication is roughly the same as that in its idle state [Ge et al. 2010]. We calculate the total energy consumption by accumulating energy costs at different phases of an HPC run: $(Q + (P-1)\mu Q)(1-\alpha)T$ is the energy costs of all P cores when the sequential code is executed (one core runs at full speed while the others are idle). $PQ\frac{\alpha T}{P}$ is the P-core energy costs when the program runs in parallel. $P\mu Q\kappa(N,P)$ refers to the energy costs the P cores produce during communication. $\mathbb{C}$ is the power costs of non-CPU components in the HPC system (we assume non-CPU components consume constant power during HPC runs, regardless of DVFS and undervolting for CPU only). As an extension to the Amdahl's Law that quantified speedup only of HPC runs, Equation (4) gives the total system power costs of HPC runs in general cases.

Without loss of generality, in Equation (4), we normalize $Q = 1$ to simplify the later discussion. Moreover, we set $\mathbb{C} = 0$ in the following modeling to focus on CPU power/energy efficiency due to two primary reasons: (a) CPU is the most power/energy consumer in a homogeneous HPC system [Ge et al. 2010] [Tan et al. 2014], i.e., saving energy for CPU has the most significant impacts on improving the system energy efficiency, and (b) we investigate the DVFS and undervolting solutions that directly affect CPU power/energy costs but not impact other non-CPU components. More hardware components such as GPU/memory/networks can also be incorporated if power/energy efficient techniques on GPU/memory/networks are considered. Due to space limitation, we elaborate our idea in this work taking CPU for example, and leave studying power/energy efficiency of other components in the HPC system as future work.

4.3.2. Communication of HPC Runs. As stated, we denote the communication time in an HPC run as $\kappa(N,P)$. Empirically, $\kappa(N,P)$ highly depends on the communication algorithm employed. Regardless of the basic point-to-point communication scheme, there exist a large body of studies on highly-tuned communication algorithms [Chan et al. 2006] [Solomonik et al. 2011] [Ballard et al. 2012] [Lee et al. 2013]. Here we briefly discuss two classic broadcast algorithms: binomial tree and pipeline broadcast. Their performance comparison was summarized in [Culler et al. 1993] [Tan et al. 2013], where the time complexity of binomial tree broadcast was modeled as $T_B = \frac{S_{msg}}{BD} \times logP$, and the pipeline broadcast time complexity was modeled as $T_P = \frac{S_{msg}}{BD} \times (1 + \frac{P-1}{\eta(N)})$ (S_{msg} is the message size in one broadcast, BD refers to network bandwidth in the communication, and $\eta(N)$ is the number of message chucks in the message transmission). We can see that performance of binomial tree broadcast depends on P only while performance of pipeline broadcast is determined by both P and N. In practice, the communication scheme and algorithm vary from different HPC applications, which determine what time complexity of $\kappa(N,P)$ is and ultimately affect the power/energy costs of HPC runs.

4.3.3. Power Efficiency with DVFS and Undervolting. Given the fact that the total power consumption of a core is composed of leakage/static power costs and dynamic power costs, the Amdahl's Law can be intuitively rewritten to model the potential core power savings for HPC runs, under the circumstances of DVFS and undervolting respectively:

$$\texttt{PE}_{\texttt{dvfs}} = \frac{P_s + P_d}{\frac{(1-\beta)(P_s+P_d)}{n_1} + \frac{\beta(P_s+P_d)}{n_2}} = \frac{1}{\frac{1-\beta}{n_1} + \frac{\beta}{n_2}} \tag{5}$$

$$\texttt{PE}_{\texttt{uv}} = \frac{P_s + P_d}{\frac{(1-\beta)(P_s+P_d)}{n_1} + \frac{\beta(P_s+P_d)}{n_3}} = \frac{1}{\frac{1-\beta}{n_1} + \frac{\beta}{n_3}} \tag{6}$$

where P_s and P_d refer to the leakage and dynamic power costs of all used cores individually, and β is the percentage of dynamic power costs within the total power costs. n_1, n_2, and n_3 are the leakage power reduction factor of DVFS/undervolting, the dynamic power reduction factor of DVFS, and the dynamic power reduction factor of undervolting, individually, denoting the ratios of leakage/dynamic power reduction respectively. Equation (5) and Equation (6) model the power efficiency when DVFS and undervolting are applied separately. For simplicity of the discussion, we assume that the DVFS technique used is able to eliminate all slack in the HPC run, and the undervolting technique adopted is able to scale to the voltage paired with the lowest frequency.

EXAMPLE. We can use the above two formulae to calculate the theoretical upper bound of power savings using DVFS and undervolting respectively. Assume we have the following parameters already known: $\beta = 0.6$ (dynamic power amounts to 60% of the total core power, which is empirically reasonable). For the processors used, we assume the maximum frequency $f_h = 2.4$ GHz and the minimum frequency $f_l = 0.8$ GHz. Consider the extreme case that reducing frequency from 2.4 GHz to 0.8 GHz can eliminate all possible slack for all tasks. Since $P_s = I_{sub}V$ [Taur et al. 2004] (i.e., $\Delta P_s \propto \Delta V$), $P_d = AC'fV^2$ [Miyoshi et al. 2002], and empirically $\Delta P_d \propto \Delta f^{2.5}$ [Rotem et al. 2014], where A and C' are the percentage of active gates and the total capacitive load in a CMOS-based processor respectively, and I_{sub} refers to subthreshold leakage current, we can derive that $\Delta V \propto \Delta f^{0.75}$, and thus $n_1 = (\frac{f_h}{f_l})^{0.75}_{dvfs} = (\frac{2.4}{0.8})^{0.75} \approx 2.28$ and $n_2 = (\frac{f_h}{f_l})^{2.5}_{dvfs} = (\frac{2.4}{0.8})^{2.5} \approx 15.59$ in the assumed case. By substituting n_1 and n_2 into Equation (5), we have the range of power savings from DVFS: $\texttt{PE}_{\texttt{dvfs}} \le \frac{1}{\frac{0.4}{2.28} + \frac{0.6}{15.59}} \approx 4.67$. Moreover, consider in the case of undervolting, we have the same $n_1 = (\frac{f_h}{f_l})^{0.75}_{uv} = (\frac{2.4}{0.8})^{0.75} \approx 2.28$ and $n_3 = (\frac{f_h}{f_l})^{1.5}_{uv} = (\frac{2.4}{0.8})^{1.5} \approx 5.26$. Substituting all known parameters we have the range of power savings from undervolting: $\texttt{PE}_{\texttt{uv}} \le \frac{1}{\frac{0.4}{2.28} + \frac{0.6}{5.26}} \approx 3.45$.

4.4. Extended Karp-Flatt Metric for Speedup with Resilience

From the Karp-Flatt speedup formula, i.e., Equation (3), we can further derive the extended Karp-Flatt speedup formula when failures occur and resilience techniques are employed in HPC runs. Although we take the C/R technique for example in the later discussion, our modeling also applies to other resilience techniques by making changes on the modeling of error detection and recovery. Next we formulate the Karp-Flatt Metric for the scenario with failures and C/R, by adopting Daly's simplified C/R performance model $T_{cr} = \frac{1}{\lambda}e^{R\lambda}(e^{\lambda(\tau+C)} - 1)\frac{T}{\tau}$ [Daly 2006]. We substitute the original solve time of the parallelized code with the solve time with failures and C/R T_{cr}:

$$\begin{aligned}\texttt{Speedup}^{\texttt{cr}}_{\texttt{kf}} = \frac{T_{orig}}{T_{cr}} &= \frac{(1-\alpha)T + \alpha T}{\frac{1}{\lambda}e^{R\lambda}(e^{\lambda(\tau+C)} - 1)\frac{(1-\alpha)T + \frac{\alpha T}{P} + \kappa(N,P)}{\tau}} \\ &= \frac{1}{\frac{1}{\lambda}e^{R\lambda}(e^{\lambda(\tau+C)} - 1)\frac{1-\alpha+\frac{\alpha}{P}+\frac{\kappa(N,P)}{T}}{\tau}}\end{aligned} \tag{7}$$

4.5. Quantifying Integrated Energy Efficiency

Based on the two extended models, further taking HPC communication, energy saving DVFS and undervolting solutions, and failures and resilience techniques into consideration, we define the integrated energy efficiency of an HPC system as follows, in terms of the speedup in different HPC scenarios achieved per unit energy per unit time:

$$\frac{\mathsf{Perf}}{\mathsf{Watt}} = \frac{\texttt{Speedup}}{\texttt{Power}} \tag{8}$$

We next consider the following four typical HPC scenarios individually, where different integrated energy efficiency metrics can be produced according to Equation (8).

COMMUNICATION-TIME-TO-TOTAL-TIME-RATIO. The later discussion depends on a practical assumption that our models work well for HPC applications with the following characteristics: The ratio of communication time to the total execution time $\frac{\kappa(N,P)}{T}$ (this term appears in Equations (9-15)) does not vary much as problem size N and number of cores P change. There exist a large body of such applications [Jin and der Wijngaart 2004] [Wu et al. 2012], including a majority of benchmarks evaluated in Section 5. Nevertheless, for applications with a great variation of $\frac{\kappa(N,P)}{T}$ (e.g., the IS benchmark from NPB [npb]), the accuracy of our proposed models may be affected but the trend of energy efficiency manifested by our models may still retain.

We first consider the baseline case that there are no failures during an HPC run, without DVFS and undervolting techniques. Based on the definition, we calculate the energy efficiency in this case, as shown in Equation (9), where we assume that the HPC run uses P cores in total that solves a size-N problem. A number of parameters are influential to the energy efficiency, including several fixed application-specific and architecture-specific invariants: the percentage of parallelized code α, and the ratio μ, power consumed by one idle core normalized to that by one fully-loaded running core. Equation (9) models the baseline for building energy efficiency models of the following HPC scenarios, with energy saving solutions, failures, and resilience techniques.

$$\begin{aligned}\frac{\mathsf{Perf}}{\mathsf{Watt}} &= \frac{1}{1-\alpha+\frac{\alpha}{P}+\frac{\kappa(N,P)}{T}} \times \frac{1-\alpha+\frac{\alpha}{P}+\frac{\kappa(N,P)}{T}}{(1+\mu(P-1))(1-\alpha)+\alpha+\mu P\frac{\kappa(N,P)}{T}} \\ &= \frac{1}{(1+\mu(P-1))(1-\alpha)+\alpha+\mu P\frac{\kappa(N,P)}{T}}\end{aligned} \tag{9}$$

<u>**Scenario 1**</u>: HPC Runs with Faults and C/R (No DVFS + No Undervolting)

If faults occur at a rate of λ and the C/R technique is employed in the HPC run, resilience-related factors including failure rates λ, checkpoint intervals τ, checkpoint overhead C, and restart overhead R are involved in the speedup formula, which affect the overall energy efficiency as well. Using the performance model T_{cr} for the scenario with failures given in Equation (7), we first model the power costs P_{cr} in this scenario as Equation (10), based on the assumption that the power draw of a node during checkpointing and restarting is very close to that during its idle state [Meneses et al. 2012]. In Equation (10), $\mathbb{N}$ is the expected number of failures, approximated in [Daly 2006] as $\mathbb{N} = \lambda T(1+\frac{C}{\tau})$. If the solve time T is comparatively long, we can assume $\mathbb{N} \gg 1$, and thus further simplify the power formula. The ultimate energy efficiency is given in Equation (11). In contrast to the energy efficiency in the baseline case, there appears a new term $\mu P\lambda(1+\frac{C}{\tau})(C+R)$ in the new energy model. Intuitively, since the speedup is degraded due to the checkpoint and restart overhead, and the overall energy costs are raised due to the extra energy costs during checkpointing and restarting, the energy efficiency in this scenario is expected to go down, compared to the baseline case.

$$P_{cr} = \frac{E_{cr}}{T_{cr}} = \frac{(1+\mu(P-1))(1-\alpha)+\alpha+\mu P\frac{\kappa(N,P)}{T}+\mu P\frac{(\mathbb{N}-1)C+\mathbb{N}R}{T}}{\frac{1}{\lambda}e^{R\lambda}(e^{\lambda(\tau+C)}-1)\frac{1-\alpha+\frac{\alpha}{P}+\frac{\kappa(N,P)}{T}}{\tau}} \tag{10}$$

$$\begin{aligned}\frac{\text{Perf}}{\text{Watt}} &= \frac{1}{\frac{1}{\lambda}e^{R\lambda}(e^{\lambda(\tau+C)}-1)\frac{1-\alpha+\frac{\alpha}{P}+\frac{\kappa(N,P)}{T}}{\tau}} \times \frac{\frac{1}{\lambda}e^{R\lambda}(e^{\lambda(\tau+C)}-1)\frac{1-\alpha+\frac{\alpha}{P}+\frac{\kappa(N,P)}{T}}{\tau}}{(1+\mu(P-1))(1-\alpha)+\alpha+\mu P\frac{\kappa(N,P)}{T}+\mu P\lambda(1+\frac{C}{\tau})(C+R)} \\ &= \frac{1}{(1+\mu(P-1))(1-\alpha)+\alpha+\mu P\frac{\kappa(N,P)}{T}+\mu P\lambda(1+\frac{C}{\tau})(C+R)}\end{aligned} \tag{11}$$

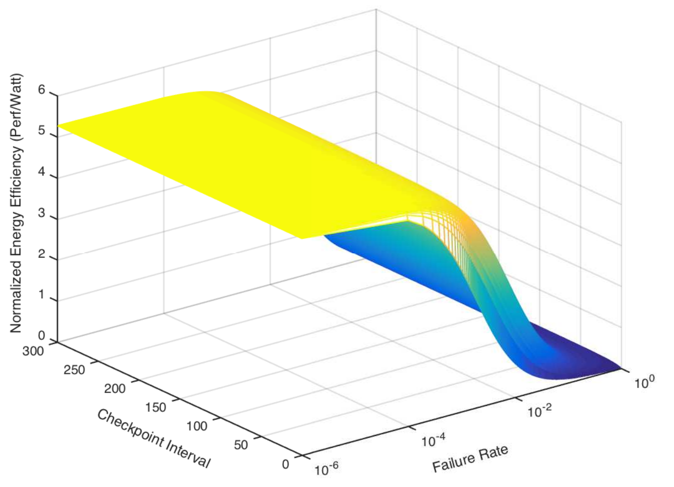

Fig. 4. Energy Efficiency of HPC Runs with Faults and Resilience Techniques (Checkpoint/Restart).

Figure 4 depicts the energy efficiency curve when the C/R technique is used in the presence of failures. Here we adopt some premise to facilitate the analysis: Per empirical measurement (see Section 5.2.5), we let $\mu = 0.6$, and without loss of generality, we assume the communication in the HPC run takes 50% of the total execution time, i.e., $\frac{\kappa(N,P)}{T} = 0.5$ (from the evaluation in Section 5, we found that values of $\frac{\kappa(N,P)}{T}$ did not alter the trend of energy efficiency). Moreover, we set $P = 50$ and $\alpha = 0.9$ to showcase an HPC environment. Regarding the C/R technique used, we assume checkpoint overhead $C = 10$ and restart overhead $R = 20$ in seconds. From Figure 4, we can see that when failure rates λ are comparatively small, i.e., in the range of $[10^{-6}, 10^{-4}]$, the energy efficiency is dominated by the impacts from the sequential and parallelized code and the communication per Equation (11). Variation of checkpoint intervals τ barely affects the energy efficiency in this case. However, the energy efficiency experiences a dramatic drop when the failure rates λ lie in around $[10^{-4}, 10^{-2}]$, and the drop becomes flattened when λ is further increased to a value close to 1. The impacts of τ are manifested when τ is small enough so that $\frac{C}{\tau}$ is larger than 1. We next discuss the scenario where energy saving techniques are used to improve the energy efficiency.

<u>Scenario 2</u>: HPC Runs with Faults and C/R, using DVFS to Save Energy

Likewise, we model the energy efficiency in the scenario with failures and C/R, and energy saving DVFS solutions in the presence of slack. From Equation (12), we can see that during the slack, using DVFS can indeed improve the energy efficiency due to the

$$\frac{\text{Perf}}{\text{Watt}} = \frac{1}{\frac{1}{\lambda}e^{R\lambda}(e^{\lambda(\tau+C)}-1)\frac{1-\alpha+\frac{\alpha}{P}+\frac{\kappa(N,P)}{T}}{\tau}} \times \frac{\frac{1}{\frac{1-\beta}{n_1}+\frac{\beta}{n_2}} \times \left(\frac{1}{\lambda}e^{R\lambda}(e^{\lambda(\tau+C)}-1)\frac{1-\alpha+\frac{\alpha}{P}+\frac{\kappa(N,P)}{T}}{\tau}\right)}{(1+\mu(P-1))(1-\alpha)+\alpha+\mu P\frac{\kappa(N,P)}{T}+\mu P\lambda(1+\frac{C}{\tau})(C+R)}$$

$$= \frac{\frac{1}{\frac{1-\beta}{n_1}+\frac{\beta}{n_2}}}{(1+\mu(P-1))(1-\alpha)+\alpha+\mu P\frac{\kappa(N,P)}{T}+\mu P\lambda(1+\frac{C}{\tau})(C+R)} \quad (12)$$

reduced power costs by a factor of $\frac{1}{\frac{1-\beta}{n_1}+\frac{\beta}{n_2}}$ (the only difference between Equation (11)), if the DVFS techniques incur negligible performance loss as described in Figure 1. n_1 and n_2 are determined by capability of the DVFS techniques and the amount of slack in the HPC run, and β depends on specific processor architectures. Note that during the non-slack time, power consumption with DVFS is the same as the original run.

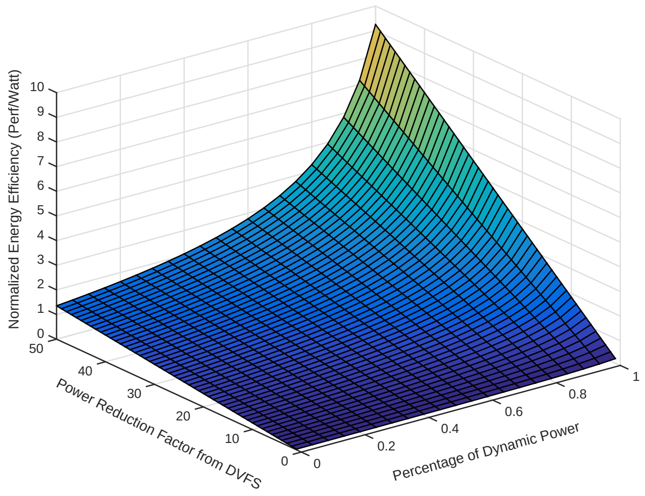

Fig. 5. Energy Efficiency of HPC Runs with Faults, Checkpoint/Restart, and DVFS Techniques.

As Equation (12) is derived based on Equation (11), we fix $\lambda = 10^{-5}$ and $\tau = 150$ to discuss the impacts from β and n_1/n_2 on the energy efficiency in the scenario with faults and DVFS. Given the known relationship that $n_1 = \Delta f^{0.75}$ and $n_2 = \Delta f^{2.5}$ (i.e., $n_2 = \Delta f^{1.75} n_1$), we rewrite the term using a joint parameter n as $\frac{1}{\frac{1-\beta}{n}+\frac{\beta}{\Delta f^{1.75}n}}$. As shown in Figure 5, we can see that a larger β or n value results in higher energy efficiency in general, which is consistent with Equation (12). We can also observe that for β values close to 1, the variation of energy efficiency is more manifested, since in which case the energy efficiency is dominated by the larger n_2 ($\Delta f^{1.75} > 1$), according to $\frac{1}{\frac{1-\beta}{n}+\frac{\beta}{\Delta f^{1.75}n}}$. Empirically, typical β values range from 0.65 to 0.8 for different processor technologies nowadays [Esmaeilzadeh et al. 2011] [Mair et al. 2007] [Rusu et al. 2010].

Scenario 3: HPC Runs with Faults and C/R, using Undervolting (Increased Failure Rates)

If the undervolting technique is leveraged to further save energy during non-slack time, compared to Scenario 2 where DVFS solutions apply to slack only, power can be saved by a factor of $\frac{1}{\frac{1-\beta}{n_1}+\frac{\beta}{n_3}}$ by undervolting, as modeled in Equation (13). However,

$$\frac{\text{Perf}}{\text{Watt}} = \frac{1}{\frac{1}{\lambda'}e^{R\lambda'}(e^{\lambda'(\tau'+C)}-1)\frac{1-\alpha+\frac{\alpha}{P}+\frac{\kappa(N,P)}{T}}{\tau'}} \times \frac{\frac{1}{\frac{1-\beta}{n_1}+\frac{\beta}{n_3}} \times \left(\frac{1}{\lambda'}e^{R\lambda'}(e^{\lambda'(\tau'+C)}-1)\frac{1-\alpha+\frac{\alpha}{P}+\frac{\kappa(N,P)}{T}}{\tau'}\right)}{(1+\mu(P-1))(1-\alpha)+\alpha+\mu P\frac{\kappa(N,P)}{T}+\mu P\lambda'(1+\frac{C}{\tau'})(C+R)}$$

$$= \frac{\frac{1}{\frac{1-\beta}{n_1}+\frac{\beta}{n_3}}}{(1+\mu(P-1))(1-\alpha)+\alpha+\mu P\frac{\kappa(N,P)}{T}+\mu P\lambda'(1+\frac{C}{\tau'})(C+R)} \tag{13}$$

lower supply voltage by undervolting given a chosen operating frequency by DVFS causes higher failure rates ($\lambda' > \lambda$) and shorter checkpoint intervals ($\tau' < \tau$) for tolerating raised failures, and thus inevitably results in longer execution time. Consequently, energy efficiency in this scenario can be either improved or degraded compared to Scenario 2, due to the coexisting power savings and performance loss. Generally, there exists a balanced undervolting scale that maximizes the energy efficiency in this scenario. We present details of this discussion in Section 4.6. Next we look into energy saving effects from leakage and dynamic power reduction factors n_1 and n_3.

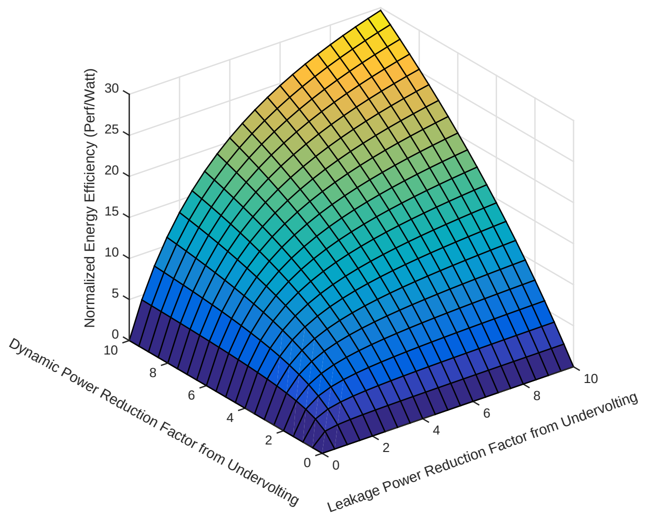

Fig. 6. Energy Efficiency of HPC Runs with Faults, Checkpoint/Restart, and Undervolting (Setup I).

Figure 6 shows the energy efficiency trend as n_1 and n_3 change, in the scenario with failures, C/R and undervolting techniques, with given values $\beta = 0.7$, $\lambda' = 10^{-1}$, and $\tau' = 15$ (both of failure rates and the optimal checkpoint interval are increased due to undervolting). Due to the fact that undervolting increases both n_1 and n_3 but $n_3 > n_1$, compared to the leakage power reduction factor n_1, the dynamic power reduction factor n_3 has greater impacts on the variation of energy efficiency – for a given n_1, larger n_3 values increase the energy efficiency more than larger n_1 values given a fixed n_3. From Figure 6, we can see that the energy saving effects from n_3 is almost linear for a given n_1, while for a fixed n_3 value, the impacts from n_1 are smaller: The energy efficiency curves of n_1 become flattened as n_1 increases, especially for small n_3 values. Regarding energy saving techniques, the energy efficiency trend is similar as in Scenario 2: Larger power reduction values n_1, n_2, and n_3 always improve energy efficiency monotonically. These observations can also be drawn from the causal term $\frac{1}{\frac{1-\beta}{n_1}+\frac{\beta}{n_3}}$ in Equation (13).

4.6. Energy Saving Effects of Typical HPC Parameters

With the energy efficiency models under different circumstances, we can further investigate the impacts of typical HPC parameters on the optimal energy efficiency individually. Since the parameters inherently affect each other, for highlighting the effects of individual factors, we fix other parameters likewise as the previous discussion.

4.6.1. Optimal Checkpoint Interval. Given a failure rate λ, for a certain C/R technique with the checkpoint and restart overhead C and R respectively, there exists an optimal checkpoint interval τ_{opt} that minimizes the total checkpoint and restart overhead. τ_{opt} is beneficial to improving performance, and thus also contributes to energy efficiency. Daly proposed a refined optimal value $\tau_{opt} = \sqrt{2C(\frac{1}{\lambda} + R)}$ for the condition $\tau + C \ll \frac{1}{\lambda}$ [Daly 2006]. Under the circumstance of undervolting, failure rates vary exponentially according to Equation (1). The previously proposed τ_{opt} does not apply to the case that Mean Time To Failure (MTTF, the reciprocal of failure rates) becomes comparable to the checkpoint overhead C. Daly also discussed a perturbation solution in [Daly 2006] that can be employed to handle this case of large MTTF due to undervolting:

$$\tau_{opt} = \begin{cases} \sqrt{\frac{2C}{\lambda}} - C & \text{for } C < \frac{1}{2\lambda} \\ \frac{1}{\lambda} & \text{for } C \geq \frac{1}{2\lambda} \end{cases} \tag{14}$$

Depending on the relationship between C and $\frac{1}{2\lambda}$, we use Equation (14) to calculate the most cost-efficient checkpoint interval in the scenario of undervolting. As already shown in Figure 4, larger checkpoint intervals τ barely enhance the energy efficiency while smaller ones do. Specifically, the difference between energy saving effects at nominal voltage and those at reduced voltage by undervolting is as follows: For nominal voltage with close-to-zero failure rates, e.g., at failure rate range [10^{-6}, 10^{-4}], the variation of τ barely affects the energy efficiency. For failure rates larger than 10^{-4}, the variation around small τ (compared to C) does affect the energy efficiency. The smaller τ incurs lower energy efficiency overall, since the resulting larger $\frac{C}{\tau}$ in Equation (11) makes the negative energy saving effects from C/R more manifested. For failure rates close to 1, the energy efficiency becomes insensitive to the variation of τ again, since in this case, within the C/R term $\mu P\lambda(1 + \frac{C}{\tau})(C + R)$ in Equation (11), λ is the dominant factor instead of τ. Generally for large values of λ, there exists an optimal value of τ for energy efficiency – any values greater than it barely affect the energy efficiency.

4.6.2. Optimal Supply Voltage. As stated earlier, values of supply voltage V_{dd} affect the performance and energy efficiency of HPC runs in two aspects: (a) V_{dd} values determine failure rates λ per Equation (1) and thus the optimal checkpoint interval τ_{opt} per Equation (14), and (b) V_{dd} values determine the leakage and dynamic power costs according to the relationship $P_s \propto V$ and $P_d \propto fV^2$. Nevertheless, the two aspects by nature conflict with each other in achieving high energy efficiency: Decreasing V_{dd} causes an exponential increase of λ, and thus a decrease of τ_{opt}, which results in higher overhead on error detection and recovery using C/R. Therefore, for energy saving purposes, larger values of V_{dd} should be adopted to minimize C/R overhead. On the other hand, larger V_{dd} brings higher leakage and dynamic power costs that degrade the energy efficiency overall, without affecting the solve time (operating frequency f is already selected per DVFS techniques and not modified by undervolting). Targeting the optimal energy efficiency for different HPC scenarios illustrated in Section 4.5, we tend to choose an optimal V_{dd} value that balances the two conflicting aspects above.

$$\frac{\mathsf{Perf}}{\mathsf{Watt}} = \frac{n'}{(1 + \mu(P-1))(1-\alpha) + \alpha + \mu P\frac{\kappa(N,P)}{T} + \mu P\lambda'(1 + \lambda' C)(C + R)} \tag{15}$$

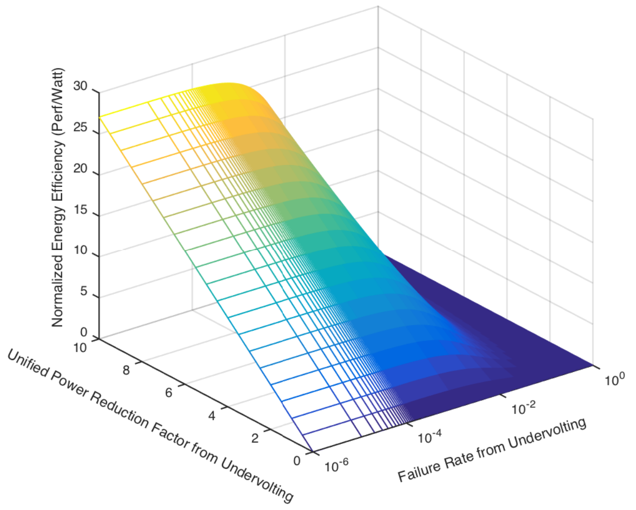

Fig. 7. Energy Efficiency of HPC Runs with Faults, Checkpoint/Restart, and Undervolting (Setup II).

Recall that Figure 4 clearly shows that the variation of λ greatly impacts the energy efficiency, as a result of undervolting. However, it does not reflect the interplay between the two conflicting factors (a) and (b). We thus theoretically quantify this interplay to see if there exists an optimal supply voltage for the given HPC configuration. Since undervolting reduces both n_1 and n_3, for simplicity of the discussion of the optimal V_{dd}, we relax the energy efficiency formula in Scenario 4 by letting $n_1 = n_3 = n'$, where n' is the *unified power reduction factor* from undervolting. In addition, we assume the optimal checkpoint interval τ_{opt} is always adopted (due to greatly raised λ by undervolting, we adopt the branch $\tau_{opt} = \frac{1}{\lambda}$ for $C \geq \frac{1}{2\lambda}$ in Equation (14)). The simplified energy efficiency is modeled in Equation (15), which stresses the interplay between the conflicting power savings and raised failure rates from undervolting.

We thus plot the curve of the simplified formula using the same assumption that fixes parameters other than n' and λ', as shown in Figure 7. It is clear to see that the larger n' is and the smaller λ' is, the higher the energy efficiency reaches. From the previous discussion, we know that achieving higher power savings from undervolting incurs higher failure rates, and vice versa. Therefore, selecting a voltage value for undervolting that balances the two factors will fulfill the optimal energy efficiency. We next empirically explore the optimal voltage for the highest energy efficiency.

5. EVALUATION

In this section, we present details of empirical evaluation for our speedup and energy efficiency models in the above different HPC scenarios on two HPC clusters. The goals of the evaluation are to experimentally demonstrate that: (a) The proposed models are well-grounded and accurate to predict the speedup, power and energy efficiency for scalable HPC runs, with prior knowledge on several HPC parameters, and (b) the proposed models are able to capture the interplay among typical HPC parameters in discussed HPC scenarios that ultimately affects the overall energy efficiency.

5.1. Experimental Setup

We conducted all the measurement-based experiments on a wide spectrum of HPC applications as our benchmarks, summarized in Table I. The benchmarks were selected

Table I. Benchmark details. From left to right: benchmark name, benchmark suite, benchmark description and test case used, problem domain, lines of code in the benchmark, parallelization system employed, and parallelized code percentage relative to the total.

Benchmark	Suite	Description and Test Case	Domain	LOC	Parallelized in	Percentage of Parallelized Code
MG	NPB	Solve a discrete Poisson equation using multigrid method (Class B).	discrete mathematics	2568	OpenMP/MPI	73.0%
CG	NPB	Estimate eigenvalue of a sparse matrix with conjugate gradient method (Class B).	numerical linear algebra	1864	OpenMP/MPI	93.3%
FT	NPB	Solve a partial differential equation using fast Fourier transform (Class B).	numerical linear algebra	2034	OpenMP/MPI	58.7%
EP	NPB	Generate Gaussian random variates using Marsaglia polar method (Class B).	probability theory and statistics	359	OpenMP/MPI	94.7%
MatMul	Self-coded	Matrix multiplication on two 10k×10k global matrices, saving into a third one.	numerical linear algebra	1532	OpenMP/MPI /Pthreads	99.2%
Chol	FT-ScaLAPACK	Cholesky factorization on a 10k×10k global matrix to solve a linear system.	numerical linear algebra	2182	MPI	92.7%
LU	FT-ScaLAPACK	LU factorization on a 10k×10k global matrix to solve a linear system.	numerical linear algebra	2892	MPI	61.6%
QR	FT-ScaLAPACK	QR factorization on a 10k×10k global matrix to solve a linear system.	numerical linear algebra	3371	MPI	76.5%
LULESH	DARPA UHPC	Approximate hydrodynamics equations using 512 volumetric elements on a mesh.	hydrodynamics	6014	OpenMP/MPI	14.6%
AMG	CORAL	An algebraic multigrid solver for linear systems on a 4×4×6 unstructured grid.	numerical linear algebra	3098	OpenMP/MPI	65.1%

from NPB benchmark suite [npb], LULESH [lul], AMG [amg] and our fault tolerant ScaLAPACK [Wu and Chen 2014], including a self-implemented matrix multiplication program [Tan et al. 2014], For assessing our goals, experiments were performed on two different-scale power-aware clusters: HPCL and ARC. Table II lists the hardware configuration of the two clusters. Since undervolting requires the permission of root users, we managed to perform undervolting-related experiments on HPCL only while all other experiments were conducted on both clusters. We measured the total power consumption for all compute nodes involved in the experiments, including both dynamic and leakage power costs, collected by Watts up? PRO [wat]. Energy consumption of HPC runs was measured using PowerPack [Ge et al. 2010], a comprehensive software and hardware framework for energy profiling and analysis of HPC systems and applications, which enables individual power and energy measurement on all hardware components such as CPU, memory, disk, motherboard, etc. of an HPC system. Before presenting experimental results of running the benchmarks for evaluating each proposed goal individually, we detail the implementation of our approach.

5.2. Implementation Details

5.2.1. Frequency-Directed DVFS. For demonstrating the effectiveness of our approach, we need to employ start-of-the-art energy efficient DVFS and undervolting techniques to evaluate the impacts from energy savings on the integrated energy efficiency of HPC systems (due to the similarity of energy saving trend between DVFS and undervolting, as shown in Equations (12) and (13), we only present results on undervolting). For different benchmarks, we used the most efficient DVFS approaches we developed in previous work: an adaptively aggressive energy efficient DVFS technique for NPB benchmarks [Tan et al. 2013], an energy efficient high performance matrix multiplication [Tan et al. 2014], and energy efficient distributed dense matrix factorizations [Tan and Chen 2015]. CPU DVFS was implemented via the CPUFreq infrastructure [cpu] that directly reads and writes CPU operating frequency system configuration files.

5.2.2. Fixed-Frequency Undervolting. For saving energy beyond the DVFS solutions, we leveraged an energy saving undervolting approach for HPC systems [Tan et al. 2015],

Table II. Hardware Configuration for All Experiments.

Cluster	HPCL	ARC
System Size (# of Nodes)	8	108
Processor	2×Quad-core AMD Opteron 2380	2×8-core AMD Opteron 6128
CPU Freq.	0.8, 1.3, 1.8, 2.5 GHz	0.8, 1.0, 1.2, 1.5, 2.0 GHz
CPU Voltage (Undervolting)	1.300, 1.100, 1.025, 0.850 V ($V_h/V_l/V_{safe\_min}/V_{th}$)	N/A
Memory	8 GB RAM	32 GB RAM
Cache	128 KB L1, 512 KB L2, 6 MB L3	128 KB L1, 512 KB L2, 12 MB L3
Network	1 GB/s Ethernet	40 GB/s InfiniBand
OS	CentOS 6.2, 64-bit Linux kernel 2.6.32	CentOS 5.7, 64-bit Linux kernel 2.6.32
Power Meter	PowerPack	Watts up? PRO

Table III. Northbridge/CPU FID/VID Control Register Bit Format.

Bits	Description
63:32, 24:23, 21:19	Reserved
32:25	Northbridge Voltage ID, Read-Write
22	Northbridge Divisor ID, Read-Write
18:16	P-state ID, Read-Write
15:9	Core Voltage ID, Read-Write
8:6	Core Divisor ID, Read-Write
5:0	Core Frequency ID, Read-Write

where undervolting is conducted for a production cluster by modifying corresponding bits of the northbridge/CPU frequency and voltage ID control register. The register values consist of 64 bits in total, where different bit fragments manage various system power state variables individually. Table III summarizes the register bit format [AMD 2012] for processors on the HPCL cluster: The Core Voltage/Frequency/Divisor ID fragments (CoreVid/CoreFid/CoreDid) are used for undervolting. As a general-purpose software level undervolting approach, the interested bits of register values are altered using the Model Specific Register (MSR) interface [msr]. Next we illustrate how to extract various ID fragments from specific register values and modify voltage/frequency of cores using corresponding formula. For instance, we input the register with a hexadecimal value 0x30002809 via MSR. From the bit format, we can extract the Core Voltage/Frequency/Divisor ID as 20, 9, and 0 respectively. Moreover, from [AMD 2012], we have the following architecture-dependent formulae to calculate voltage/frequency:

$$\text{frequency} = 100\text{MHz} \times (\text{CoreFid} + 16)/2^{\text{CoreDid}} \tag{16}$$

$$\text{voltage} = 1.550\text{V} - 0.0125\text{V} \times \text{CoreVid} \tag{17}$$

Given the register value 0x30002809, it is easy to calculate voltage/frequency to be 1.300 V and 2.5 GHz individually using the above equations. Using MSR, undervolting is implemented by assigning the register with desirable voltage values at the voltage bits. The frequency bits are unchanged to ensure fixed frequency during undervolting.

5.2.3. Failure Emulation by Injecting Hard and Soft Errors. As stated, due to hardware constraints of production processors, voltage cannot be scaled to the levels where observable errors occur. Alternatively, we emulate the real error cases as follows: Using the failure rates at error-triggering voltage levels calculated by Equation (1) (demonstrated [Tan et al. 2015] to be highly accurate compared to real failure rates [Bacha and Teodorescu 2013]), we inject hard and soft errors respectively at the calculated failure rates to emulate the incurred failures in HPC runs due to undervolting to such voltage levels. Specifically, hard error injection is performed by manually killing an arbitrary MPI processes during program execution at OS level (for general HPC applications). Soft error injection is however conducted at library level (for matrix-based HPC applications): We randomly select some matrix elements using a random number generator and modify values of the matrix elements to erroneous ones (soft error detection is done within the error checking module of matrix benchmarks). Without loss of generality, we adopt the general-purpose widely used resilience technique Checkpoint/Restart (C/R) [Duell 2003] to detect (hard errors only) and recover from the introduced failures (note that C/R is not the optimal resilience technique to protect from soft errors, and here we employ C/R to simplify the evaluation of the proposed models).

5.2.4. Power/Energy Measurement and Estimation. Limited extent of undervolting for production machines also prevents us from measuring power/energy directly for the case that real errors are observed from undervolting. Nevertheless, the emulated scaling undervolting scheme adopted from [Tan et al. 2015] allows us to utilize measured power costs at V_l (the lowest undervolted voltage for production machines) to estimate the power costs at V_{safe_min} (the lowest undervolted voltage for pre-production machines) based on the following power models, which enables our approach to work for general production machines. The three power models represent the baseline power costs at the highest frequency and voltage, the power costs at the highest frequency and the lowest voltage, and the power costs at the lowest frequency and voltage individually. Since our approach cannot undervolt to V_{safe_min}, P_m and P_l are empirically not measurable. We manage to obtain P_m and P_l as follows: Substituting V_{safe_min} in P_m and P_l with V_l, we measure the power costs P'_m and P'_l at V_l and P_h to solve constants AC', I_{sub}, and P_c using the three formula. With AC', I_{sub}, and P_c known, we can calculate P_m and P_l using V_{safe_min} in the formula of P_m and P_l. Given the power costs at V_{safe_min}, we can further calculate the energy costs when undervolting to V_{safe_min}.

$$\begin{cases} P_h = AC' f_h V_h^2 + I_{sub} V_h + P_c \\ P_m = AC' f_h V_{safe\_min}^2 + I_{sub} V_{safe\_min} + P_c \\ P_l = AC' f_l V_{safe\_min}^2 + I_{sub} V_{safe\_min} + P_c \end{cases} \tag{18}$$

5.2.5. Input Selection and Model Parameter Derivation. For extrapolating the power costs and resilience-aware speedup of HPC runs using the proposed power/speedup models, we need to derive the values of parameters in Equations (4) and (7). We also need to find out if the model parameters vary across different tests (i.e., with different problem sizes), since the parameter variation can greatly impact the accuracy of our models.

Table IV. Architecture-Dependent Power Constants in Our Models on HPCL/ARC Clusters.

Cluster	Single Core Peak Power Q	Non-CPU Power $\mathbb{C}$	CPU Idle Power Fraction μ
HPCL	11.49 Watts	107.02 Watts	0.63
ARC	N/A	N/A	0.75*

We first determine some application/architecture-dependent constants in our models. Power constants Q, $\mathbb{C}$, and μ were straightforward to obtain from empirical power measurement. Table IV lists measured values of the constants for the two clusters, where only HPCL was equipped with PowerPack that is able to isolate CPU power consumption from other components. We obtain the power costs of a single core at its peak performance Q by dividing the total CPU power costs with the CPU counts in a node. Note that for ARC we report the system idle power fraction instead of CPU. This number is slightly larger than the CPU idle power fraction for HPCL, which is reasonable since the power costs of other components barely vary between busy and idle modes [Ge et al. 2010]. Similarly, for a given C/R technique and an HPC application, we can empirically profile the checkpoint overhead C and the restart overhead R.

Table V. Calculated Failure Rates at Different Supply Voltage on the HPCL Cluster (Unit: Voltage (V) and Failure Rate (errors/minute)).

Supply Voltage	Calculated Failure Rate	Error Injection Needed?
1.300	3.649×10^{-6}	No
1.250	4.713×10^{-5}	No
1.200	5.437×10^{-4}	No
1.150	1.700×10^{-2}	No
1.100	0.397	Yes
1.050	2.717	Yes

For each supply voltage V_{dd} level, we can easily solve the failure rate λ under the given voltage using Equation (1). Table V shows the calculated failure rates at different voltage levels. Note that we refer to failure rates reported in [Bacha and Teodorescu 2013] when estimating the failure rates in our experiments, as their work was able to undervolt to the error-triggering voltage levels. Since the non-test HPC runs in our experiments finish in 71 - 266 seconds, at all voltage levels except for 1.100 V and 1.050 V, error injection is not needed according to the calculated failure rates.

Table VI. Communication Time to Total Time Ratio for All Benchmarks with Different Number of Cores and Problem Sizes on the ARC Cluster (Unit: $\kappa(N, P)$ (second) and T (second)).

$\kappa(N,P), T, \frac{\kappa(N,P)}{T}$	Run 1	Run 2	Run 3
MG	P=256, N=Class A 0.03, 0.05, **55.6%**	P=256, N=Class B 0.12, 0.22, **52.7%**	P=256, N=Class C 1.36, 2.32, **58.6%**
CG	P=64, N=Class A 0.06, 0.12, **52.2%**	P=64, N=Class B 2.33, 5.69, **41.0%**	P=64, N=Class C 5.64, 14.78, **38.2%**
FT	P=64, N=Class A 0.11, 0.33, **32.8%**	P=64, N=Class B 1.47, 4.20, **35.1%**	P=64, N=Class C 4.60, 16.57, **27.8%**
EP	P=16, N=Class A 0.30, 2.33, **13.0%**	P=64, N=Class B 0.36, 2.70, **13.4%**	P=256, N=Class C 0.37, 2.70, **13.5%**
MatMul	P=64, N=10k×10k 1.34, 6.12, **22.0%**	P=16, N=20k×20k 33.87, 160.37, **21.1%**	P=256, N=30k×30k 11.35, 41.44, **27.4%**
Chol	P=64, N=5k×5k 0.16, 0.983, **16.3%**	P=64, N=15k×15k 1.18, 5.79, **20.4%**	P=64, N=25k×25k 4.39, 23.25, **18.9%**
LU	P=64, N=10k×10k 0.77, 5.68, **13.5%**	P=64, N=20k×20k 7.40, 45.94, **16.1%**	P=64, N=30k×30k 21.33, 182.34, **11.7%**
QR	P=64, N=10k×10k 0.85, 7.59, **11.2%**	P=64, N=20k×20k 8.77, 60.90, **14.4%**	P=64, N=30k×30k 40.25, 236.76, **17.0%**
LULESH	P=8, N=64 1.66, 320.09, **0.52%**	P=64, N=512 4.12, 328.96, **1.25%**	P=216, N=1728 12.39, 336.90, **3.68%**
AMG	P=96, N=90×90×90 5.14, 107.33, **4.79%**	P=49152, N=150×150×150 N/A	P=960k, N=360×360×1080 N/A

Recall that when developing power and performance models with the consideration of communication, we rely on an assumption that the ratio between communication time $\kappa(N, P)$ and the total execution time T barely varies with the variation of problem size N and number of cores P. For each benchmark, $\frac{\kappa(N,P)}{T}$ was collected from multiple test runs (we assume that for each test run and experimental run, N and P are fixed during the execution). We selected different test runs by altering both N and P to ensure cross input validation. For NPB benchmarks, we used input size Class A, B, and C and changed number of cores used from 16, 64, to 256 (for LULESH, due to the application characteristics that P must be a cube of an integer, P ranges from 8, 64, 216). For matrix benchmarks, we chose global matrix sizes 10000×10000, 20000×20000, and 30000×30000. Table VI presents the values of the communication time versus the total execution time for all benchmarks with different N and P, except for AMG where increasing N and P exceeds available resources on our experimental platform. We can observe that for matrix benchmarks, $\frac{\kappa(N,P)}{T}$ has minor variation as the global matrix size N increases; for NPB benchmarks MG, CG, FT, and EP, there also exists such variability on $\frac{\kappa(N,P)}{T}$ for CG and FT, although it is stable for MG and EP. We calculate the percentage of parallelized code α (see the last column in Table I) in accordance with Equations (2) and (3), where the empirical speedup, P, $\kappa(N,P)$, and T values were obtained in advance from dynamic profiling of such test runs as well.

5.3. Validation of Modeling Accuracy

Using the above fine-grained tuned inputs and parameters, we can thus predict power costs and performance of HPC runs with our extended Amdahl's Law and Karp-Flatt Metric built in Section 4. Moreover, for validating the accuracy of our models, we next make head-to-head comparison between the real measured data on our experimental platform and the predicted data from the theoretical modeling, and calculate the modeling accuracy in terms of average error rate. Note that the following presented power and energy data are the total values for the whole HPC system evaluated.

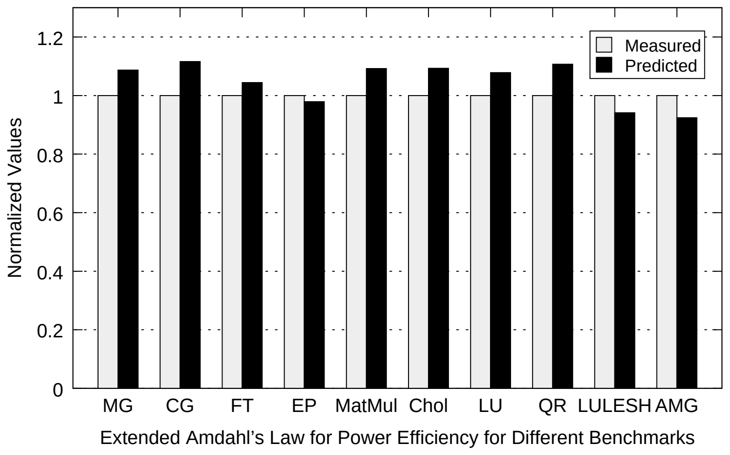

Fig. 8. Measured and Predicted System Power Consumption for HPC Runs on the HPCL Cluster.

5.3.1. Extended Amdahl's Law for Power Efficiency. Figure 8 shows the measured and predicted system power costs on HPCL for the ten benchmarks to evaluate the accuracy of our extended Amdahl's Law for power efficiency, i.e., Equation (4). Given all parameters in Equation (4) using the above measurement and derivation methods, we can easily extrapolate power costs for various HPC runs. As shown in Figure 8, the predicted data matches well with the measured data with an average error rate of 7.7% for three runs of each benchmark with different N and P. Generally, the errors from the power extrapolation for all benchmarks result from the variability of the term $\frac{\kappa(N,P)}{T}$ (see Table VI): For applications with stable $\frac{\kappa(N,P)}{T}$, the average error rate of the extrapolation is small (e.g., EP and LULESH), while for application having $\frac{\kappa(N,P)}{T}$ with minor variation, the extrapolation errors are more manifested (e.g., CG and QR).

5.3.2. Extended Karp-Flatt Metric for Speedup with Resilience. We also evaluated the accuracy of our extended Karp-Flatt speedup formula for the HPC runs with failures, equipped with the C/R technique. Figure 9 (we averaged the results from the two clusters) compares the measured and predicted speedup for the benchmarks running in such a scenario, based on Equation (7). From Figure 9, we can see that compared to our extended power model, our extended speedup model for HPC runs with failures and resilience techniques undergoes a higher average error rate (9.4%) against the

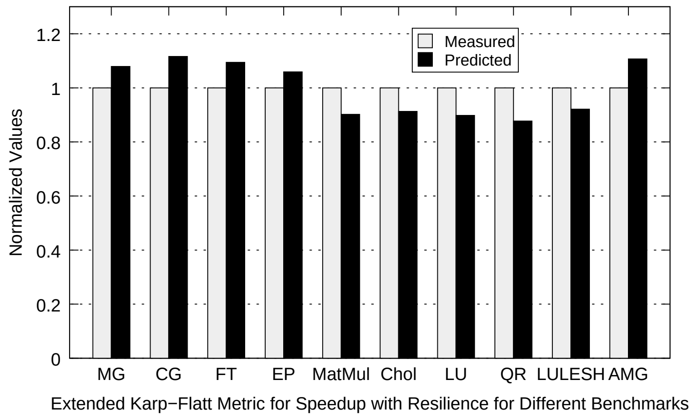

Fig. 9. Measured and Predicted System Speedup with Resilience for HPC Runs on the HPCL/ARC Clusters.

measurement. In addition to the empirical minor variation of assumed fixed $\frac{\kappa(N,P)}{T}$, another reason for the errors between the measured and predicted speedup comes from the failure distribution formula we adopt. As mentioned in [Daly 2006], the simplified solve time of applications with C/R in the presence of failures is an approximation to the solution, which may incur errors in the extrapolation. We notice that for matrix benchmarks (MatMul/Chol/LU/QR) and AMG, the predicted results have the most margins with the measured data, which indicates that our speedup model may be less accurate for applications with comparatively large-size checkpoints (i.e., C is large). Refining our model for this type of HPC applications may achieve higher accuracy.

5.4. Effects on Energy Efficiency from Typical HPC Parameters

Next we evaluate several critical HPC parameters discussed in Section 4.6 that have potentially significant impacts on the integrated energy efficiency in various HPC scenarios. We aim to empirically determine if there exist the optimal values of these parameters for the highest energy efficiency with resilience to validate our models.

5.4.1. Impacts from Checkpoint Intervals. According to the discussion in Section 4.6.1, we know that there exists an optimal checkpoint interval τ_{opt} for achieving the highest performance, defined by Daly's two sets of equations individually. One applies for the nominal voltage case and the other works for the undervolting case. For a given failure rate λ and a certain C/R technique (checkpoint overhead C and restart overhead R are known), τ_{opt} is calculated via $\tau_{opt} = \sqrt{2C(\frac{1}{\lambda} + R)}$ for the nominal voltage, and is calculated via Equation (14) for the reduced voltage by undervolting. Figure 10 shows the normalized system energy efficiency for a given λ (voltage is fixed) and different τ for MG and LULESH. We select to present the data of the two benchmarks because they have similar solve time according to our tests. All other benchmarks follow a similar pattern as the two. We injected errors in the failure rate at V', $V_{safe\_min} < V' < V_l$ (see Table II for core voltage specification), calculated to be 1.057 errors/minute per Equation (1). We also highlight in the figure the calculated optimal checkpoint interval τ_{opt}. From this figure, we can see that the optimal energy efficient τ slightly differs from the theoretical τ_{opt}. Note that in this case the optimal energy efficient τ is also the optimal τ for the highest performance, since the power savings do not change by using a fixed voltage for the same λ and thus only performance affects energy efficiency.

Moreover, we can see from Figure 10 that there exists the following fluctuation pattern: The energy efficiency drops greatly if checkpointing and restarting was applied

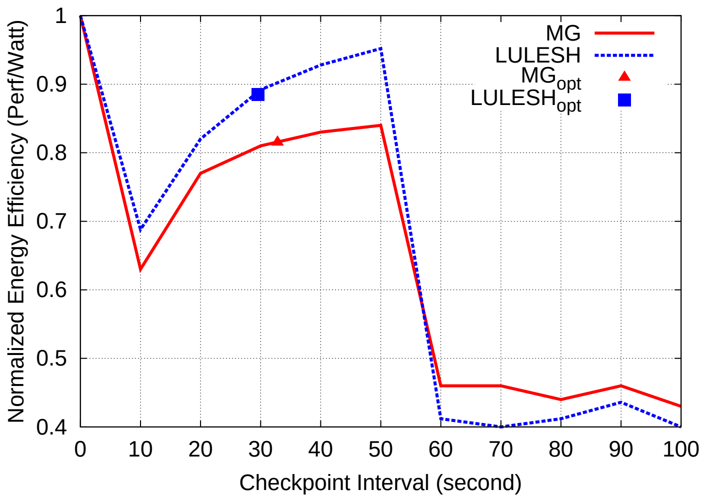

Fig. 10. Energy Efficiency (MG and LULESH) for Different Checkpoint Intervals on the HPCL Cluster.

(37.4% and 30.9% degradation for MG and LULESH respectively). Note that Figure 4 does not show a reference point where the normalized energy efficiency is 1 as in Figure 10. Since we injected a fixed number of errors, the restart overhead was also fixed. The energy efficiency increases if less checkpoints were used (i.e., longer checkpoint intervals), which matches well with Figure 4. We injected an error at the time around the 55^{th} second (the execution time of MG and LULESH runs is around 100 seconds), the energy efficiency quickly decreases for checkpoint intervals larger than this time period (not shown in Figure 4), since the re-execution of the program is necessary due to no checkpoints available (the first checkpoint has not made yet), which greatly increases the total execution time. Lastly, the energy efficiency barely changes for larger checkpoint intervals due to the same amount of checkpoints (1 in this case).

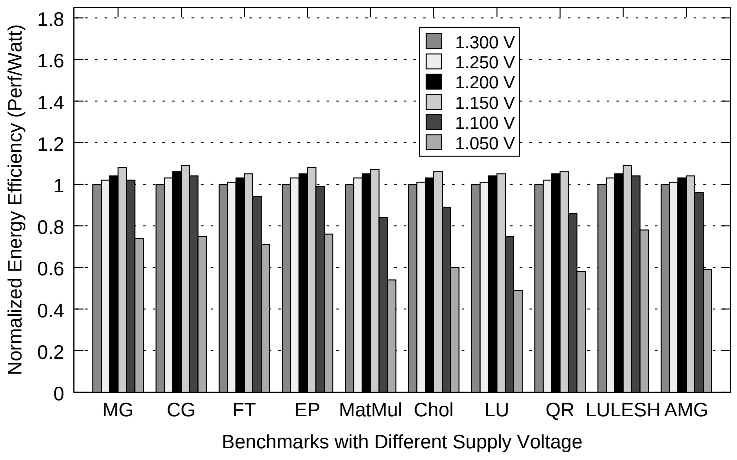

Fig. 11. Energy Efficiency for HPC Runs with Different Supply Voltage on the HPCL Cluster.

5.4.2. Impacts from Supply Voltage. As previously discussed in Section 4.6.2, for the scenario of undervolting in the presence of failures, theoretically there exists an optimal supply voltage value that balances the positive energy saving effects from the power savings and the negative energy saving effects from the C/R overhead. The highest energy efficiency is fulfilled at the optimal voltage. Figure 11 shows the experimental results on the normalized system energy efficiency under different supply voltage values for all benchmarks. The baseline case is the energy efficiency at the nominal voltage, 1.300 V in our experiments. We observe several findings from the figure: (a) For all benchmarks, the energy efficiency is improved by 1% - 9% when no errors were injected and dramatically degrades at the lowest voltage, i.e., 1.050 V. This is because the failure rates are exponentially raised as the voltage decreases per Equation (1), and at the lowest voltage failure rates are increased to values much larger than 1 error/minute (Table V lists all used failure rates at different voltage), (b) the optimal voltage for all benchmarks to achieve the highest energy efficiency is 1.150 V, although for some benchmarks (MG, CG, and LULESH), energy can be saved at 1.100 V as well. This is because 1.150 V is in our experiments the lowest voltage that incurs negligible increase in the number of failures during the HPC runs (no error injection is needed per the calculated failure rates and length of runs), and (c) for benchmarks with higher checkpoint and restart overhead C and R, such as MatMul, Chol, LU, QR, and AMG, no energy savings can be achieved from undervolting to V_l or lower compared to the baseline; for benchmarks with lightweight C and R, the energy efficiency at different voltage does not vary much (the highest improvement overall from undervolting is 9%), except for the lowest voltage 1.050 V that incurs significantly more failures. This indicates there exists comparable competition between the positive effects from power savings via voltage reduction, and the negative effects from the extra time costs on detecting and recovering from failures. The experimental results are very constructive to find the optimal voltage for the highest energy efficiency in the HPC environment. In general, selecting the smallest voltage that does not incur significantly more observable failures during an HPC run should fulfill the optimal energy efficiency.

6. CONCLUSIONS

Future HPC systems are required to be encompassing over performance, energy efficiency, and resilience, three crucial dimensions of great concerns by the HPC community nowadays. Enhancing one dimension of the three concerns does not necessarily improve the others, since the variation of some joint HPC parameters can be beneficial to one dimension, while be harmful to the others. There exists a lack of efforts that investigate the entangled effects among the three concerns, between energy efficiency and resilience in particular. In this paper, we quantify the interplay between energy efficiency and resilience for scalable HPC systems both theoretically and empirically. We propose comprehensive analytical models of the integrated energy efficiency with resilience, by incorporating power and resilience into Amdahl's Law and Karp-Flatt Metric, two classic HPC performance metrics. We also discuss the energy saving effects from typical HPC parameters that have inherent causal relationship with each other. Experimental results on two power-aware HPC clusters indicate that our models for scalable energy efficiency with resilience are accurate to capture the conflicting effects from typical HPC parameters. Our showcases on a wide spectrum of HPC benchmarks also demonstrate that it is feasible using our models to find the balanced HPC configuration for the highest integrated energy efficiency with resilience. We plan to discuss more types of hardware components and resilience techniques, and evaluate on larger scale HPC systems to further validate the capability of our models in the future.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their insightful comments and valuable suggestions, Prof. Rong Ge from Clemson University and Prof. Frank Mueller from North Carolina State University for providing the clusters, and in particular Prof. Rong Ge for granting the permission to undervolting.

REFERENCES

CPUFreq - CPU Frequency Scaling. https://wiki.archlinux.org/index.php/CPU_frequency_scaling.

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

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

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

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/.

Watts up? Meters. https://www.wattsupmeters.com/.

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

P. Alonso, M. F. Dolz, F. D. Igual, R. Mayo, and E. S. Quintana-Ortí. 2012. Reducing energy consumption of dense linear algebra operations on hybrid CPU-GPU platforms. In *Proc. ISPA*. 56–62.

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

G. M Amdahl. 1967. Validity of the single-processor approach to achieving large-scale computing capabilities. In *Proc. AFIPS Spring Joint Computer Conference*. 483–485.

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

G. Ballard, J. Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. 2012. Communication-optimal parallel algorithm for strassen's matrix multiplication. In *Proc. SPAA*. 193–204.

K. D. Bois, T. Schaeps, S. Polfliet, F. Ryckbosch, and L. Eeckhout. 2011. SWEEP: Evaluating computer system energy efficiency using synthetic workloads. In *Proc. HiPEAC*. 159–166.

A. S. Cassidy and A. G. Andreou. 2012. Beyond Amdahl's Law: An objective function that links multiprocessor performance gains to delay and energy. *IEEE Trans. Computers* 61, 8 (Aug. 2012), 1110–1126.

E. Chan, R. van de Geijn, W. Gropp, and R. Thakur. 2006. Collective communication on architectures that support simultaneous communication over multiple links. In *Proc. PPoPP*. 2–11.

D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. 1993. LogP: Towards a realistic model of parallel computation. In *Proc. PPoPP*. 1–12.

J. T. Daly. 2006. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems* 22, 3 (Feb. 2006), 303–312.

J. Duell. 2003. *The design and implementation of Berkeley lab's Linux Checkpoint/Restart*. Technical Report. Lawrence Berkeley National Laboratory.

H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. 2011. Dark silicon and the end of multicore scaling. In *Proc. ISCA*. 365–376.

R. Ge and K. W. Cameron. 2007. Power-aware speedup. In *Proc. IPDPS*. 1–10.

R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron. 2010. PowerPack: Energy profiling and analysis of high-performance systems and applications. *IEEE Trans. Parallel Distrib. Syst.* 21, 5 (May 2010), 658–671.

H. Jin and R. F. Van der Wijngaart. 2004. Performance characteristics of the multi-zone NAS parallel benchmarks. In *Proc. IPDPS*.

A. H. Karp and H. P. Flatt. 1990. Measuring parallel processor performance. *Commun. ACM* 33, 5 (May 1990), 539–543.

H. Kaul, M. Anders, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar. 2012. Near-Threshold Voltage (NTV) Design – Opportunities and Challenges. In *Proc. DAC*. 1153–1158.

I. A. Lee, C. E. Leiserson, T. B. Schardl, J. Sukha, and Z. Zhang. 2013. On-the-fly pipeline parallelism. In *Proc. SPAA*. 140–151.

J. Li and J. F. Martínez. 2005. Power-performance considerations of parallel computing on chip multiprocessors. *ACM Transactions on Architecture and Code Optimization* 2, 4 (Dec. 2005), 397–422.

H. Mair, A. Wang, G. Gammie, D. Scott, P. Royannez, S. Gururajarao, M. Chau, R. Lagerquist, L. Ho, M. Basude, N. Culp, A. Sadate, D. Wilson, F. Dahan, J. Song, B. Carlson, and U. Ko. 2007. A 65-nm mobile multimedia applications processor with an adaptive power management scheme to compensate for variations. In *Proc. VLSI Symposium*. 224–225.

E. Meneses, O. Sarood, and L. V. Kalé. 2012. Assessing energy efficiency of fault eolerance protocols for HPC systems. In *Proc. SBACPAD*. 35–42.

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

A. Rafiev, A. Iliasov, A. Romanovsky, A. Mokhov, F. Xia, and A. Yakovlev. 2014. Studying the interplay of concurrency, performance, energy and reliability with ArchOn – an architecture-open resource-driven cross-layer modelling framework. In *Proc. ACSD*. 122–131.

E. Rotem, R. Ginosar, U. C. Weiser, and A. Mendelson. 2014. Energy aware race to halt: A down to EARtH approach for platform energy management. *IEEE Computer Architecture Letters* 13, 1 (Jan. 2014), 25–28.

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

B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz. 2007. Bounding energy consumption in large-scale MPI programs. In *Proc. SC*. 1–9.

S. Rusu, S. Tam, H. Muljono, J. Stinson, D. Ayers, J. Chang, R. Varada, M. Ratta, S. Kottapalli, and S. Vora. 2010. A 45 nm 8-Core Enterprise Xeon® Processor. *IEEE Journal of Solid-State Circuits* 45, 1 (Jan. 2010), 7–14.

E. Solomonik, A. Bhatele, and J. Demmel. 2011. Improving communication performance in dense linear algebra via topology aware collectives. In *Proc. SC*. 77.

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

M. A. Suleman, O. Mutlu, M. K. Qureshi, and Y. N. Patt. 2009. Accelerating critical section execution with asymmetric multi-core architectures. In *Proc. ASPLOS*. 253–264.

L. Tan, L. Chen, Z. Chen, Z. Zong, R. Ge, and D. Li. 2013. Improving performance and energy efficiency of matrix multiplication via pipeline broadcast. In *Proc. CLUSTER*. 1–5.

L. Tan, L. Chen, Z. Chen, Z. Zong, R. Ge, and D. Li. 2014. HP-DAEMON: High Performance Distributed adaptive energy-efficient matrix-multiplication. In *Proc. ICCS*. 599–613.

L. Tan and Z. Chen. 2015. Slow down or halt: Saving the optimal energy for scalable HPC systems. In *Proc. ICPE*. 241–244.

L. Tan, Z. Chen, Z. Zong, R. Ge, and D. Li. 2013. A2E: Adaptively aggressive energy efficient DVFS scheduling for data intensive applications. In *Proc. IPCCC*. 1–10.

L. Tan, S. R. Kothapalli, L. Chen, O. Hussaini, R. Bissiri, and Z. Chen. 2014. A survey of power and energy efficient techniques for high performance numerical linear algebra operations. *Parallel Comput.* 40, 10 (Dec. 2014), 559–573.

L. Tan, S. L. Song, P. Wu, Z. Chen, R. Ge, and D. J. Kerbyson. 2015. Investigating the interplay between energy efficiency and resilience in high performance computing. In *Proc. IPDPS*. 786–796.

Y. Taur, X. Liang, W. Wang, and H. Lu. 2004. A continuous, analytic drain-current model for DG MOSFETs. *IEEE Electron Device Letters* 25, 2 (Feb. 2004), 107–109.

Z. Wang. 2009. Reliability speedup: An effective metric for parallel application with checkpointing. In *Proc. PDCAT*. 247–254.

M. Weiser, B. Welch, A. Demers, and S. Shenker. 1994. Scheduling for reduced CPU energy. In *Proc. OSDI*. 2.

C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu. 2008. Trading off cache capacity for reliability to enable low voltage operation. In *Proc. ISCA*. 203–214.

D. H. Woo and H.-H. S. Lee. 2008. Extending Amdahl's Law for energy-efficient computing in the many-core era. *Computer* 41, 12 (Dec. 2008), 24–31.

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

X. Wu, V. Deshpande, and F. Mueller. 2012. ScalaBenchGen: Auto-generation of communication benchmarks traces. In *Proc. IPDPS*. 1250–1260.

Y. Yetim, S. Malik, and M. Martonosi. 2012. EPROF: An energy/performance/reliability optimization framework for streaming applications. In *Proc. ASP-DAC*. 769–774.

L. Yu, D. Li, S. Mittal, and J. S. Vetter. 2014. Quantitatively modeling application resilience with the data vulnerability factor. In *Proc. SC*. 695–706.

Z. Zheng and Z. Lan. 2009. Reliability-aware scalability models for high performance computing. In *Proc. CLUSTER*. 1–9.

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