RSVP: Soft Error Resilient Power Savings at Near-Threshold Voltage using Register Vulnerability

Li Tan, Nathan DeBardeleben, Qiang Guan, Sean Blanchard, and Michael Lang
Ultrascale Systems Research Center [1]
Los Alamos National Laboratory
darkwhite29@gmail.com, {ndebard, qguan, seanb, mlang}@lanl.gov

***Abstract*—With the ever-growing scaling of computing capability, computing systems like supercomputers and embedded systems are bounded by limited power nowadays. Upon the mutually constrained nature between power efficiency and resilience, trade-offs of them have been extensively studied for achieving the optimal performance-power ratio, either under a certain power cap, or within the requirement of quality metrics of applications. Theoretically, running programs in the low-power mode of computational components (e.g., CPU/GPU) can lead to increasing on-chip failure rates in terms of register-level susceptibility to soft errors. However, experimentally, such errors may not arise due to register vulnerability – errors occur at non-vulnerable register access intervals are invalidated and thus will not propagate to later execution. In this work, leveraging register vulnerability, we investigate the validity of failure rates in computing systems at Near-Threshold Voltage (NTV), and empirically evaluate the practice of achieving optimal power savings without incurring observable number of soft errors during program runs. We propose the framework of *R*egi*S*ter *V*ulnerability based *P*ower efficiency (RSVP) for reliable and power efficient computing. Experimental results for a wide spectrum of applications on a power-aware simulated platform demonstrate the power saving capability of RSVP, by 11.2% on average, without incurring runtime soft errors at the optimal NTV level for power savings.**

I. Introduction

Considering ever-growing power bills and limited power supply capability of electrical facilities, power efficiency has already become a first-class citizen in scalable and cost-efficient High Performance Computing (HPC) systems, including supercomputers, embedded/mobile systems. Numerous software/hardware solutions have been proposed to achieve power savings at different levels of system abstraction, with little performance degradation. Operating HPC runs in the low-power mode is one of the most promising solutions to system power efficiency, with the well-studied trade-off of system reliability – Various types of errors can be incurred into any stages of HPC runs, potentially leading to severe system-level faults. Resilience to such errors in HPC systems can be expensive in terms of software costs (e.g., performance loss) and hardware costs (e.g., extra hardware). For reliability purposes, hardware components such as processors are conservatively equipped with high supply voltage for stable operations without errors. However, considerable power savings can be achieved by aggressive but appropriate voltage reduction, like near-threshold voltage computing [1].

[1]This work was performed at the Ultrascale Systems Research Center at Los Alamos National Laboratory, assigned the LANL identifier LA-UR-17-23454.

CMOS-based hardware components such as CPU, GPU, and memory are generally the dominant power consumers in a computing system. Lowering supply voltage of them can effectively reduce power costs of the system, given that power consumption of these components is proportional to the product of operating frequency and supply voltage squared [2], (i.e., voltage reduction has a greater impact on power savings than frequency reduction). Existing efforts extensively show that Dynamic Voltage and Frequency Scaling (DVFS) techniques can effectively save power/energy when the peak performance of components (CPU [3] [4], GPU [5] [6], memory [7] [8], and network [9]) is not necessary. Nevertheless, a critical limitation of existing DVFS work lies in the fact that the employed DVFS techniques are essentially *frequency-directed*: Voltage and frequency are scaled down together in the presence of hardware idle time, i.e., *slack* [10], which fails to fully exploit potential power saving opportunities during HPC runs. As a remedy solution, *undervolting* [11] [12] [10] has been employed to further reduce power costs, which reduces voltage of computational components independent of frequency scaling to maintain performance. Namely, the undervolting-enabled hardware components are supplied with a voltage that is lower than the one paired with a given frequency used in DVFS.

As a generic NTV approach applicable during any stages of HPC runs, undervolting is capable of saving extra power in addition to that achieved using DVFS during hardware slack. Nevertheless, the trade-off is increased failure rates [1] of the components with lowered voltage, e.g., register-level vulnerability to soft errors [13]. Quantifying register vulnerability is beneficial to determine the extent of power savings when NTV techniques are used. In this work, we investigate the validity of failure rates at Frequency-independent NTV (FiNTV for short), proposing a framework of `RegiSter Vulnerability based Power` efficiency (`RSVP` for short) to save the most power without incurring observable number of soft errors during HPC runs. In summary, the contributions of this paper include:

- We quantify register vulnerability metrics to exclude invalid soft errors at register level, in order to achieve the optimal power savings at FiNTV without incurring observable amount of soft errors during HPC runs;
- We devise an integrated quantitative framework `RSVP` to calculate register vulnerability of HPC runs, and utilize fine-grained fault and power models to estimate adjusted failure rates and power savings at FiNTV;
- Our approach is experimentally evaluated on a power-aware simulated platform for a number of HPC applications to save 11.2% in power on average, without affecting program outputs by incurring soft errors.

The remainder of the paper is organized as follows: Brief background knowledge is introduced in Section 2, and Section 3 discusses related work. Details of the proposed empirical RSVP framework are presented in Section 4. Experimental results are provided in Section 5. Section 6 concludes.

II. Background

A. Register Vulnerability for HPC Systems

For high performance computing, modern microprocessors employ a large number of quickly accessible registers with multiple ports. For instance, Alpha 21264 processor utilizes 80 integer and 72 floating-point physical registers [14]; Intel Itanium processor uses 128 general-purpose registers, 128 floating-point registers, 64 predicate registers, and 8 branch registers [15]. Registers are the most frequently accessed microarchitectural component, since in general the employed registers are on the critical path of program execution. Regardless of the intensive access of registers, enabled by state-of-the-art nano-technology, lower supply voltage, higher operating frequency, and higher density of processors cause increasing chip susceptibility to soft errors, i.e., a type of transient errors such as bit-flips and logic circuit errors. For HPC systems, register-level vulnerability to soft errors is of the greatest concern, since errors occurred in registers can easily propagate to other critical functional units of the systems such as ALU, FPU, and memory, and consequently lead to more severe system-level faults [16], e.g., *crashes*, *hangs*, and *Silent Data Corruption* (SDC) that spread throughout the whole system.

B. Near-Threshold Voltage Reduction

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

Nowadays, hardware vendors have launched cutting-edge nano-technology that enables processors to be supplied with a significantly low voltage, e.g., using Intel's Near-Threshold Voltage (NTV) design [1], the 32nm low-leakage Intel Claremont processor can operate from 280 mV at 3 MHz and scale up to 1.2 V at 915 MHz, with the minimum power of 2 mW. Note that this NTV technique requires low supply voltage accompanied by low operating frequency as well, different from our FiNTV technique in this work. Empirically, protected by the OS, production processors are generally locked and will typically shut down or crash, when near-threshold voltage reduction without frequency reduction is performed. Experimentally, for a customized pre-production Intel Itanium II 9560 processor [12], FiNTV has been validated to save power effectively at the cost of increased ECC memory errors, as the first practical undervolting effort on real machines. Tan *et al.* [10] [17] relaxed the hardware requirements of undervolting to general production machines by emulated scaling, evaluated on an HPC cluster with production AMD Opteron 2380 processors. Figure 1 shows different voltage levels voltage reduction techniques are capable of scaling to, where V_h/V_l, V_{ntv}, V_{th} refer to the maximum/minimum voltage (paired with the maximum/minimum frequency), a near-threshold voltage, and the threshold voltage individually. General production processors can be scaled within the voltage range $[V_l, V_h]$, while for customized pre-production processors, they can be supplied with a near-threshold voltage V_{ntv}. Our goal is to enable production processors with FiNTV for the maximum power savings, although empirically they cannot be supplied with V_{ntv} without accompanying frequency reduction.

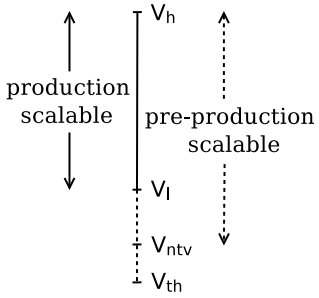

Fig. 1. Voltage Levels for Empirical Processor Voltage Reduction.

C. The gem5 Simulator

The gem5 simulator [18] is a modular platform that simulates system-level architecture as well as processor microarchitecture, extensively used for computer-system architecture research. It is a highly configurable cycle-accurate simulation framework that encompasses multiple Instruction Set Architectures (ISA) (Alpha, ARM, SPARC, MIPS, POWER, and x86), and diverse CPU models (simple one-CPI, detailed in-order, detailed out-of-order, and KVM-accelerated). It also supports two operation modes: System Call Emulation (SE) and Full System (FS), where the SE mode simulates user-space only programs while system services are provided directly by the simulator (system calls are emulated by passing to the host OS), and the FS mode is capable of simulating a complete system with devices and OS. Specifically, gem5 models the processor microarchitectural components including rename map, history/reorder buffer, instruction/load-store/fetch/decode queue, and allows to track the read and write operations at fine-grained bit-level to the components, which is thus the basis of modeling register vulnerability accurately. In this work, our implementation using gem5 supports both SE and FS operation modes, and also two mainstream ISA, ARM and x86.

III. Related Work

There exist a large body of work on vulnerability factors at software level, e.g., Program Vulnerability Factor (PVF) [19] and enhanced Program Vulnerability Factor (ePVF) [20], and at hardware/microarchitecture level, e.g., Architectural

Vulnerability Factor (AVF) [21] and Register Vulnerability Factor (RVF) [22]. Next we overview existing efforts based on RVF and studying power efficiency using vulnerability factors.

REGISTER VULNERABILITY FACTOR: Carretero *et al.* [23] presented a hardware mechanism using linear regression to estimate AVF of register files at low costs. Their approach needs to track microarchitectural events, as well as physical and architectural registers, while our approach only need to track accesses to physical and architectural registers. They further optimized their approach by sampling while accuracy was affected as a trade-off. Yan *et al.* [22] devised two cost-effective compiler-guided techniques to reduce RVF up to 9.5%, specifically, hyperblock-based instruction re-scheduling and reliability-oriented register assignment. They claimed that their solution can effectively protect register files against transient errors. Although RVF values are lowered, transient errors may still arise while our work focuses on pinpointing the optimal voltage that does not incur observable number of soft errors. Lee *et al.* [24] proposed another compiler approach based on interprocedural program analysis to reduce RVF values by around 30% on average. This solution benefited from integer linear programming and an efficient heuristic algorithm. However, their goal is to reduce RVF while ours is to save the maximum power leveraging RVF. As a compiler-based approach, they used extensive program analysis techniques, while we conduct analysis on the power and fault models.

POWER SAVING USING VULNERABILITY FACTORS: Lee *et al.* [25] proposed an energy efficient soft error protection scheme based on compiler-managed register vulnerability for embedded systems. They formulated and solved several compiler optimization problems to save energy for register file protection schemes by up to 24%. This effort differs from our work by the focus on compile-time optimization and embedded benchmarks. Considering GPU register file reliability, a recent study [26] modeled the process variation impacts on GPU register files at different voltage levels, and presented an architectural solution that utilizes register dead time for reliable operations from unreliable register files at low voltage. In our work, we focus on register access and its vulnerability impacts on the validity of failure rates in HPC environments. Fazeli *et al.* [27] presented a circuit-level soft error tolerant technique for register files in embedded processors, where the most vulnerable registers are cached in a small reliable register cache. Clock gating techniques was exploited to enable low-power register operations. Experimental results showed the AVF values of register files become about 1% at low-power, area, and performance overhead to the evaluated LEON processor. This effort employed clock gating as the power reduction solution, different from our FiNTV technique, and they targeted embedded processors instead of HPC clusters.

IV. RSVP: QUANTIFICATION OF REGISTER VULNERABILITY, FAILURE RATES, AND POWER SAVINGS AT THE OPTIMAL NEAR-THRESHOLD VOLTAGE

In this section, we explore the quantification of register vulnerability, and discuss models of failure rates and power costs in terms of supply voltage, and how NTV impacts the models as a joint parameter for resilience and power efficiency. We elaborate the steps of power saving using `RSVP`. Note that the terms *failure*, *fault*, and *error* are used interchangeably.

A. Register Access Model and Register Vulnerability Metrics

Registers are the most frequently accessed microarchitectural component, due to its nature of enabling high performance data access between CPU and memory. In general, register accesses are on the critical path of program execution, which means errors occurred in register files can easily propagate to other critical system functional units involved in program execution, e.g., ALU/FPU within CPU, and storage components, e.g., caches and memory cells. If values in register files are contaminated, more severe system-level faults can be caused as a consequence of extensive error propagation [16]. Therefore, it is crucial to protect registers from soft errors. It is well studied that not all soft errors occurred during register accesses will lead to observable erroneous system-level outputs, due to the factor of register vulnerability [22], a factor that measures the probability that soft errors can effectively propagate to other system components. In general, soft errors arising at non-vulnerable register access intervals are overwritten/invalidated by new register values, and thus will not manifest themselves in later program execution. Such soft errors need not to be accounted for a source of potential system-level faults, as an over-estimation.

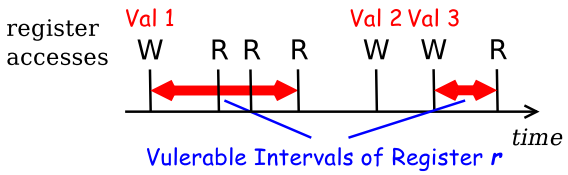

Fig. 2. An Example of Write/Read Accesses to Register ***r***.

Figure 2 depicts an example of register accesses to register ***r*** occurred in HPC runs, where W and R represent register *write* and register *read* individually, and Val1/Val2/Val3 represents register values written during the program execution. Typically, a register *write* operation is followed by zero or more register *read* operations. The lifetime of a register value spans from the first access (must be a *write*) to the last access (can be either a *read* or a *write*). *Vulnerable* intervals of a register value refer to the time periods ranging from a *write* w_1 to the last *read* before the next *write* w_2. Any time periods otherwise are *non-vulnerable* by definition. Therefore, any soft errors occurred after w_1, however falling into a *non-vulnerable* interval before w_2, is overwritten/invalidated by w_2, and thus cannot be propagated to later program execution. A well-known metric to measure register vulnerability is referred to as Register Vulnerability Factor (RVF), similar to other vulnerability metrics such as AVF and PVF. Equation 1 formulates the calculation of original RVF, under the circumstance of the nominal frequency/voltage of computational components, i.e., in their nominal power mode. The original RVF is calculated as the average of the sum of vulnerable intervals of register r_i divided by the lifetime of register r_i ($0 \leq i < n$, where n denotes the number of registers used in an HPC run).

$$\phi^{orig}_{RegVul} = \underset{i=0}{\overset{n}{\text{avg}}} \left(\frac{\sum VulTime(r_i)}{LifeTime(r_i)} \right) \quad (1)$$

When FiNTV is employed during HPC runs, vulnerable intervals of registers remain the same, since operating frequency of computational components is kept unchanged throughout

the runs and thus the time of register *write* and *read* operations is the same as that in the nominal power mode. Therefore, in Equation 2, we hypothesize that the RVF in FiNTV (low-power mode) equals to the RVF in the nominal power mode:

$$\phi_{RegVul}^{ntv} = \phi_{RegVul}^{orig} \tag{2}$$

B. Voltage-directed Failure Rate Modeling and RVF-based Failure Validation

Existing studies indicate failures of combinational logic circuits comply with a Poisson distribution, determined by both operating frequency and supply voltage [28]. Our previous work summarized a handy formula of average failure rates in terms of supply voltage only [10], by substituting frequency with voltage (frequency has a non-linear positive correlation with voltage [29]) in the general equation stated in [30]:

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

Constants d and β are architecture-dependent, reflecting the sensitivity of failure rate variation with frequency/voltage scaling. V_{th} and V_{dd} are threshold voltage and supply voltage respectively. f_{max} and f_{min} are the maximum frequency and the minimum frequency within the scope of DVFS individually. As mentioned in Section 2, near-threshold voltage reduction is disabled by the OS for production machines as hardware protection mechanism. Therefore real errors at NTV levels cannot be observed experimentally, since our target is production machines used in real HPC clusters. Based on the observed error data for an Intel pre-production processor [12], Equation 3 was demonstrated to be accurate to model failure rates at different voltage levels [10], and is thus adopted in this work for failure modeling when the optimal NTV level is determined during failure rate validation.

However, the failure rates calculated by Equation 3 can be inaccurate, due to the fact that only a fraction of register access intervals are vulnerable – errors fall into the non-vulnerable intervals are overwritten/invalidated by later register *writes* and thus have no chance to affect program outputs. Straightforward, Equation 4 shows the calculation of adjusted failure rates by considering RVF in FiNTV, where only errors occurred in vulnerable intervals of all registers are regarded as valid sources of potential system-level faults.

$$\lambda(f, V_{dd})' = \lambda(f, V_{dd}) \times \phi_{RegVul}^{ntv} \tag{4}$$

$\lambda(f, V_{dd})'$ denotes the real valid failure rate when FiNTV is employed, and it is used to obtain the optimal power savings at FiNTV, as shown later in Figure 3 and Algorithm 1.

C. Measurement/Estimation-based Power Modeling

For an HPC system, the following power model is used to calculate the nodal power consumption [31], assuming the processor is DVFS-enabled and FiNTV-enabled:

$$\begin{aligned} P &= P_{dynamic}^{processor} + P_{leakage}^{processor} + P_{leakage}^{other} \\ &= ACfV_{dd}^2 + I_{sub}V_{dd} + I'_{sub}V'_{dd} \end{aligned} \tag{5}$$

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

D. Soft Error Resilient Power Saving using RSVP

We now demonstrate that RSVP is an effective approach for HPC power savings without incurring errors from low-power operations. Figure 3 overviews the `RSVP` framework, consisting of *dynamic profiling* of HPC runs to obtain power data and RVF values for adjusted failure rates, and *static estimation* of the maximum power savings at FiNTV without incurring observable number of soft errors in HPC runs.

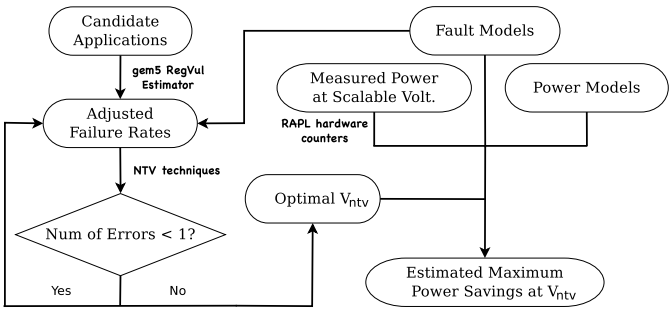

Fig. 3. Overview of the `RSVP` Framework.

The goal of using `RSVP` is to calculate adjusted failure rates that exclude the errors fall into the non-vulnerable intervals of registers used in HPC runs, and leverage the adjusted failure rates to find out the optimal supply voltage V_{ntv}^{opt} that does not incur observable number of soft errors in HPC runs, thus achieving the maximum power savings without errors. As shown in Figure 3, utilizing the fault models presented in Section III.B and an RVF estimator built on top of the gem5 simulator, we obtain adjusted failure rates at different NTV levels. Given an HPC run and the failure rates, number of errors during the HPC run can be calculated. In order to maximize power savings, we keep lowering down voltage if the calculated number of errors is still below 1. Otherwise, the lowest supply voltage that does not cause more than 1 error during HPC runs is adopted as the optimal NTV level V_{ntv}^{opt}. Based on measured power data using embedded on-chip hardware counters and the power estimation models presented in Section III.C, we can easily estimate the power savings when supply voltage is scaled down to V_{ntv}^{opt}.

Specifically, the workflow of `RSVP` proceeds as shown in Algorithm 1: Given a DVFS-enabled and FiNTV-enabled platform, there exist a group of scalable frequency/voltage levels of processors, from which failure rates can be calculated using the fault models in Equation 3. We test run a candidate HPC application to obtain the RVF value, the nominal power costs, and the execution time during the HPC run, and solve the constants in Equation 5 by sampling different power

Algorithm 1: Calculation of the Maximum Power Savings at the Optimal NTV Level without Incurring Observable Number of Soft Errors at Runtime

Input: A candidate application app, frequency/voltage pairs used in DVFS: $\{f_h/V_h, f_m/V_m, f_l/V_l\}$, near-threshold voltage V_{ntv} between V_l and V_{th}, and the calculated failure rate λ_{ntv} at V_{ntv}.

Output: Maximum power savings at the lowest near-threshold voltage V_{ntv}^{opt} without incurring observable number of soft errors.

```
begin
    make the system idle, operating at the nominal f/V
    for f/V ∈ {f_h/V_h, f_m/V_m, f_l/V_l} do
        scale to the selected f/V and run app
        measure power P = ACfV^2 + I_sub V + P_c
        if f_h/V_h then
            measure power P_h and execution time T
    solve AC, I_sub, and P_c, given P_h, P_m, and P_l
    for V ∈ (V_th, V_l) do
        scale to the selected V and run app
        calculate φ^ntv_RegVul and λ(f, V_dd)'
        if ⌈GetErrNum(λ(f, V_dd)', T)⌉ < 1 then
            measure power P_ntv
            V^opt_ntv = V
        else
            /* Errors arise (may need fault injection) */
    P_sav ← (P_h − P_ntv)/P_h × 100%
    return V^opt_ntv
```

modes. When FiNTV is applied, per the scaled voltage, we calculate adjusted failure rates from original failure rates and RVF values. Based on the adjusted failure rates and execution time of HPC runs, we can further get the number of errors during HPC runs. Therefore, the optimal NTV level V_{ntv}^{opt} can be obtained by scaling down voltage until the number of errors with FiNTV is greater than 1. P_{sav} refers to the percentage of power savings relative to the nominal power costs, achieved by near-threshold voltage reduction. Function $GetErrNum()$ abstracts the straightforward procedure of calculating the number of runtime errors at FiNTV.

Note that the output power P_{ntv} is obtained by a measurement/estimation-based method: All constants used in Equation 5 are solved from measured power data, while power costs at unscalable V_{ntv} are calculated from the same equation with f_h and V_{ntv} in place. Leveraging both measurement and fine-grained power models, this method is more accurate than purely estimation-based methods, and can emulate power costs at unscalable voltage for production machines in HPC clusters.

V. Evaluation

In this section, we present details of experimental evaluation on our RSVP approach for a wide spectrum of HPC applications running on a power-aware simulated platform based on the gem5 simulator. Overall, the empirical study aims to showcase that: (a) RSVP is capable of calculating RVF values with accuracy for both sequential and parallel versions of HPC applications, and (b) RSVP is capable of identifying the optimal NTV level that saves the maximum power costs without incurring observable number of soft errors at runtime.

A. Experimental Setup

TABLE I. Hardware Configuration for All Experiments.

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

Experiments were conducted on a wide scope of nine mainstream HPC applications to benchmark our approach, from domains of numerical linear algebra, cryptography, and financial analytics, summarized in Table III. The benchmarks were selected from PARSEC [32], MiBench [33], and NPB [34] benchmark suites, including one self-coded implementation of matrix multiplication. Table I lists hardware configuration of the experimental server where all gem5 simulation ran. It is equipped with two Intel Xeon E5-2660 processors based on the Haswell microarchitecture [35]. Various voltage levels are scalable using our FiNTV techniques. Power consumption of simulated HPC runs on the server was collected by Intel RAPL hardware counters [36] that reports in-band power costs of the total processor package and DRAM, which can be used to estimate the total system power costs during HPC runs (note that power savings reported henceforth are system-wide). Using RAPL and its API, it is convenient and lightweight to emulate power consumed during HPC runs without physical power meters attached. Table II lists detailed microarchitectural parameters used in the gem5 simulation. We evaluated our approach on both SE and FS modes and on two popular ISA, ARM and x86. Before presenting experimental results, we detail the implementation of techniques used in our approach.

TABLE II. Simulator Configuration for All Experiments.

Operation Mode	System Call Emulation and Full System
OS (Full System Mode)	Ubuntu 14.04, 64-bit Linux kernel 3.16.0
Memory (Full System Mode)	512 MB
Instruction Set Architecture	ARM and x86
L1 Cache Size	8 KB (data), 4 KB (instruction)
L1 Cache Associativity	2
L2 Cache Size	32 KB
L2 Cache Associativity	4
Cache Line Size	64

B. Empirical Implementation

As discussed and shown in Figure 1, production computing nodes used in HPC clusters are protected by the OS from scaling down to voltage levels lower than V_l. Keeping the processor frequency constant, we manage to implement Frequency-independent Near-Threshold Voltage reduction (FiNTV) below V_l on the experimental Intel processors by directly modifying the core voltage Model-Specific Register (MSR, specifically MSR_PERF_STATUS), defined in [36]. The high 16-bit (i.e., VID) of the 48 bit register value determines the P-state core voltage, which can be calculated as $\text{VID}/2^{13}$. The optimal

TABLE III. BENCHMARK DETAILS. FROM LEFT TO RIGHT: BENCHMARK NAME, BENCHMARK SUITE, BENCHMARK DESCRIPTION AND TEST CASE USED, PROBLEM DOMAIN, MAIN FUNCTION, EXECUTION TIME PERCENTAGE OF THE FUNCTION RELATIVE TO THE TOTAL, AND PARALLELIZATION SYSTEM USED.

Benchmark	Suite	Description and Test Case	Domain	Function	Runtime (in %)	Parallelized by
FT	NPB	Solve partial differential equations using fast Fourier transform, Class A.	Numerical Linear Algebra	cffts1	79.7%	OpenMP
LU	NPB	Solve partial differential equations using Lower-Upper symmetric Gauss-Seidel, Class A.	Numerical Linear Algebra	ssor	89.7%	OpenMP
CRC32	MiBench	Calculate 32-bit CRC for 10 files of total size 3.065 MB.	Coding Theory	main	100.0%	Pthreads
bitcount	MiBench	Perform bit counting functions using bit lookup table with a 75000 bit set.	Coding Theory	main	100.0%	OpenMP
patricia	MiBench	Insert/remove nodes, and search in a Patricia trie for IP addresses and netmasks.	Computation Theory	pat_search	86.9%	OpenMP
sha	MiBench	Perform the Secure Hash Algorithm with a security string file of 311824 bytes.	Cryptography	sha_stream	91.2%	OpenMP
MatMul	Self-coded	Calculate matrix multiplication on two 10k×10k global matrices, saving into a third one.	Numerical Linear Algebra	mmm	99.7%	MPI (gem5 *dist* tool)
blackscholes	PARSEC	Perform option pricing with the Black-Scholes partial differential equation.	Financial Analytics	bs_thread	82.4%	Pthreads
swaptions	PARSEC	Compute prices of a portfolio of 64 swaptions with 20000 Monte Carlo simulations.	Financial Analytics	main	100.0%	Pthreads

TABLE IV. ORIGINAL/ADJUSTED (BITCOUNT) FAILURE RATES AT VOLTAGE LEVELS (UNIT: VOLTAGE (V); FAILURE RATE (ERRORS/MIN.)).

Core Supply Voltage	Original Failure Rate (Calc. by Equation 3)	Adjusted Failure Rate (Calc. by Equation 4)	Observable Num. of Errors (≥ 1)?
1.05	1.33×10^{-5}	0.21×10^{-5}	No
0.95	1.62×10^{-4}	0.26×10^{-4}	No
0.85	1.77×10^{-3}	0.28×10^{-3}	No
0.75	1.70×10^{-2}	0.27×10^{-2}	No
0.65	0.14	0.02	No
0.55	1.06	0.17	No
0.49	2.79	0.45	Yes
0.45	6.72	1.07	Yes

NTV level was selected using Algorithm 1, given an application, the experimental hardware platform, and the power and fault models. We scaled to this selected voltage by altering the value of the high 16-bit of the core voltage MSR.

As stated, although power efficient, FiNTV on production processors incurs an exponentially growing number of failures. The calculated failure rates at different voltage levels using Equation 3 are theoretical and can be adjusted using Equation 4 by RVF. In Table IV, we obtain the adjusted failure rates using an example RVF value (0.159) of the sequential run of *bitcount* with the input listed in Table III, to demonstrate failure rate difference by adjustment. Note that in Equation 3, there is a reference failure rate λ_0 at V_h/f_{max}, which can be eliminated by dividing an unknown failure rate with a known one. We used the failure rates reported in [12] as known data as our baseline. Table IV lists the original and adjusted failure rates at voltage levels ranging from V_h to V_{th}. We can see that for voltage levels higher than V_l (including V_l), the rates is low so that no errors arise at runtime during our experimental HPC runs. For NTV levels that incur more than one error in an HPC run, e.g., 0.55 V, 0.49 V and 0.45 V, the adjusted failure rate at 0.55 V practically cause less than one error at runtime, while the adjusted failure rates at 0.49 V and 0.45 V could incur observable number of soft errors. Therefore, for the given RVF value 0.159, the optimal NTV level V_{ntv}^{opt} for power savings while incurring no observable number of soft errors, is experimentally 0.55 V. Likewise, we can obtain V_{ntv}^{opt} for other HPC runs using corresponding RVF values.

We implemented the register-level vulnerability estimation module `RegVul` as a plug-in functionality in the O3 (Out-of-Order) CPU model of gem5 (the tracking sub-module is embedded in existing code, while the calculating sub-module is alongside). When an instruction is being executed in gem5, the module captures register *reads* and *writes* in the simulated instruction pipeline stages (all register classes: *integer*, *floating-point*, and *miscellaneous* of *physical* and *architectural* registers are tracked), and accumulate register access intervals throughout all associated microarchitectural components, based on the register index and the CPU cycle (tick). After the instruction is committed, the module finishes calculating the vulnerability for registers involved in this instruction, based on the RVF definition in Figure 2 and Equation 1. When the simulated program execution is completed, the module collects all register vulnerability data and outputs an integrated RVF value for this execution, according to Equation 1.

gem5 is a simulation framework and system-wide power measurement is not supported in gem5 (while DVFS is enabled in the latest release). The most accurate solution to measure power consumption of the gem5 simulated environments is to integrate gem5 with power modeling enabled tools such as the McPAT framework [37]. We observed that using hardware counters (or physical power meters), the measured power costs on the gem5 host follow the trend of real power usage in the simulated system using gem5, with acceptable minor errors. In order to focus on our approach itself, we adopt the host power data reported by the RAPL hardware counters and leave the power modeling with gem5 as future work. Although RAPL only collects power data of processor and DRAM, we estimate the system-wide power costs using sampling and the nodal power model in Equation 5, i.e., the measurement/estimation-based method elaborated in Section IV.C and Algorithm 1.

For comparing the RVF difference between sequential and parallel versions of HPC applications, we parallelize the selected benchmarks using the industrial-standard parallelization libraries Pthreads and OpenMP, as shown in Table III. Note that Pthreads and OpenMP are not supported in the SE operation mode of gem5, we used *m5threads* in gem5 to convert the parallelized code to gem5-enable multithreaded semantics. We ran multithreaded versions of all benchmarks in 4 or 8 simulated cores using the SE mode of gem5. In particular, we parallelized the matrix multiplication program by MPI, using

the gem5 utility tool *dist* [38]. We ran the MPI version of *MatMul* in 10 processes using the FS mode of gem5.

C. Results

In this section, we present detailed experimental results for all benchmarks on power saving capabilities of our RSVP approach based on RVF values and adjusted failure rates, where RVF is in percentage and power data is system-wide power consumption and normalized. We elaborate the collected data by analyzing trends, root causes, and inferences.

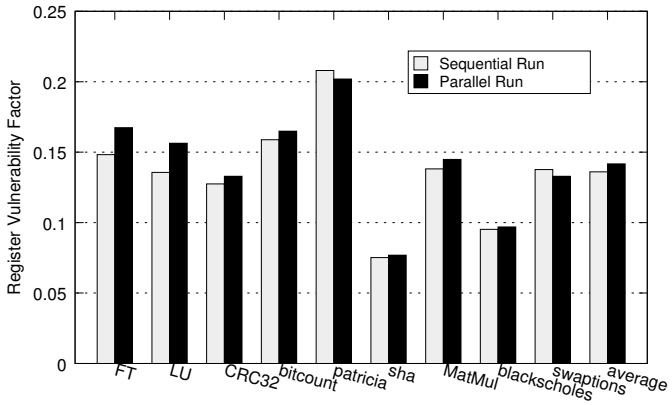

Fig. 4. Register Vulnerability Factors of Seq./Para. Runs for All Benchmarks.

1) Register Vulnerability Factors of Sequential and Parallel HPC Runs: Figure 4 depicts RVF values of both sequential and parallel runs of all benchmarks. We can see that the RVF values of parallel runs are in general higher than (except for *patricia* and *swaptions*) those of sequential runs (by 4.1% on average), since although the program semantics between the parallel version and the sequential version are basically preserved, overall the parallel version brings more data contention and thus more vulnerable register intervals due to accessing shared program resources. Moreover, compute-intensive applications (e.g., *FT*, *LU*, and *bitcount*) tend to have higher RVF values than other types of applications. On average, the RVF values of all benchmarks amount to around 14%, which means a large proportion ($\sim$86% on average) of register access intervals is non-vulnerable, and thus errors falling into such intervals are invalidated and cause no effects on the final program outputs.

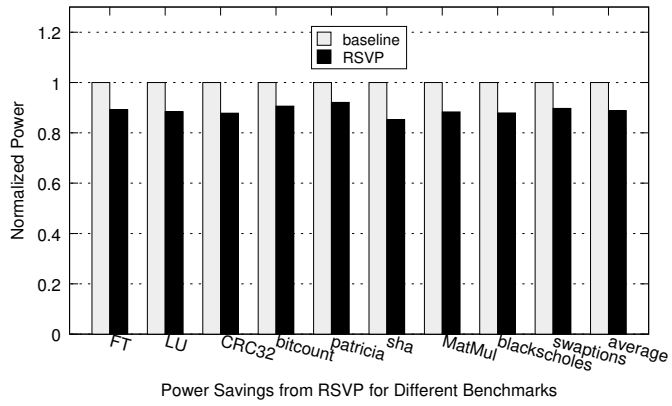

Fig. 5. Maximum Power Savings at V_{ntv}^{opt} for All Benchmarks.

2) Power Savings at the Optimal NTV: RSVP is evaluated to be effective in power savings by aggressively scaling down to the optimal NTV level, without incurring observable number of soft errors. As shown in Figure 5, we can see that on average, RSVP is capable of saving 11.2% system-wide power costs, compared to original HPC runs without RSVP. Note that all the power data presented here is for sequential HPC runs (power data for parallel HPC runs is similar and thus not shown). In summary, HPC runs with lower RVF values (e.g. *sha*, *blackscholes*, and *CRC32*) tend to achieve higher power savings, due to a higher percentage of non-vulnerable register intervals, and thus lower V_{ntv}^{opt}. The total power savings do not seem to be significantly high. This is because the reported power data is system-wide, while the voltage reduction technique employed in our approach, FiNTV, only affects processor power costs. The 11.2% nodal power savings can however make a huge difference if considering the massive amount of compute nodes in large-scale HPC clusters nowadays.

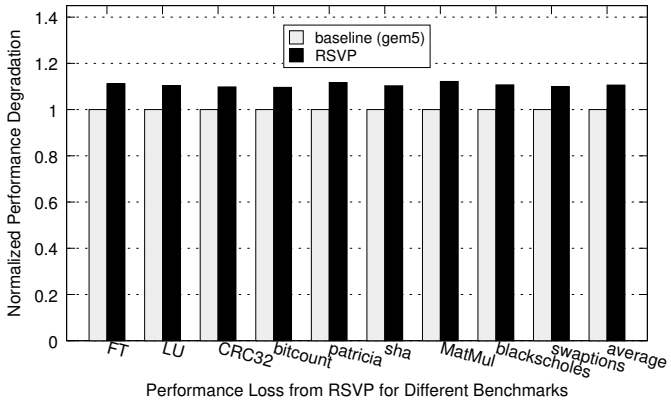

Fig. 6. Performance Loss from RSVP for All Benchmarks.

3) Performance Degradation: As demonstrated, our RSVP approach has been evaluated effective to save power, using RVF to adjust failure rates, for voltage reduction opportunities without incurring soft errors. The primary performance degradation of RSVP comes from tracking microarchitectural components for calculating RVF values. Figure 6 shows an average performance loss of 10.6% for all benchmarks in our experiments. We can see that for HPC applications with higher instruction count and more types of instructions (e.g., *patricia* and *MMM*), the performance degradation from RSVP tends to be higher, due to more trackable microarchitecture components relative to the total. Obviously, the fundamental simulation platform based on gem5 incurs significant performance overhead compared to the native execution, due to the heavyweight processor and system emulation. It is well studied that using hardware acceleration techniques such as KVM [39] can efficiently speed up the simulation/emulation, and thus potentially accelerate our RSVP approach as well. However, improving the performance of simulation in our approach is out of the scope of this work. We consider this part of efforts as potential future work for the usability of our framework.

VI. Conclusions

Upon the mutually constrained nature between system-wide resilience and power efficiency for current and projected supercomputers, trade-offs of them have been widely studied

in HPC community, with the goal of achieving the optimal performance-power ratio under power caps or resilience requirements. Although registers are susceptible to soft errors due to its nature of the most frequently accessed microarchitectural component, we observe that soft errors may not necessarily manifest themselves in register accesses because of the fact that errors occur at non-vulnerable register access intervals are overwritten/invalidated, and will not propagate to later program execution. In this work, theoretically, we quantify the validity of failure rates using the calculated RVF values of HPC runs. Practically, we propose and develop a power saving framework `RSVP` that leverages the adjusted failure rates to exclude invalid soft errors at register level, and identifies the optimal NTV level to maximize power savings without incurring observable amount of soft errors, based on fine-grained fault and power models. For a wide spectrum of HPC applications, using the popular simulation framework gem5, the proposed systematic approach is evaluated on a power-aware simulated platform to accurately obtain RVF for both sequential and parallel HPC runs, and save considerable system-wide power with no effects on program outputs.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

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

[5] C. Liu, J. Li, W. Huang, J. Rubio, E. Speight, and X. Lin, "Power-efficient time-sensitive mapping in heterogeneous systems," in *Proc. PACT*, 2012, pp. 23–32.

[6] J. Chen, L. Tan, P. Wu, D. Tao, H. Li, X. Liang, S. Li, R. Ge, L. N. Bhuyan, and Z. Chen, "GreenLA: Green linear algebra software for GPU-accelerated heterogeneous computing," in *Proc. SC*, 2016, p. 57.

[7] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power mamagement via dynamic voltage/frequency scaling," in *Proc. ICAC*, 2011, pp. 31–40.

[8] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "Memscale: Active low-power modes for main memory," in *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011, pp. 225–238.

[9] L. Tan, L. Chen, Z. Chen, Z. Zong, R. Ge, and D. Li, "HP-DAEMON: High performance distributed adaptive energy-efficient matrix-multiplication," in *Proc. ICCS*, 2014, pp. 599–613.

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

[11] 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.

[12] 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.

[13] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel, "Characterizing the effects of transient faults on a high-performance processor pipeline," in *Proc. DSN*, 2004, p. 61.

[14] R. E. Kessler, "The Alpha 21264 microprocessor," *IEEE Micro*, pp. 24–36, Mar. 1999.

[15] *Intel® Itanium® Processor*. http://ark.intel.com/products/family/451/Intel-Itanium-Processor, 2016.

[16] R. A. Ashraf, R. Gioiosa, G. Kestor, R. F. DeMara, C.-Y. Cher, and P. Bose, "Understanding the propagation of transient errors in HPC applications," in *Proc. SC*, 2015, p. 72.

[17] L. Tan, Z. Chen, and S. L. Song, "Scalable energy efficiency with resilience for high performance computing systems: A quantitative methodology," *ACM Trans. Architecture and Code Optimization*, vol. 12, no. 4, p. 35, Jan. 2016.

[18] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, May 2011.

[19] V. Sridharan and D. Kaeli, "Eliminating microarchitectural dependency from architectural vulnerability," in *Proc. HPCA*, 2009, pp. 117–128.

[20] B. Fang, Q. Lu, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi, "ePVF: An enhanced program vulnerability factor methodology for cross-layer resilience analysis," in *Proc. DSN*, 2016, pp. 168–179.

[21] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proc. MICRO*, 2003, p. 29.

[22] J. Yan and W. Zhang, "Compiler-guided register reliability improvement against soft errors," in *Proc. EMSOFT*, 2005, pp. 203–209.

[23] J. Carretero, E. Herrero, M. Monchiero, T. Ramírez, and X. Vera, "Capturing vulnerability variations for register files," in *Proc. DATE*, 2013, pp. 1468–1473.

[24] J. Lee and A. Shrivastava, "Software-based register file vulnerability reduction for embedded processors," *ACM Trans. Embedded Computing Systems*, vol. 13, no. 1s, p. 38, Nov. 2013.

[25] ——, "Compiler-managed register file protection for energy-efficient soft error reduction," in *Proc. ASP-DAC*, 2009, pp. 618–623.

[26] J. Tan, S. L. Song, K. Yan, X. Fu, A. Marquez, and D. Kerbyson, "Combating the reliability challenge of GPU register file at low supply voltage," in *Proc. PACT*, 2016, pp. 3–15.

[27] M. Fazeli, A. Namazi, and S. G. Miremadi, "An energy efficient circuit level technique to protect register file from MBUs and SETs in embedded processors," in *Proc. DSN*, 2009.

[28] 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.

[29] 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.

[30] 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.

[31] C.-H. Hsu and W.-C. Feng, "A power-aware run-time system for high-performance computing," in *Proc. SC*, 2005, p. 1.

[32] *The Princeton Application Repository for Shared-Memory Computers (PARSEC)*. http://parsec.cs.princeton.edu/index.htm.

[33] *MiBench: A free, commercially representative embedded benchmark suite*. http://vhosts.eecs.umich.edu/mibench/.

[34] *NASA Advanced Supercomputing Parallel Benchmarks (NPB)*. http://www.nas.nasa.gov/publications/npb.html.

[35] *Intel® Xeon® Processor E5-2660 v3 Specifications*. http://ark.intel.com/products/81706/Intel-Xeon-Processor-E5-2660-v3-25M-Cache-2_60-GHz, 2014.

[36] *Intel® 64 and IA-32 Architectures Software Developer Manuals*. http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html, 2016.

[37] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. MICRO*, 2009, pp. 469–480.

[38] M. Alian, D. Kim, and N. S. Kim, "pd-gem5: Simulation infrastructure for parallel/distributed computer systems," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 41–44, Jan. 2016.

[39] *KVM (Kernel-based Virtual Machine)*. http://www.linux-kvm.org/.