Energy Analysis and Optimization for Resilient Scalable Linear Systems

Zheng Miao, Jon Calhoun and Rong Ge
Clemson University
{*zmiao, jonccal, rge*}*@clemson.edu*

***Abstract*—Exascale computing must simultaneously address both energy efficiency and resilience as power limits impact scalability and faults are more common. Unfortunately, energy efficiency and resilience have been traditionally studied in isolation and optimizing one typically detrimentally impacts the other. To deliver the promised performance within the given power budget, exascale computing mandates a deep understanding of the interplay among energy efficiency, resilience, and scalability.**

In this work, we propose novel methods to analyze and optimize costs of resilience techniques including checkpoint-restart and forward recovery for large sparse linear system solvers. In particular, we present experimental and analytical methods to analyze and quantify the time and energy costs of recovery schemes on computer clusters. We further develop and prototype performance optimization and power management strategies to improve energy efficiency. Experimental results show that recovery schemes incur different time and energy overheads and optimization techniques significantly reduce such overheads. This work suggests that resilience techniques should be adaptively adjusted to a given fault rate, system size, and power budget.

***Index Terms*—Resilience; Energy-Efficiency; Forward-Recovery; HPC**

1. Introduction

Power and resilience are two major yet intertwined challenges in exascale computing. Today's top Petascale computers consume over 10 megawatts (MW) of power and have a mean time between failure (MTBF) in 1-7 days [19]. Future exascale systems are expected to have a similar power budget of 20 MW [5]. Their MTBF would be within an hour, due to massive concurrency and unreliability induced by miniaturizing feature size and ultra low power technologies [38]. Resilience techniques allow programs to continue progressing in the presence of failures through redundancy and recomputing, and thus are indispensable in exascale computing design and operation. However, resilience incurs power and time overhead and exacerbates the power challenge. Thus, exascale computing mandates simultaneously addressing scalability, resilience, and energy efficiency.

Previous studies have mainly focused on improving either resilience or energy efficiency, usually at the expense of one another. Resilience technologies aim to reduce time-to-solution (TTS) for programs in case of failures without considering energy requirements. For example, checkpoint-restart (CR) investigates and balances checkpointing time and rollback distance [18], and triple modular redundancy consumes $3\times$ the power to provide error detection and correction. Algorithm based fault tolerance [25, 13] exploits partial redundancy, and forward recovery [27, 2] explores approximations of lost or corrupted data to recover from faults. Meanwhile, power management technologies — e.g., near-threshold voltage — generally increases the cost of resilience by making computing software and device more complex and unreliable [4].

Recent work investigates the energy cost of resilience, but is limited to checkpointing and message logging [31] and the energy impact of checkpointing frequency [6] and Dynamic Voltage and Frequency Scaling (DVFS) [32]. More comprehensive studies are needed to answer multiple prominent *research questions*: (1) what is the resilience ability of various recovery mechanisms? (2) what is the power requirement of resilience and how can power management help? (3) what are the time and energy costs of resilience? (4) which recovery mechanism is most energy efficient for a given workload? (5) how does the resilience cost scale with system size and decreasing MTBF?

Answering these questions requires deep understanding of performance, energy efficiency, resilience, and their interplay in faulty environments. Different resilience techniques incur different amounts of time and energy. Moreover, a resilience technique responds differently to algorithms, workload characteristics, failure rate, hardware, and power allocation. For example, forward recovery for iterative Krylov solvers approximates lost or corrupted data in multiple ways [2]. Typically, better approximations take longer time and more energy to be constructed but allow faster progress to solution than poor approximations. Quantifying the time and energy costs accurately is necessary to evaluate the time-energy tradeoffs and identify the optimal resilience technique for a given situation.

Analytical models built from fine-grain measurement data are a promising approach to project time, energy, and resilience on large-scale systems for multiple reasons. First, generalized analytical models capture the first-order cost factors for various resilience techniques. Second, analytical models can be customized to reflect unique features of specific techniques. Third, fine-grained measurement of performance, power and resilience at a thread level and of

computer components can accurately capture model parameters. Fourth, fine-grain models can be used to predict the effect of power management at the system and component levels.

In this work, we present a set of analytical models that describe the energy efficiency, resilience, and performance of scientific applications under faults. We examine the impact of various fault recovery schemes for iterative linear solvers and further propose techniques to minimize time and energy overhead. Based on model parameters we derive from experiments on a cluster, we parameterize a model and use weak scaling to project program behavior for large-scale systems.

We make the following main contributions in this work:

- To the best of our knowledge, this work is the first of its kind that co-studies performance, scalability, resilience and energy efficiency of large scale scientific computing in a faulty environment.
- The proposed analytical models capture the first order time and energy cost factors for various fault recovery schemes and are customized to fit specific ones. They are used to identify the best recovery schemes for given fault situations.
- The optimization techniques reduce the time and energy overhead of recovery schemes by 16% for parallel iterative algorithms.
- The analysis reveals the interplay between power, performance, and resilience on large-scale systems.

2. Motivation

To motivate the need for co-analyzing scalability, resilience, and energy, we demonstrate that emerging HPC systems have frequent failures and require resilient computing to warrant application progress. We further discuss the associated time and energy costs, show the power behaviors of resilience techniques, and explore opportunities for power management.

2.1. Faults on Emerging HPC Systems

Faults are caused by incorrect states of software or hardware. They are classified into hard and soft faults based on their impacts. Soft faults, such as bit-flips and silent errors, cause an erroneous deviation in applications but without an interruption. Hard faults, such as processor failures and node failures, cause an application or system to crash [7].

In this work, we focus on both hard and soft faults in hardware and assume that the software environment is faultless[1]. Soft faults are commonly grouped into three categories [38]: Detected and Corrected Error (DCE), Detected but Uncorrected Error (DUE), and Silent Data Corruption (SDC). Hard faults have more categories. Here we select three common and frequent hard faults at system level [19]: System-Wide Outage (SWO), Single Node Failure (SNF), Link and Node Failure (LNF).

Figure 1 indicates that the MTBF of an exascale system is within an hour if projected from Petascale systems [19]. Here we assume a petascale machine consists of 20K compute nodes built with today's technology and an exascale machine consists of 1M compute nodes with 11 nm technology [5, 38]. We use the same method as in [19, 38] to estimate MTBF of various fault classifications on a single node or the whole system. We conservatively assume that MTBF is only affected by system size and node-level technology. The actual situation might be worse [11, 38].

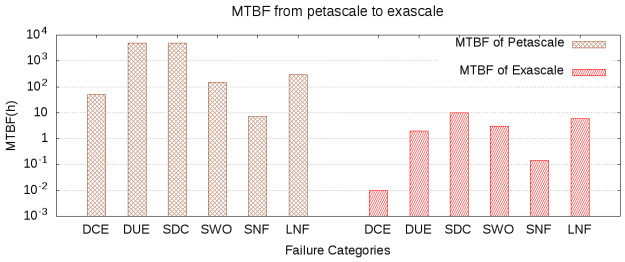


Figure 1. Estimated MTBF for exascale systems from petascale systems

2.2. Resilient Computing and Its Impacts

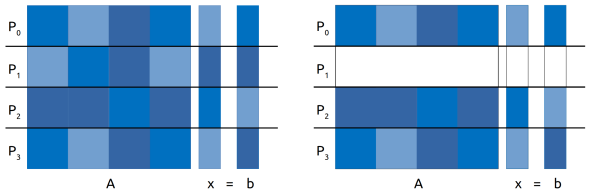
Resilient computing is indispensable at exascale where MTBF is small — i.e., within hours or minutes. Without resilience, most applications will make little forward progress in computation or return erroneous results. Resilient computing requires fault detection and recovery. In this work, we focus on fault recovery and assume that faults are detected and confined to a subset of data structures [10]. There are three main recovery approaches for soft and hard faults: Double Modular Redundancy (RD), Checkpoint-restart (CR), and Forward-recovery (FW).

All recovery mechanisms incur time and energy overhead, but differ in the amounts. To illustrate the difference of their overheads, we use the Conjugate Gradient (CG) method as an example. CG iteratively solves linear equations in the form of $Ax = b$, where the $n \times n$ matrix A is symmetric positive-definite, and column vectors x and b have n entries. CG terminates when a fixed tolerance or maximal number of iterations is achieved. When a fault occurs in process p_i, data in its memory is erroneous or lost, as shown in Figure 2. While the static data A and b can be restored from persistent storage, the dynamic data x needs to be recovered via RD, CR, or FW.

Figure 3 shows the time and energy overhead for various recovery mechanisms for CG to reach the same accuracy. In this experiment, the matrix A is $Andrews$ from [1], MTBF is set to 0.1 hours and CR checkpoints x to disks. The experiments are conducted on a 192-core cluster. We observe that:

- Each recovery mechanism incurs significant time and/or energy overhead (at most 2×).

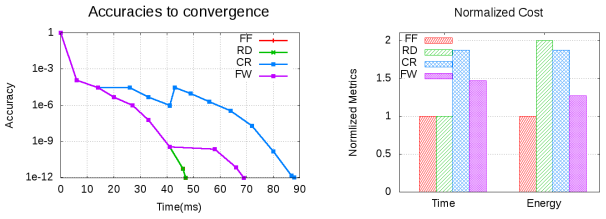
1. Although we do not consider faults in the software environment, the software environment is still able to propagate errors generated by hardware faults.



(a) Fault free (b) Fault occurs in p_1

Figure 2. CG with a block-row partition. (a) matrix A, vectors x and b are partitioned to four processes in a fault-free (FF) environment. (b) When a fault occurs in process p_1, part of x is erroneous or lost.

- FW consumes the least energy among the recovery mechanisms — i.e., 30% in comparison to 68% and 63% by CR and RD respectively.
- RD does not incur a time overhead but doubles the energy consumption in comparison to the fault-free (FF) case.



(a) Accuracy (b) Time and Energy

Figure 3. Accuracy and cost of different recovery mechanisms.

Resilience also changes the power profile of applications. In a fault-free environment, CG consumes a constant amount of power over its execution. In a faulty environment, its power consumption doubles if RD is employed. However, if CR is employed, the power consumption alternates between the high and low plateaus. The high plateaus correspond to the normal CG execution phases, and the low plateaus correspond to phases where checkpoints are written to disk. Similarly, power consumption drops when FW is used to generate an approximation of the lost intermediate results. Such details are presented in the results section.

2.3. The Performance-Energy-Resilience Interplay

The changes in the application's power profile should be taken into consideration, especially on systems that are imposed with power budgets. The additional power required to provide resilience reduces the power available for computation and thus impacts the application's performance and scalability. Meanwhile, the power drops during FW and CR under utilize the available power and reduce system throughput.

The above examples show that performance, resilience, and energy have complex trade-offs in a large-scale system. Balancing them in exascale computing requires understanding and co-analyzing an application's scalability, resilience, and power behaviors. Analytical models are a good approach to explore such scalable issues. Therefore, we create and analyze such models in this work.

3. Performance, Energy, and Resilience Co-Modeling

We focus on three performance metrics for a given workload w: time-to-solution T, power P, and energy-to-solution E. T had been the sole measure in parallel computing until power and energy began to constrain performance and scalability [30]. These metrics interact and their interplay depends on workload characteristics, performance optimization, and power and energy saving technologies.

Each of the metrics is altered by faults and the resilience techniques employed to tolerate faults. Faults, if occurring frequently, can have a dominating effect in large scale computing. In this work, we analytically model the impact of faults and evaluate the inherent time and energy costs of different resilience techniques.

We use CG as a case of study and examine workload properties commonly in parallel computing. Particularly, we focus on sparse banded matrices. We investigate weak scaling to project the performance and costs for large-scale systems. Specifically, we adopt the *fixed time scaling* approach [37], i.e., the execution time is constant for scaled workloads if parallel overhead is negligible. In our context, the number of non-zero entries and number of degrees-of-freedom per process remains constant.

3.1. Generalized Models

We first present general models to capture the time, power, and energy costs of all resilience techniques under study. The metrics and parameters are presented in Table 1.

TABLE 1. METRICS AND MODEL PARAMETERS.

	Symbol	Description
Metric	T	Time to solution
	P	Power consumption
	E	Energy to solution
Workload	w	Original workload
	w'	Scaled workload (fixed time)
Parameter	λ	Failure rate
	N	Number of cores

Time-to-solution for the original workload $T_1(w)$: the amount of time to complete workload w sequentially on a single core. We denote this time as T_{solve}:

$$T_1(w) = T_{solve} \tag{1}$$

Time-to-solution for the scaled workload $T_N(w')$: the amount of time to complete a scaled workload w' on a system with $N \geq 1$ CPU cores. Equation 2 includes the time to solve the scaled problem and parallel overhead in a fault free situation.

$$T_N(w') = T_{solve} + T_O(N) \tag{2}$$

Here T_O is the parallel overhead and is a function of N. Note the fixed time between the original and scaled workloads when the parallel overhead is not considered. The scaled workload w' has the same characteristics as the original workload w, but requires $N\times$ the computation. In the CG case, the size of the matrix A scales accordingly to keep a constant amount of work per process.

In faulty environments with a failure rate of λ, resilience incurs extra cost. We focus on recovery and assume fault detection is performed by other techniques [10] and the detection overhead is factored into the base running time for the solver. Therefore we extend Equation 2:

$$T_N(w') = T_{solve} + T_O(N) + T_{res}(w', N, \lambda) \quad (3)$$

where T_{res} is the total time overhead for resilience, including time to checkpoint, recompute lost progress, reconstruct an approximate state, and restart the external environments.

Power consumption for the original workload $P_1(w)$: the amount of power consumed by workload w during a sequential execution. Conceptually, the power consumption is summed over all computer components. For simplicity, we only account for the CPU core's power for two reasons: (1) cores are the dominant power consumer; (2) their power varies the most across resilience techniques.

Power consumption for the scaled workload $P_N(w')$: the amount of power consumed by N processor cores when executing the scaled workload w'. For the fixed time workload scaling, each processor core maintains the same computational intensity and thus power. Therefore,

$$P_N(w') = N \times P_1(w) \quad (4)$$

In a fault free situation, the application execution consists of useful problem progress periods and parallel overhead time such as communication and synchronization. Since we are more interested in the impacts of resilience, we assume that the power profile of CG is the same during progress phases and parallel overhead.

In faulty environments, the power profiles may alter between disjoint normal execution phases and recovery phases, and overlapped execution-recovery phases.

$$P_N(w') = \begin{cases} N \times P_1(w) & \text{execution phase} \\ P_{N,res} & \text{recovery phase} \\ N \times P_1(w) + P_{N,res} & \text{overlapped phase} \end{cases} \quad (5)$$

The power consumption during the recovery phase $P_{N,res}$ is discussed and quantified for each resilience technology in Section 3.2 and Section 5.

Energy-to-solution for the original workload $E_1(w)$: the total amount of energy to complete the workload w on a single core in a fault-free situation. It is the product of power and time — i.e.,

$$E_1(w) = P_1(w) \cdot T_1(w) \quad (6)$$

Energy-to-solution for the scaled workload $E_N(w')$: the total amount of energy to complete the scaled workload w' with N processor cores.

In a fault-free situation, it accounts for the energy to solve the problem and the parallel overhead.

$$E_N(w') = N \cdot P_1(w) \cdot \Big(T_{solve} + T_O(N)\Big) \quad (7)$$

In faulty environments with a failure rate of λ, additional energy is consumed to support resilience (see Section 3.2).

$$E_N(w') = P_N(w')_{avg} \cdot \Big(T_{solve} + T_O(N) + T_{res}(w', N, \lambda)\Big) \quad (8)$$

3.2. Specific Models for Recovery Schemes

We analyze the recovery cost of a hard or a soft fault, which causes data loss or corruption on a single process p_i. Recovery is needed for the computational environment, lost static data, and lost dynamic data [28]. We assume that the computational environment and the lost static data including $A_{:,p_i}$ and b_{p_i} are recovered immediately as in [2]. Let x^k be the solution vector when a fault occurs in the kth iteration of CG. Thus, the challenge is to recover the lost dynamic data — i.e., $x^k_{p_i}$ on the failed process p_i — see Figure 2(b).

We discuss several recovery schemes grouped into Checkpoint/Restart (CR), Redundancy (RD) and Forward Recovery (FW) as shown in Table 2. CR and FW include multiple variations. The general models are applicable to all these schemes. However, $T_{res}(w', N, \lambda)$ and $P_{N,res}$ are further refined for each resilience technique.

TABLE 2. RECOVERY SCHEMES UNDER STUDY

Type	Scheme	Description
CR	CR-D	Checkpoint to/rollback from disk
	CR-M	Checkpoint to/rollback from memory
RD	DMR	Double modular redundancy
FW	F0	Assign 0 to $x^k_{p_i}$
	FI	Assign initial guess to $x^k_{p_i}$
	LI	linearly interpolate lost $x^k_{p_i}$
	LSI	Interpolate lost $x^k_{p_i}$ with least squares

Checkpoint/Restart. The iterative solution vector x is checkpointed to storage periodically at certain iterations and recovered from the most recent correct checkpoint after a fault. Let x^{C_m} be the most recent checkpointing of x performed after the mth iteration when a fault occurs in the kth ($k \geq m$) iteration, the resilience cost $T_{res}(w', N, \lambda)$ with CR includes time to checkpoint the solution vector x and the time lost to compute from x^{C_m} to x^k.

$$T_{res}(w', N, \lambda) = T_{chkpt}(w', N, \lambda) + T_{lost}(w', N, \lambda) \quad (9)$$

where T_{chkpt} is the total time spent checkpointing, and T_{lost} is the total time spent re-computing to arrive at the state before the failure/error occurred.

T_{chkpt} is the product of per checkpointing cost t_C and the number of checkpoints taken. The latter is derived from the total execution time and checkpointing interval I_C, i.e.,

$$T_{chkpt} = t_C \cdot \frac{T_N(w')}{I_C} \tag{10}$$

t_C differs with the checkpoint storage — e.g. local-memory(cheap) or remote disk(expensive). The optimal checkpointing interval, I_C, is a function of failure rate and commonly approximated with Young's and Daly's approaches [41, 16].

T_{lost} is dependent on the failure rate and the average amount of recomputation time t_{lost}. The latter is approximated as a half of the checkpointing interval. For a failure rate λ, T_{lost} is derived as

$$T_{lost} = t_{lost} \cdot \lambda \cdot T_N(w') \approx \frac{I_C}{2} \cdot \lambda \cdot T_N(w') \tag{11}$$

In general, CPUs are not highly utilized during checkpointing and thus consume less power than in computation phase. That is, $P_{N,res} < N \cdot P_1(w)$. For cases when checkpointing takes a long time, transitioning the CPU's power to a lower power state saves power.

Redundancy. A dual-modular redundancy (DMR) resilience scheme requires $2N$ CPUs to support redundant computation. Assuming an unlimited number of CPUs without a power budget and two independent sets, the recovery time for x^k from the redundant replica after a fault is negligible. Nevertheless, the resilience phases are always concurrent with the normal program progress phases. Resilience causes additional power $P_{N,res}$ for the duration of the application by requiring double the power.

$$P_{N,res} = N \cdot P_1(w) \tag{12}$$

Forward Recovery. Forward recovery approximates lost data with simple assignments or reconstruction techniques. A more precise approximation of x^k takes more time/energy to construct but takes fewer extra iterations to converge to the final solution.

The time cost for FW resilience is modeled as:

$$T_{res}(w', N, \lambda) = T_{const} + T_{extra} \tag{13}$$

Where T_{const} captures the cost of reconstructing an approximation for x^k, and T_{extra} captures the cost of extra iterations required to converge. The former is the product of the reconstruction count and the cost per reconstruction t_{const}.

$$T_{const} = \lambda \cdot T_N(w') \cdot t_{const} \tag{14}$$

Constructing an approximation of the lost data may or may not involve all CPUs depending on the recovering algorithm. For example, $\widetilde{N} = 1$ during reconstruction for the FW methods under study. Given $\widetilde{N} \leq N$ processes actively constructing the approximation and $N - \widetilde{N}$ CPUs idle, the power during construction is less than that during normal execution.

$$\begin{cases} P_{N,const} = \widetilde{N} \cdot P_1(w) + (N - \widetilde{N}) \cdot P_{idle}, & \text{if constructing} \\ P_{extra} = N \cdot P_1(w), & \text{if extra iter.} \end{cases} \tag{15}$$

here P_{idle} is the power consumption when the core is idle.

The energy cost for resilience is the sum over the reconstruction and extra iterations, i.e.,

$$E_{N,res} = P_{N,const} \cdot T_{const} + N \cdot P_1(w) \cdot T_{extra} \tag{16}$$

We investigate four FW schemes: filling $x^k_{p_i}$ with all zeros (F0) and the initial guess (FI), linear interpolation (LI) [28] and least squares interpolation (LSI) [2]. These schemes have different reconstruction costs and accuracy. F0 and FI are assignment based and thus do not incur a construction cost — i.e., $T_{const} = 0$. However, they incur large T_{extra} to converge. On the contrary, LI and LSI are interpolation based and take time to construct more accurate approximations, but require fewer extra iterations to converge. The specific construction cost and extra iteration cost are determined by the workload and matrix properties.

Let $x^{LI}_{p_i}$ be the approximation of $x^k_{p_i}$ for the linear system solved by CG, LI constructs it with linear interpolation:

$$\begin{cases} x^{LI}_{p_i} = A^{-1}_{p_i,p_i}(b_{p_i} - \sum_{j \neq i} A_{p_i,p_j} x^k_{p_j}) & \text{for } j = i \\ x^{LI}_{p_j} = x^k_{p_j} & \text{for } j \neq i \end{cases} \tag{17}$$

LSI uses a more complex interpolation scheme and provides a more accurate approximation than LI. Let $x^{LSI}_{p_i}$ be the interpolation of $x^k_{p_i}$, LSI approximates it with:

$$\begin{cases} x^{LSI}_{p_i} = \min\limits_{x_{p_i}} \|b - \sum\limits_{j \neq i} A_{:,p_j} x^k_{p_j} - A_{:,p_i} x_{p_i}\| & \text{for } j = i \\ x^{LSI}_{p_j} = x^k_{p_j} & \text{for } j \neq i \end{cases} \tag{18}$$

The analytical modeling distinguishes between different resilience schemes. Corresponding model parameters are derived from experimental data in the following section.

4. Minimizing Recovery Cost

We present several strategies to reduce the overhead for the LI and LSI recovery schemes. Specifically, we introduce algorithms to efficiently construct the approximations and further use DVFS to reduce power during construction.

4.1. Localized Construction Algorithms

LI Optimization: $x^{LI}_{p_i}$ in Equation 17 requires solving a linear system. Let $y = b_{p_i} - \sum_{j \neq i} A_{p_i,p_j} x^k_{p_j}$. The failed process p_i uses linear interpolation to reconstruct $x^k_{p_i}$ via solving the following equation:

$$A_{p_i,p_i} x^{LI}_{p_i} = y \tag{19}$$

where all entries of A_{p_i,p_i} are static and are recovered from local storage on process p_i, and y is calculated using entries

of x from all the other processes. After a communication step, this problem is solved locally on process p_i.

Previous work [2] uses a sequential LU factorization of A_{p_i,p_i} to get the *exact* solution of $x_{p_i}^{LI}$. LU factorization requires a large amount of memory [24], and incurs high time and energy costs. A possible faster alternative is to parallelize LU factorization. However, parallelization increases communication time and can increase energy consumption by using all the cores.

We propose a more efficient approach to solve Equation 19. The key idea is to derive an approximation of $x_{p_i}^{LI}$ locally on process p_i. The exact solution is not necessary because itself is an approximate of the lost data $x_{p_i}^{k}$. Sequential execution eliminates communications and allows other processes to enter sleep sates for power savings.

LSI Optimization: $x_{p_i}^{LSI}$ in Equation 18 solves a least-squares linear system. Let $\beta = b - \sum_{j \neq i} A_{:,p_j} x_{p_j}^{k}$, it solves:

$$(A_{:,p_i}^T A_{:,p_i}) x_{p_i}^{LSI} = A_{:,p_i}^T \beta \tag{20}$$

here $A_{:,p_i}$ is a parallel matrix distributed among all processes. Previous work [2] uses a parallel sparse QR factorization of $A_{:,p_i}$ to get the exact solution of $x_{p_i}^{LSI}$. It involves a high volume of communication depending on the sparsity pattern of A.

We use CG to locally solve for $x_{p_i}^{LSI}$ on process p_i. We first transform the problem to enable local computation. Given the SPD matrix A, then $A_{:,p_i} = A_{p_i,:}^T$. Thus, we transform Equation 20 as follows:

$$(A_{p_i,:} A_{p_i,:}^T) x_{p_i}^{LSI} = A_{p_i,:} \beta \tag{21}$$

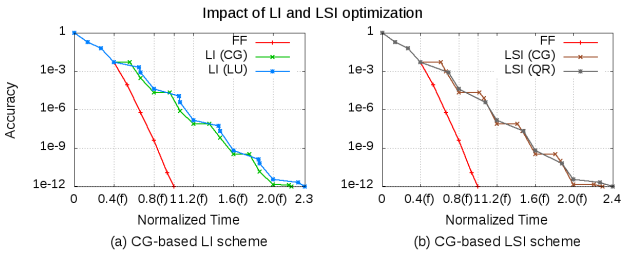


Figure 4. Time-to-solution with the CG-based construction algorithm for LI and LSI schemes on Matrix Kuu with 5 faults. LI/LSI (CG): CG-based LI/LSI forward recovery; LI (LU): LU-based LI forward recovery; LSI (QR): QR-based LSI forward recovery. A smaller y-axis value indicates a higher accuracy.

Figure 4 shows that using CG has a shorter time-to-solution than previous solutions for both LI and LSI. The improvement is 4-15%, depending on the tolerance. By computing a less accurate approximation, CG-based LI and LSI require less recovery time and total time than LU-based LI and QR-based LSI.

4.2. Power Reduction

Besides reducing the time-to-solution, using CG for the LI and LSI schemes provides power saving opportunities during the reconstruction phases. Since only p_i constructs the lost data of x_i, cores running other processes are able to transition to low speed states to reduce power consumption without impacting application performance.

In this work, we exploit DVFS commonly available on HPC CPUs for power reduction [29]. We bind processes to cores and adjust the core speed during the reconstruction phases for the LI and LSI schemes. Process-core binding is a common resource management technique and typically a one-to-one mapping is adopted for HPC applications. The core with process p_i always runs at the highest CPU frequency, while the other cores scale down to the lowest CPU frequency before reconstruction and scale up to the highest CPU frequency when reconstruction finishes.

Employting this power optimization techniques reduces power consumption during reconstructions by 40% with the power-aware LI scheme on a 24-core node (detailed results in Section 5.3). During reconstruction, 23 CPUs are idle, and the node consumes 0.75× of the power of normal execution phases without DVFS scheduling. When we apply DVFS scheduling, the node power drops to 0.45×. While not shown, the power-aware LSI scheme achieves similar power savings.

5. Results

This section evaluates the resilience and energy efficiency of different recovery schemes, and answers the research questions raised in the introduction section. We first present our experimental setup and benchmarks. We then evaluate the resilience of recovery mechanisms, and lastly assess their time and energy costs.

5.1. Experiment Setup

The experiment platform consists of 8 dual-socket nodes. Each node has two 12-core Xeon(R) E5-2670v3 processors and 128 GB DDR4 DRAM evenly distributed between the two NUMA sockets. DVFS is controlled using the CPUfreq interface. Each core can independently transition from 1.2 GHz to 2.3 GHz with a step of 0.1 GHz. Each core supports 2-way hyperthreading, which is only enabled for resilience evaluation and disabled for power and energy related experiments.

Data measurement is through benchmarks and utilities available on the system. Specifically, execution time is collected from benchmark reports, and processor power is collected with the Intel Running Average Power Limit (RAPL) interface.

We focus on symmetric positive definite (SPD) matrices with various sizes, densities, and convergence speeds, as shown in Table 3. These matrices are from the Suite Sparse Matrix Collection [1]. Each matrix is distributed among all parallel MPI processes in our experiment. CG and all resilience schemes are implemented from routines in RAPtor [9].

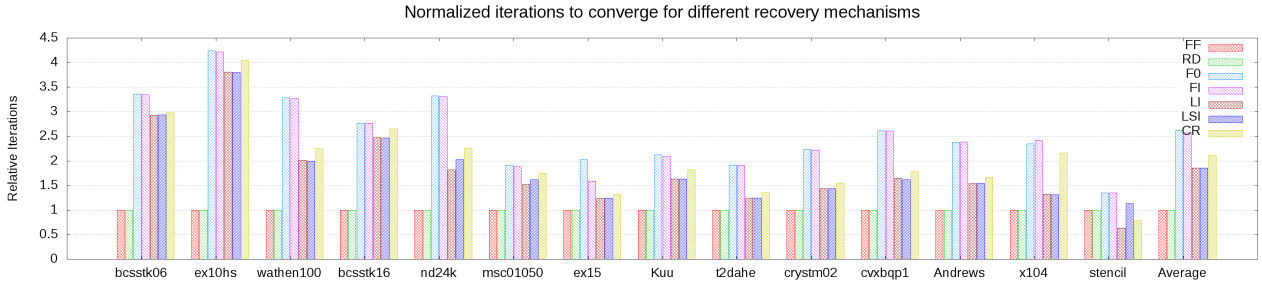


Figure 5. Iterations to convergence for different matrices using 256 processes with 10 faults. Each matrix uses its own normalization base, which is the fault free case.

TABLE 3. PROPERTIES FOR MATRICES TAKEN FROM SUITE SPARSE MATRIX COLLECTION.

Name	#Rows	#NNZ/row	Problem Kind	#Iters
bcsstk06	420	19	structural	4,476
msc01050	1,050	25	structural	35,765
ex10hs	2,548	22	CFD	3,217
bcsstk16	4,884	59	structural	553
ex15	6,867	17	CFD	1,074
Kuu	7,102	24	structural	849
t2dahe	11,445	15	model reduction	82,098
crystm02	13,965	23	materials	1,154
wathen100	30,401	16	random 2D/3D	355
cvxbqp1	50,000	7	optimization	11,863
Andrews	60,000	13	graphics	216
nd24k	72,000	399	2D/3D	10,019
x104	108,384	80	structure	96,704
5-point stencil	640,000	5	structure	3162

TABLE 4. NORMALIZED ITERATIONS TO CONVERGE UNDER VARIOUS PARALLEL SETTINGS FOR MATRIX CRYSTM02.

#p	FF	RD	F0	FI	LI	LSI	CR
4	1	1	2.17	2.16	1.44	1.44	1.55
16	1	1	2.16	2.15	1.44	1.44	1.56
64	1	1	2.23	2.23	1.44	1.44	1.55
256	1	1	2.23	2.22	1.44	1.44	1.55

5.2. Resilience of Recovery Mechanisms

What is the resilience of various recovery mechanisms? To answer this question, we investigate how resilient each recovery mechanism is, how it performs for different problems, and how it reacts to multiple faults.

The work evaluates recovery schemes for CG, but our results are applicable to other iterative solvers. CG iteratively refines an initial guess at each iteration. The algorithm terminates when the iterative solution is deemed accurate enough based on a small relative residual or when a fixed number of iterations is reached. In the presence of faults, the number of iterations to reach the same accuracy can increase. A recovery mechanism that takes fewer iterations to reach a desired accuracy is more resilient.

Note the resilience analyses in this subsection only accounts for iterations. Section 5.3 extends this discussion to cover both time, energy, and power.

In the following experiments, 10 faults are inserted evenly over the iterations required by the fault free execution (no more faults inserted after the fault free execution converges). The solver tolerance is set at $1e{-}12$. Since the number of iterations is the same regardless of where the checkpoint is stored, we do not delineate between memory and disk checkpointing in this subsection. Instead, we present results of disk checkpointing with a frequency of every 100 iterations.

Resilience vs. Parallelization. We first examine how parallel computing affects resilience for a given recovery mechanism. We use the crystm02 matrix as a case study, and solve a fixed-size problem with different numbers of MPI precesses on our cluster.

Table 4 shows the number of iterations to converge for each recovery mechanism that is normalized to the fault-free execution. Each mechanism has a constant number of iterations relative to process counts. We only present experimental results for a given problem and process count, because the choice of recovery scheme and process count does not change the fixed size problem.

We also observe that different schemes take different numbers of iterations to converge. RD performs the same as the fault-free execution. F0 and FI take the highest number of iterations to converge due to their inaccurate data reconstruction method. LI and LSI outperform F0 and FI by leveraging part of the intermediate computed results to create more accurate data reconstructions. CR rolls back to a previously saved state and takes more iterations than LI and LSI with the preset checkpointing frequency.

Mechanisms vs. Problems. We examine how the recovery mechanisms perform on different matrices. Figure 5 presents the number of iterations normalized to the fault-free performance. Overall, F0 and FI take the highest number of iterations (2.5× on average) to converge. RD takes the lowest number of iterations. LI, LSI, and CR perform similar to F0 and FI for matrices such as bcsstk06 and ex10hs, but perform much better for other matrices such as ex15 and t2dahe. This is due to the fact that LI and LSI construct less accurate solutions for the matrices with an irregular structure. CR requires more iterations than LI and LSI because it rolls back to a prior iteration state, and its overhead is due to the recomputation of the lost iterations. LI and LSI do not require as many iterations because of a more accurate reconstruction of x.

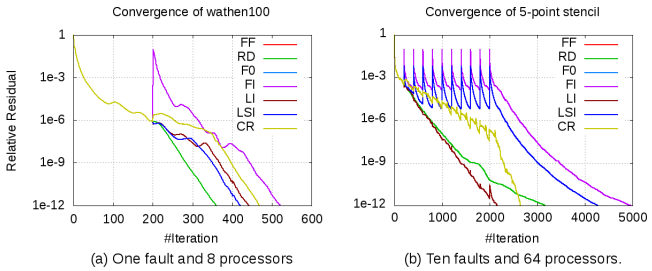


Figure 6. *residual∼#iteration* relation and correction under various recovery mechanisms. FF and RD are overlapped. F0 and FI are overlapped.

Number of Iterations to Converging and Correction. Recovery mechanisms takes a number of extra iterations due to faults/failures. Figure 6 shows the variation in the residual history when solving two different linear systems with various number of faults and recovery schemes.

With a single fault injected at the 200th iteration, Figure 6(a), the residual increases for all recovery schemes except for RD, which overlaps with the FF case. This is due to the fact that RD recovery the exact solution. Different recovery schemes result in a different change in the residual. F0 and FI (overlapped) has the largest increase, while LI and LSI (overlapped) get a minimal increase by constructing a more accurate approximation. Note that CR has a noticeable increase by rolling back to a previously checkpointed result. Figure 6(b) shows an example with 10 faults for a 5-point stencil matrix. LI and CR take fewer iterations to converge. In CG, reconstructing x forces reconstruction of other renew other variables in each iteration, including CR. In this example, their constructed solution makes the path to converge shorter.

5.3. Time, Power, and Energy Costs of Resilience

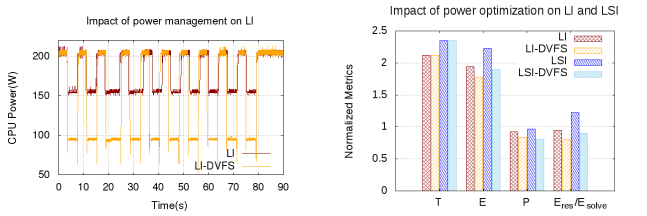
The previous analysis only captures extra iterations required by resilience. Iterations do not tell the entire time cost. In this section, we analyze the time, power, and energy costs of resilience, and begin with power consumption. From this subsection, MTBF is set as the same of that in Section 5.2. The checkpointing frequency of CR is computed via Young's formula [41].

What is the power requirement of resilience and how does power management help? Here we focus on the LI and LSI mechanisms and how they benefit from power management. We limit our discussion on power management for checkpointing as it has been previously investigated [31].

Figure 7(a) illustrates how DVFS-based power management changes the power profiles of matrix $nd24k$ on a single node with the LI scheme. We compare our optimization denoted LI-DVFS with the OS-level power management. The OS-level management uses the "ondemand" governor and scales up CPU speed if the CPU utilization is high or scales the frequency down if low. LI-DVFS uses the "userspace" governor. It runs all CPUs at 2.3 GHz in computation phase, and runs all but one CPU at 1.2 GHz during the construction phase. The one CPU that actively reconstructs an estimation of lost data runs at 2.3 GHz. LI-DVFS reduces power by 39% during the construction phase without performance degradation. While not shown, LSI-DVFS achieves similar power reduction.

TABLE 5. TIME AND ENERGY COST OF RESILIENCE OF VARIOUS MECHANISMS. FF IS THE NORMALIZATION BASE.

	Time	Power	Energy
FF	1	1	1
RD	1	2	2
LI-DVFS	2.12	0.84	1.78
LSI-DVFS	2.35	0.81	1.90
CR-M	1.83	0.98	1.79
CR-D	2.42	0.93	2.25



(a) Power reduction for nd24k (b) Energy savings for 14 matrices

Figure 7. Power reduction and energy savings with LI-DVFS and LSI-DVFS. (a):Power profile of $nd24k$ with simple LI and LI-DVFS; (b) average time, power, and energy for 14 matrices included in Figure 5. T, E, and P are normalized based on the fault-free case. E_{res} / E_{solve} is the ratio of energy cost for resilience and for fault-free case.

Figure 7(b) presents the overall performance, power, and energy impact for the 14 matrices presented in Figure 5. LI-DVFS and LSI-DVFS maintain the same performance, and reduce energy by 11% and 16% respectively. With these optimizations, more energy is allocated to problem solving rather than resilience, as demonstrated by E_{res} / E_{solve}.

What are the time and energy costs of resilience? Since LI-DVFS and LSI-DVFS consume less power than LI and LSI, we only include the former in discussions henceforth. We implement both CR with memory (CR-M) and CR with disk (CR-D) to give a range of checkpointing/restart cost. We apply various recovery mechanisms to the benchmark matrices under study, and analyze the time, energy, and power cost of resilience.

Table 5 presents the normalized time, power, and energy costs of resilience for various schemes. The values are averaged over all the matrices under study, and the power consumption is averaged over the entire execution including problem solving, parallel overhead, and recovery. Overall, LI-DVFS incurs the least energy overhead, and CR-M incurs the least time overhead except for RD. In contrast, CR-D takes the most time and energy. RD always consumes the most power. We assume that the disk is shared between multiple users and consumes a constant amount of power regardless of configuration.

Which recovery mechanism is the most efficient for a given workload? The solution to this question lies in the workload properties and fault situation. Here, we keep the

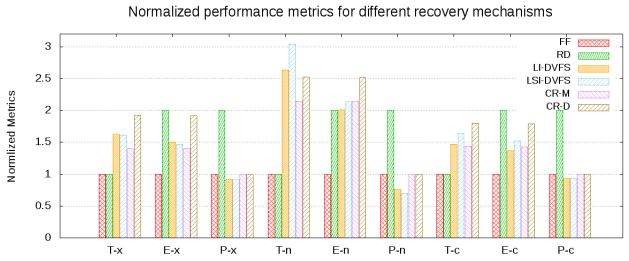


Figure 8. Normalized time, energy and average CPU power for three matrices under various recovery mechanisms. x—matrix $x104$; n—matrix $nd24k$; and c—matrix $cvxbpq1$. MTBF is 0.1 hour and MPI count is 24.

TABLE 6. VALIDATION OF MODEL FOR MATRIX X104 NORMALIZED TO FAULT FREE (FF)

Scheme	Model			Experiment		
	T_{res}	P	E_{res}	T_{res}	P	E_{res}
FF	0	1	0	0	1	0
RD	0	2	1	0	2	1
LI-DVFS	0.79	0.92	0.73	0.61	0.86	0.52
LSI-DVFS	0.75	0.91	0.68	0.58	0.83	0.48
CR-M	0.42	0.98	0.41	0.38	1	0.38
CR-D	0.92	0.96	0.88	0.89	0.97	0.86

same fault situation and perform experiments on the various matrices.

Figure 8 show that among the schemes, the best option varies depending on what constraint to optimize: time, power, or energy. LI and LSI need the least amount of power and energy if reconstruction time is relatively short, and CR-M often consumes the least amount of energy. The energy-efficiency of these resilience schemes is also comprehensively determined by several matrix properties. For matrices like $x104$ with an irregular pattern, CR-M is most efficient while FW costs more time and energy in reconstruction. For matrices like $nd24k$ with more non-zeros per row, RD costs the least time and energy, and both CR-M and FW cost more due to less accurate solution reconstruction. For matrices like $cvxbqp1$, FW is most efficient by reconstructing a more accurate solution and requiring the fewest extra iterations.

We evaluate the accuracy of our models by comparing the model's estimated costs and measured cost for multiple experiments. Table 6 shows the resilience overhead for matrix $x104$ normalized to FF. FF and RD uses the same data in the models and in the experiments. The resilience overhead of other recovery schemes is computed via corresponding models. For LI-DVFS and LSI-DVFS, the unit time for reconstruction t_{const} is measured. For CR-M and CR-D, unit time for checkpointing t_C is measured. For LI-DVFS and LSI-DVFS, our models over estimate T_{res} and E_{res} as higher extra time due to reconstruction is predicted. As our main goal is to provide comparison and relative order between the schemes under study, such estimation is acceptable. We will refine the model's parameterizations in the future to improve the model's accuracy.

6. Cost Projection for Large Systems

Section 5 shows how recovery schemes perform in a 8-node cluster. This section uses our models and experimental data to project how the schemes work on large systems.

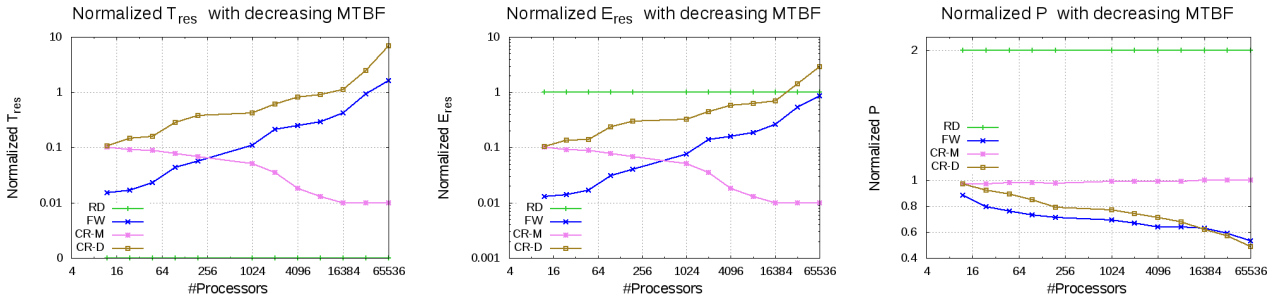
How does the resilience cost scale with system size and a decreasing MTBF? To answer this question, we need to project T_{res}, E_{res}, and P from our experimental platform to a very large system.

We compute overhead for a scaled workload via the models from Section 3. First, we project T for a fault-free baseline, where $T = T_{solve} + T_O$. In our measured data, parallel overhead T_O roughly equals the communication overhead. In each CG iteration, communication incurs to transfer data for sparse matrix-vector multiplications (SpMV) and vector-inner products. We use the average communication time cost of a SpMV from experimental data from a large system [8], where the SpMV's weak scaling performance is studied for matrices with 50K nnz per processor ranging from 1K to 60K processes. The time cost of a vector-inner product is linear with system size [40]. We project T_O with the average communication time cost of SpMV and vector-inner product.

We project resilience overheads for RD, CR-D, CR-M, and the best case of FW from our experimental data on a large system. We project T_{res} to the large system based on our models in Section 3. t_C of CR-D increases linearly as system size increases in our experimental data. We assume it continues to increase linearly in the large system. t_C of CR-M is stable, we assume this continues in the large system. t_{const} of FW increases linearly as system size increases in our experimental data. We assume that this trend continues in large systems. For t_{extra} of FW, we adopt an average normalized overhead based on the fault-free case. We adopt this average data to project FW in the large system. We project P based on models, where $P_{idle} = 0.45 \cdot P_1(w)$ for FW and $P_{idle} = 0.4 \cdot P_1(w)$ for CR-D. E_{res} is then calculated by all terms in T_{res} and P. The constants are derived from results in Section 5.

Figure 9 presents the projected resilience overheads for different resilience schemes. We scale the matrices to maintain 50K nnz per process, and assume a constant per-processor MTBF of 6K hours and a linearly decreasing system MTBF. T_{res}, E_{res} and P are all normalized to the fault-free case for each system size. Resilience schemes have various patterns. T_{res} and E_{res} show similar trends. As system size increases and MTBF decreases, T_{res} and E_{res} of RD keeps the same as the fault-free case. T_{res} and E_{res} of FW increases roughly linearly because t_{const} is linear and t_{lost} per fault is fixed. T_{res} and E_{res} of CR-D increases faster because of t_C and more frequent checkpointing. T_{res} and E_{res} of CR-M decreases because of its negligible t_C. P of FW and CR-D drops as the time cost in recovery or reconstruction becomes dominant.

As system size increases and MTBF decreases, T_{res} and E_{res} for FW and CR-D become larger than time and energy required for the fault-free case, and dominate the total time-to-solution and energy-to-solution. Moreover, if



(a) Normalized resilience overhead for time (b) Normalized resilience overhead for energy (c) Normalized average power

Figure 9. Normalized resilience overhead of weak scaling for 50,000 nnz per process with a decreasing MTBF.

MTBF continues to decrease, workload progress can possibly halt. while CR-M performs best in the projection, it is not practical to common fault situations with lost data in memory.

Current resilience schemes do not meet the requirements of future larger and more faulty systems. We need more optimizations to further reduce overhead of these resilience schemes, especially the time and energy costs in checkpointing, recovery or reconstruction phases. Decreasing them can greatly improve the full application's time and energy.

7. Related Work

In large-scale systems, fault tolerance mechanisms involves checkpointing /restart (CR) from a parallel file system [38]. In related research for CR, Berkeley Lab Checkpoint Restart (BLCR) [23] are system-level CR with disks. Scalable CR (SCR) [33] uses multi-level CR and CR at the system software layer. Disadvantages of classical CR strategy exist as unacceptable time to checkpoint, and the global restart even if only one process fails. Modular redundancy has been developed significantly in recent years, which focus on thread-level [22], process-level [36], and MPI implementations [21]. Both Triple-Modular Redundancy (TMR) and Dual-Modular Redundancy (DMR) induces significant overhead. In order to deal with these challenges in CR and RD, Algorithm-Based Fault Tolerance (ABFT) is an area of active research.

ABFT techniques detect and recover from errors in linear algebra operations by the use of checksums [25]. Significant research has been proposed for different ABFT schemes to address soft faults [12, 17] or hard faults [3, 13, 27] or both [39, 14, 15]. In latest research of ABFT, Huber et al. [26] combines domain partitioning with geometric multigrid methods to obtain resilient solvers based on the redundant storage of ghost values. Scholl et al. [34] proposes a fault tolerance approach to implicitly provide error locations and to enable partial recomputations for erroneous outputs after error detection. These ABFT algorithms are often combined with a rollback-recovery mechanism, which brings an overhead for a fault-free situation. In this paper, we evaluate forward-recovery schemes in [2] to avoid an overhead when no fault occurs.

Energy-efficiency of resilience mechanisms has been one of major challenges in recent years. Diouri et al. [20] first present a study that evaluates checkpointing and other existing fault tolerance protocols from energy consideration. They conclude that in clusters, the difference of energy consumption mainly depends on the execution time because operations have close power consumption in clusters. Meneses et al.'s work [31] presents a way to understand how fault tolerance and energy consumption interplay for three recovery schemes. Their paper shows that parallel recovery consumes less energy because it reduces the restart time. Aupy et al. [6] explores energy optimization in the checkpointing period. Scholl et al. [35] adapts the underlying precision in Preconditioned Conjugate Gradient (PCG) solvers on approximate computing hardware to gain energy efficiency. Instead of only looking into execution time, in this paper, we investigate energy optimization of reducing the communication overhead during recover/restart phase.

8. Conclusion

This paper proposes a novel approach to analyze and optimize the cost of resilience techniques for iterative linear solvers. We present a set of models to better understand resilience and energy overhead of applications in a faulty environment, and we perform power optimizations to reduce the overhead of forward recovery. Our experiments show that our optimized forward-recovery algorithm significantly reduces the resilience overhead and provides insights for selecting recovery schemes for certain workloads. Our projection result reveals trends of resilience cost on large systems and provides direction for optimization of resilience schemes. In future work, we plan to extend our models to capture more resilience mechanisms and study the performance and energy optimization for more applications.

9. Acknowledgement

The authors like to thank Amanda Bienz for providing the large scale performance data for our models. This work is supported in part by the U.S. National Science Foundation under Grants CCF-1551511 and CNS-1551262.

References

[1] University of florida sparse matrix collection. https://sparse.tamu.edu/.

[2] E. Agullo, L. Giraud, A. Guermouche, J. Roman, and M. Zounon. Numerical recovery strategies for parallel resilient krylov linear solvers. *Numerical Linear Algebra with Applications*, 23(5):888–905, 2016.

[3] E. Agullo, L. Giraud, and M. Zounon. On the resilience of parallel sparse hybrid solvers. In *High Performance Computing (HiPC), 2015 IEEE 22nd International Conference on*, pages 75–84. IEEE, 2015.

[4] R. Aitken, E. H. Cannon, M. Pant, and M. B. Tahoori. Resiliency challenges in sub-10nm technologies. In *VLSI Test Symposium (VTS), 2015 IEEE 33rd*, pages 1–4. IEEE, 2015.

[5] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, et al. The opportunities and challenges of exascale computing. *Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee*, pages 1–77, 2010.

[6] G. Aupy, A. Benoit, T. Hérault, Y. Robert, and J. Dongarra. Optimal checkpointing period: Time vs. energy. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, pages 203–214. Springer, 2013.

[7] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, Jan. 2004.

[8] A. Bienz, W. D. Gropp, and L. N. Olson. Node aware sparse matrix-vector multiplication. *Urbana*, 51:61801, 2016.

[9] A. Bienz and L. N. Olson. RAPtor: parallel algebraic multigrid v0.1, 2017. Release 0.1.

[10] J. Calhoun, M. Snir, L. N. Olson, and W. D. Gropp. Towards a more complete understanding of sdc propagation. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '17, pages 131–142, New York, NY, USA, 2017. ACM.

[11] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir. Toward exascale resilience: 2014 update. *Supercomputing frontiers and innovations*, 1(1):5–28, 2014.

[12] Z. Chen. Online-abft: An online algorithm based fault tolerance scheme for soft error detection in iterative methods. In *ACM SIGPLAN Notices*, volume 48, pages 167–176. ACM, 2013.

[13] Z. Chen and J. Dongarra. Algorithm-based fault tolerance for fail-stop failures. *IEEE Transactions on Parallel and Distributed Systems*, 19(12):1628–1641, 2008.

[14] Z. Chen, G. E. Fagg, E. Gabriel, J. Langou, T. Angskun, G. Bosilca, and J. Dongarra. Fault tolerant high performance computing by a coding approach. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 213–223. ACM, 2005.

[15] T. Cui, J. Xu, and C.-S. Zhang. An error-resilient redundant subspace correction method. *Computing and Visualization in Science*, 18(2-3):65–77, 2017.

[16] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future generation computer systems*, 22(3):303–312, 2006.

[17] T. Davies and Z. Chen. Correcting soft errors online in lu factorization. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 167–178. ACM, 2013.

[18] S. Di, M. S. Bouguerra, L. Bautista-Gomez, and F. Cappello. Optimization of multi-level checkpoint model for large scale hpc applications. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 1181–1190. IEEE, 2014.

[19] C. Di Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer. Lessons learned from the analysis of system failures at petascale: The case of blue waters. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 610–621. IEEE, 2014.

[20] M. el Mehdi Diouri, O. Glück, L. Lefevre, and F. Cappello. Energy considerations in checkpointing and fault tolerance protocols. In *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*, pages 1–6. IEEE, 2012.

[21] C. Engelmann and S. Böhm. Redundant execution of hpc applications with mr-mpi. In *Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN)*, pages 15–17, 2011.

[22] A. Golander, S. Weiss, and R. Ronen. Ddmr: Dynamic and scalable dual modular redundancy with short validation intervals. *IEEE Computer Architecture Letters*, 7(2):65–68, 2008.

[23] P. H. Hargrove and J. C. Duell. Berkeley lab checkpoint/restart (blcr) for linux clusters. In *Journal of Physics: Conference Series*, volume 46, page 494. IOP Publishing, 2006.

[24] M. T. Heath. *Scientific computing*. McGraw-Hill New York, 2002.

[25] K.-H. Huang et al. Algorithm-based fault tolerance for matrix operations. *IEEE transactions on computers*, 100(6):518–528, 1984.

[26] M. Huber, B. Gmeiner, U. Rude, and B. Wohlmuth. Resilience for massively parallel multigrid solvers. *SIAM Journal on Scientific Computing*, 38(5):S217–S239, 2016.

[27] L. Jaulmes, M. Casas, M. Moretó, E. Ayguadé, J. Labarta, and M. Valero. Exploiting asynchrony from exact forward recovery for due in iterative solvers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 53. ACM, 2015.

[28] J. Langou, Z. Chen, G. Bosilca, and J. Dongarra. Recovery patterns for iterative methods in a parallel unstable environment. *SIAM Journal on Scientific Computing*, 30(1):102–116, 2007.

[29] J. Lee and N. S. Kim. Optimizing throughput of power-and thermal-constrained multicore processors using dvfs and per-core power-gating. In *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*, pages 47–50. IEEE, 2009.

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

[31] E. Meneses, O. Sarood, and L. V. Kalé. Energy profile of rollback-recovery strategies in high performance computing. *Parallel Computing*, 40(9):536–547, 2014.

[32] B. Mills, R. E. Grant, K. B. Ferreira, and R. Riesen. Evaluating energy savings for checkpoint/restart. In *Proceedings of the 1st International Workshop on Energy Efficient Supercomputing*, page 6. ACM, 2013.

[33] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *Proceedings of the 2010 ACM/IEEE international conference for high performance computing, networking, storage and analysis*, pages 1–11. IEEE Computer Society, 2010.

[34] A. Schöll, C. Braun, M. A. Kochte, and H.-J. Wunderlich. Efficient algorithm-based fault tolerance for sparse matrix operations. In *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on*, pages 251–262. IEEE, 2016.

[35] A. Schöll, C. Braun, and H.-J. Wunderlich. Energy-efficient and error-resilient iterative solvers for approximate computing. In *On-Line Testing and Robust System Design (IOLTS), 2017 IEEE 23rd International Symposium on*, pages 237–239. IEEE, 2017.

[36] A. Shye, J. Blomstedt, T. Moseley, V. J. Reddi, and D. A. Connors. Plr: A software approach to transient fault tolerance for multicore architectures. *IEEE Transactions on Dependable and Secure Computing*, 6(2):135–148, 2009.

[37] J. P. Singh, J. L. Hennessy, and A. Gupta. Scaling parallel programs for multiprocessors: Methodology and examples. *Computer*, 26(7):42–50, 1993.

[38] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, et al. Addressing failures in exascale computing. *The International Journal of High Performance Computing Applications*, 28(2):129–173, 2014.

[39] P. Wu, Q. Guan, N. DeBardeleben, S. Blanchard, D. Tao, X. Liang, J. Chen, and Z. Chen. Towards practical algorithm based fault tolerance in dense linear algebra. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, pages 31–42. ACM, 2016.

[40] Z. Xu and K. Hwang. Modeling communication overhead: Mpi and mpl performance on the ibm sp2. *IEEE Parallel & Distributed Technology: Systems & Applications*, 4(1):9–24, 1996.

[41] J. W. Young. A first order approximation to the optimum checkpoint interval. *Communications of the ACM*, 17(9):530–531, 1974.