ORNL REPORT

ORNL/TM-2013/283
Unlimited Release
Printed July 2015

Numerical analysis of fixed point algorithms in the presence of hardware faults

Miroslav K. Stoyanov, Clayton G. Webster

Prepared by
Department of Computational and Applied Mathematics
Computer Science and Mathematics Division
Oak Ridge National Laboratory
One Bethel Valley Road, Oak Ridge, Tennessee 37831

The Oak Ridge National Laboratory is operated by UT-Battelle, LLC,
for the United States Department of Energy under Contract DE-AC05-00OR22725.
Approved for public release; further dissemination unlimited.

ORNL/TM-2013/283
Date Published: July 2015

NUMERICAL ANALYSIS OF FIXED POINT ALGORITHMS IN THE PRESENCE OF HARDWARE FAULTS

M. Stoyanov [*] C. Webster [†]

Abstract.
The exponential growth of computational power of the extreme scale machines over the past few decades has led to a corresponding decrease in reliability and a sharp increase of the frequency of hardware faults. Our research focuses on the mathematical challenges presented by the silent hardware faults. i.e., faults that can perturb the result of computations in an inconspicuous way. Using the approach of selective reliability we present an analytic fault mode that can be used to study the resilience properties of a numerical algorithm. We apply our approach to the classical fixed point iteration and demonstrate that in the presence of hardware faults, the classical method fails to converge in expectation. We preset a modified resilient algorithm that detects and rejects faults resulting in error with large magnitude, while small faults are negated by the natural self-correcting properties of the algorithm. We show that our method is convergent (in first and second statistical moments) even in the presence of silent hardware faults.

Key words. algorithmic resilience, fixed point method, linear solvers, numerical linear algebra, fixed point methods

1. Introduction.

As modern science and engineering push the limits of our knowledge and understanding, there is an ever increasing demand for more complex mathematical models to describe various real world phenomena. Coupling between processes and scales as well as the need for better accuracy have created an insatiable demand for increased computing power. To this end, bigger and faster computers are proposed and built every year and every one of those machines is designed to challenge the limits of our configurations. Transistor's fabrication size shrinks exponentially and constraints on the power consumption force circuits to operate at near critical voltage, so that even a small power fluctuation may have an adverse effect on operation. In addition, the ever increasing number of components offers more and more opportunity for hardware failure. Unfortunately, the rapid increase of computational power has lead to a corresponding decrease in reliability, which has created an ever growing concern for the reliability of the computed results [5, 7, 14, 24–26].

In this work we focus on the problem of *silent* hardware faults that can perturb the result of computations in an inconspicuous way. We devise a theoretical fault model and a framework for analysis of error propagation, as well as assessment and enhancement of algorithm's resilience. In what follows, we describe the challenges associated with hardware failure and silent faults, that have led to the research described in this effort.

1.1. Hardware failure.

Hardware faults are a well known issue and measures are taken to ensure that every computer component operates reliably. For example, memory comes with error detection and correction [22, 26] and Cyclic-Redundancy-Checksums (CRC) are used to guard against data corruption in network traffic. However, processing cores remain largely unprotected [14, 17] and adding redundancy to all chips in a cluster is very expensive in terms of manufacturing cost and increased energy consumption. Faults in a single

[*]Department of Computational and Applied Mathematics, Oak Ridge National Laboratory, One Bethel Valley Road, P.O. Box 2008, MS-6367, Oak Ridge, TN 37831-6164 (`stoyanovmk@ornl.gov`).
[†]Department of Computational and Applied Mathematics, Oak Ridge National Laboratory, One Bethel Valley Road, P.O. Box 2008, MS-6164, Oak Ridge, TN 37831-6164 (`webstercg@ornl.gov`)

component are rare, however, in an extreme-scale machine, small chances of failure are multiplied so they cannot be ignored. Experiments performed in 2011 on Jaguar (the largest supercomputer at the time) showed a single node failure almost daily [14] and hundreds of (corrected) bit-flips every hour [25]. Hardware level error detection and correction are indispensable tools, however, the number of undetected and uncorrected faults is constantly increasing.

Software based resilience techniques are used to complement the hardware approach. One of the most common types of hardware fault is the fail-stop (sometimes called *hard* faults), where either the entire application, or some of the nodes of the cluster, crash and fail to return a result. This problem has been the focus of research in computational science over the past decades and the most common approach for addressing this issue is the checkpoint restart methods [2, 13]. Practically, all modern code is written so that it periodically saves the state of the software and in the event of a crash, computations can be resumed from the last saved checkpoint.

1.2. Silent faults. Hardware malfunction that doesn't result in a crash, may result in data corruption. If the fault affects a permanent data structure (e.g., an iteration matrix), then checksums can be used to detect the deviation, however, not all faults leave a "visible" permanent trace. For example, suppose two variables with values 2 (binary 010) and 4 (binary 100) need to be added. First they are moved from the DRAM into the SRAM (CPU cache). Then suppose the hardware suffers a voltage fluctuation causing the cache to fail and flip one bit in the second variable making it into a 6 (binary 110). When the add instruction is executed, it will result in the incorrect answer 8 (binary 1000), which will be stored back into DRAM and the corrupt data in the cache will be cleared. Effectively, the hardware computes erroneously that $2 + 4 = 8$. If the corrupted value contains invalid data (e.g., `inf` or `nan`) or if the values fall outside of physical boundaries (e.g., negative area), then some indication of a fault is still present. However, if the result is not "obviously" wrong, then we have the most sinister type of a fault that perturb the computations in way that is incorrect but yet believable. In our work, we call those the *silent* faults.

The main focus of this work is to propose and analyze novel numerical methods that converge even in the presence of silent faults, regardless of their source. We do not restrict our attention to bit-flips and our approach extends to any type of perturbation of the numerical computations that does not permanently alter the persistent data in main memory. The name *silent* is appropriate here, since it emphasizes the main challenge associated with detection of those faults.

Experimental studies of the effects of faults (silent or otherwise) have been performed (e.g., [6, 9]) and techniques for fault detection have been proposed (e.g., [4, 19]). Simple enhancements can improve the overall resilience of the methods, however, not all silent faults can be corrected or even detected, and experiments provide insight on the reliability but are restricted to the specific implementations and specific faults tested (e.g., bit-flips). In the context of linear algebra, Algorithm Based Fault Tolerance (ABFT) [1,16,18] is an advanced and very widely utilized technique. ABFT uses check-sums to detect data corruption in matrices and vectors as well as correct some silent faults in the basic algebraic operations, and ABFT has been successfully applied to dense [8] and sparse [23] matrix operations on large scale systems. However, the application of ABFT remains restricted to linear operations with explicit matrices, specific data-structures and implementation. Checksumming is a very powerful way to guard against data corruption in persistent structures, however, a more general higher-level approach is need to handle silent faults in the broader class of problems.

The work [3, 15] propose an alternative approach where the program execution is split into two modes, denoted *safe* and *fast* (respectively *host* and *guest* in [15]). The computations in *safe* mode are assumed to be performed reliably (i.e., zero or negligible probability of a fault), however, they are also more expensive. The safe mode has to be implemented with either redundant computation or via the use of specialized hardware, hence only the minimum amount of work should be performed in this mode. On the other hand, the *fast* mode is cheap, but prone to hardware failure. This approach is applied to a modified version of the restarted GMRES algorithm, where a convergence test is performed in safe mode between every two outer GMRES iterations [15]. The convergence test rejects the next iterate unless the residual error is reduced, thus large errors are rejected and only small ones are accepted by the algorithm. Similar approach has also been applied to Conjugate-Gradient method [21].

The approach of selective reliability is a viable technique for dealing with silent faults, however, it also presents a number of new challenges. For example, the convergence of the algorithm deteriorates as the frequency of the faults increases, due to the additional work needed to correct each fault. However, the rate of deterioration, and the relationship between the properties of the matrix (e.g., condition number), and the type of errors that can be resolved remains unknown. Thus, the frequency and type of faults that can be resisted is not fully characterized. Even if convergence is achieved, in the presence of random faults, the final result of the computations is a random variable with associated probability distribution, and the statistics of the distribution need to be quantified. As such, in what follows, a novel numerical analysis paradigm is presented in which the potential of selective reliability can be rigorously analyzed.

1.3. Analysis framework in the presence of faults. Our modern treatment of predicting the behavior of physical phenomena and complex engineering problems, relies on mathematical modeling followed by computer simulations. Traditional numerical algorithms have been developed with the assumption that the underlying algebraic operations, e.g., matrix vector products, can be carried out accurately. However, as the reliability of supercomputers decrease, scientists and engineers can no longer afford to make this assumption; especially when providing analysis and informing decision makers of the behavior of computer simulations. Numerical algorithms compute approximations to the solutions of mathematical models, and therefore, introduce an associated numerical error. Silent faults introduce additional hardware artifacts in the computer simulation that are often times indistinguishable from the numerical error, and therefore, a new analysis paradigm to simultaneously consider both numerical and hardware error must be established.

In this work, we establish a framework of numerical analysis that is free from the assumption that computations can always be performed accurately. Following the approach of selective reliability [3, 15], we assume that the result of numerical operations will be randomly perturbed. In order to provide analytic framework that is as general an agnostic to ever changing hardware architectures and implementation techniques, we make minimal assumptions on the structure of such perturbations. In the new analysis paradigm, we provide rigorous error bounds with respect to the hardware error, and we prove convergence of the algorithms, even if they are implemented on unreliable hardware (i.e., hardware susceptible to silent faults). As a demonstration of our approach on solutions of systems of equations, we analyze the widely used fixed point family of iterative solvers. We show how error of large magnitude can be rejected, so that faults can only introduce discrepancy that is of the order of the standard numerical error, thus preserving the theoretical convergence rates. This is the first step towards building a suite of resilient numerical algorithms that provide reliable results even in an unreliable computing environment.

The rest of the paper is organized as follows: in §2 we present the fault model and define convergence with respect to hardware induced error. In §3, we preset the analysis of the classical fixed point iteration and how it can be enhanced using the technique of selective reliability. In §4, we present a numerical comparison between the classical and enhanced fixed point methods.

2. Hardware error propagation and convergence. In this section we establish the framework for analyzing the effects of silent hardware faults on numerical algorithms. We propose a fault model that is agnostic to specific hardware architectures or software implementations. Using this model, we provide rigorous definition of *hardware error* and convergence in the presence of silent hardware malfunction.

2.1. Fault model. Hardware faults can be very diverse in nature and the resulting discrepancy introduced into the computations can be unpredictable. For example, consider some of the possible effects of a bit-flip, which is one of the most common types of hardware faults. A bit-flip in the opcode may convert one numerical operation into another (i.e., addition may turn into multiplication). A bit-flip in a floating point number can cause error in the entire range of values representable by the floating point standard, e.g., see our earlier work on the IEEE-754 double precision representation [11, 12]. A bit-flip in a pointer or array index can cause data to be read from a random location in memory and hence result in an error of unpredictable pattern. In complex operations, such as sparse matrix vector product, a fault in one number can propagate to multiple entries of the resulting vector depending on the sparsity pattern. Furthermore, some of the most efficient checkpoint/restart techniques for handling hard faults use partial/incomplete data recovery, that converts hard faults to soft faults and introduces additional error. Thus, the statistical properties of this new error strongly depend on the hardware circuits, floating point representation, data structures used, problem structure (e.g., matrix pattern) and even random memory content.

In order to keep our analysis as general as possible, we present a fault model that is agnostic with respect to all of the aforementioned specific conditions. First, consider that algorithms are comprised of a series of steps, where each step computes an intermediate result that leads to a final answer. Then, we assume that each algorithm is implemented with selective reliability, i.e., the steps are performed in either *safe* or *fast* modes, where the possibility of failure in safe mode is negligible, and every operation performed in fast mode is susceptible to hardware malfunction. The implementation of safe mode can be achieved via either redundancy or dedicated specialized hardware. Therefore, we assume that safe computations are significantly more expensive, and in this effort we require that the bulk of the work is performed in fast mode.

We model the hardware faults and the associated random perturbations as a two level probability model, i.e., a mixture model. We assume that the hardware faults are independent random events, and for every step performed in fast mode there is a positive probability to encounter a hardware malfunction that perturbs the intermediate result. Thus, the first level is modeled by a Bernoulli discrete distribution, where the two options are "faults were encountered at this step" and "no faults were encountered at this step". Once a fault is encountered, the magnitude and structure of the introduced discrepancy depends on too many specific factors, hence, we assume that the perturbation is a random variable with unknown probability density. We illustrate with an example below.

Example 2.1 (Adding two vectors). *Consider the following three step algorithm:*

1. *Fast: Compute some* $\boldsymbol{x} \in \mathbb{R}^N$
2. *Fast: Compute some* $\boldsymbol{y} \in \mathbb{R}^N$
3. *Safe: Return* $\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{y}$

The first two steps are performed in fast mode, while the last step is executed in safe mode. For steps 1 *and* 2*, let* $p_1, p_2 \in \mathbb{R}$ *so that* $0 < p_1, p_2 < 1$*, be the Bernoulli parameters that indicate the corresponding probability of encountering hardware faults. Each fault rate depends on the specific hardware configuration, code implementation, as well as the computational complexity of the associated step, i.e., operations that are computationally more expensive take longer to execute and are more likely to encounter faults. However, for a given algorithm and implementation on specific hardware, we assume that* p_1 *and* p_2 *are known, i.e., we assume that we can measure an approximate rate of faults of a given machine.*

If the algorithm encounters a fault on the first step, instead of returning $\boldsymbol{x}$ *we would compute* $\boldsymbol{x} + \tilde{\boldsymbol{x}}$*, where* $\tilde{\boldsymbol{x}} \in \mathbb{R}^N$ *is a random vector with unknown probability density. That is,* $\tilde{\boldsymbol{x}}$ *is associated with a complete probability space* $\left(\mathbb{R}^N, \mathcal{F}, \tilde{P}_1\right)$*, where* $\mathcal{F} = 2^{\mathbb{R}^N}$ *is the* σ*-algebra of events and* $\tilde{P}_1$ *is some unknown probability distribution. We do not assume that* $\tilde{\boldsymbol{x}}$ *has any specific structure or is bounded in any norm. Similarly, a hardware fault on the second step of the algorithm would perturb the result* $\boldsymbol{y}$ *to* $\boldsymbol{y} + \tilde{\boldsymbol{y}}$*, where* $\tilde{\boldsymbol{y}}$ *is associated with a complete probability space* $\left(\mathbb{R}^N, \mathcal{F}, \tilde{P}_2\right)$ *and unknown probability measure* $\tilde{P}_2$*. In other words, the distribution of* $\tilde{\boldsymbol{x}}$ *is a mixture between the distribution* $\tilde{P}_1$ *with weight* p_1 *and the Dirac-delta distribution* $\delta_0(\tilde{\boldsymbol{x}})$ *with weight* $(1 - p_1)$*.*

The last step of the algorithm is performed in safe mode and hence we assume that computations are done according to machine specification. However, the final answer $\boldsymbol{z}$ *would be affected by perturbations in* $\boldsymbol{x}$ *and* $\boldsymbol{y}$*, and thus the result could be perturbed by either* $\tilde{\boldsymbol{x}}$ *or* $\tilde{\boldsymbol{y}}$ *or both* $\tilde{\boldsymbol{x}} + \tilde{\boldsymbol{y}}$*. Even if all the steps of the algorithm are deterministic, the stochastic nature of hardware faults turns the final result into a random variable.*

2.2. Convergence with respect to hardware error. The perturbation associated with a hardware failure in fast mode is distinct from round-off error. Finite precision standards such as IEEE-754 introduce rounding error with each operation, however, this error is small and deterministic in nature. Algorithm conditioning with respect to round-off error is a well established subject in numerical analysis, however, conditioning cannot account for the random and potentially unbounded discrepancy introduced by hardware malfunction. Round-off error is introduced into the computations in compliance with standards and specifications, the hardware error that we consider is associated with abnormal machine behavior.

The error associated with numerical algorithms is usually measured in some norm. For all examples given in this paper, w.l.o.g., we assume that we are working with l^2 norm defined on $\mathbb{R}^N$. However, our definitions are general and norm agnostic, and many of our results trivially extend to other norms. Thus, we present the following definition:

Definition 1 (Hardware error). *Error introduced into numerical computations by hardware malfunction.*

In Example 2.1, the perturbation associated with the first step is $\tilde{\boldsymbol{x}}$ and, hence, the hardware error is $\|\tilde{\boldsymbol{x}}\|$. Similarly, the error associated with the second step is $\|\tilde{\boldsymbol{y}}\|$ and the final hardware error is $\|\tilde{\boldsymbol{x}} + \tilde{\boldsymbol{y}}\|$. The random vectors $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$ have unknown probability density, however, using our assumption about the probability of failure we have that $\mathbb{P}_B(\tilde{\boldsymbol{x}} \neq 0) = p_1$ and $\mathbb{P}_B(\tilde{\boldsymbol{y}} \neq 0) = p_2$, where $\mathbb{P}_B$ indicates the probability with respect to the Bernoulli measure. Therefore, the probability of non-zero final error is

$$\mathbb{P}_B(\tilde{\boldsymbol{x}} + \tilde{\boldsymbol{y}} \neq 0) = \mathbb{P}_B(\tilde{\boldsymbol{x}} \neq 0) + \mathbb{P}_B(\tilde{\boldsymbol{y}} \neq 0) - \mathbb{P}_B((\tilde{\boldsymbol{x}} \neq 0) \cap (\tilde{\boldsymbol{y}} \neq 0)) \leq p_1 + p_2 - p_1 p_2.$$

Note that we cannot make more specific conclusions about the distribution of $\|\tilde{\boldsymbol{x}} + \tilde{\boldsymbol{y}}\|$ since we made no assumptions about the measures $\tilde{P}_1$ and $\tilde{P}_2$.

In addition to hardware error, the accuracy of numerical algorithms is of critical importance. For example, a finite representation of an infinite dimensional operator introduces discretization error and the scheme is said to *converge* if the error goes to zero as the number of degrees of freedom increases. Another example is an iterative solver that generates a sequence of solutions. In this case, *convergence* is characterized by the sequence's ability to achieve a result as the number of iterations increases. Similarly round-off error can be reduced by using higher precision and representing real numbers with more bits, i.e., single, double, and quad precision.

In general, numerical convergence is related to the concept of additional computational works leading to an increasingly more accurate approximation. We want to extend this idea beyond numerical error and include hardware error, so that silent hardware faults are automatically corrected, accounting for additional computational complexity. The cost may be associated with extra iterations of an iterative solver or more refined mesh discretization, but in all cases more work should imply better accuracy. Furthermore, the hardware error is a random variable and hence, we consider accuracy in statistical sense, i.e., we want the statistical moments of the hardware error to tend to zero.

We formalize the idea of convergence in the presence of hardware faults by considering an algorithm with steps implemented in fast mode. We describe the hardware faults via the sequence of known Bernoulli parameters $\boldsymbol{p} = (p_1, p_2, \cdots, p_n, \cdots)$, and collection of unknown measures $\tilde{\boldsymbol{P}} = \left(\tilde{P}_1, \tilde{P}_2, \cdots, \tilde{P}_n, \cdots\right)$. Let e be the difference between the result of one realization of the algorithm and the desired exact solution, that is, e indicates the sum of numerical and hardware errors. Finally, denote by $\mathbb{E}_{\boldsymbol{p},\tilde{\boldsymbol{P}}}[e]$ and $\mathbb{V}_{\boldsymbol{p},\tilde{\boldsymbol{P}}}[e]$ the expectation and variance of e with respect to the mixture distributions defined by $\boldsymbol{p}$ and $\tilde{\boldsymbol{P}}$. The statistical moments of e strongly depend on $\tilde{\boldsymbol{P}}$, however, in order to keep our approach agnostic to specifics of the hardware, we take a conservative approach where we assume that every fault results in a the perturbation that would have the most detrimental effect on e. Following that notation, we define:

Definition 2 (Convergence with respect to hardware error). *A method is convergent with respect to the hardware error, if for every $\epsilon > 0$ there is a finite amount of work associated with the algorithm that will bound the statistics as*

$$\sup_{\tilde{\boldsymbol{P}}} \mathbb{E}_{\boldsymbol{p},\tilde{\boldsymbol{P}}}[e] < \epsilon, \qquad \textit{and} \qquad \sup_{\tilde{\boldsymbol{P}}} \mathbb{V}_{\boldsymbol{p},\tilde{\boldsymbol{P}}}[e] < \epsilon^2, \tag{2.1}$$

where the supremum is taken over all possible probability measures on $\mathbb{R}^N$.

Traditional numerical analysis assumes that the hardware is perfectly reliable, i.e., all steps of algorithms are performed in safe mode and, therefore, the hardware error is always zero, i.e., $p = 0$ at all times. However, since the hardware error is potentially unbounded (i.e., supremum over $\tilde{\boldsymbol{P}}$), even if a single step of an algorithm is performed in *fast* mode (i.e., $p > 0$), classical algorithms would fail to converge. For example, the simple approach described in Example 2.1 does not offer the ability to correct hardware error in the first two steps and hence the final error $\tilde{\boldsymbol{x}} + \tilde{\boldsymbol{y}}$ will have non-zero expectation. To correct this and guarantee convergence, we must modify the first two steps in a way that allows us to increase the computational cost and make the statistical moments of $\|\tilde{\boldsymbol{x}}\|$ and $\|\tilde{\boldsymbol{y}}\|$ decay to zero.

Our objective is to take existing algorithms and allow for as many steps as possible in fast mode while still preserving convergence. In order to achieve this goal, we modify the algorithms in such a way as to control the propagation of faults so that the final hardware error has bounded statistics.

3. Analytic framework for fixed point methods.

Iterative linear and non-linear solvers are the most widely utilized approaches for numerical solutions of large scale problems associated with most modern applications. Such methods generate a sequence of approximations of increasing fidelity to the exact solution, terminating after a convergence criteria has been achieved. One of the most widely used class of iterative methods is the family of fixed point solvers, which includes the linear methods such as Jacobi and Gauss-Siedel as well as nonlinear methods, such as Newton. Both types of methods possess properties that enable natural resiliency, and hence, are good example to illustrate our approach.

The basic idea of fixed-point methods is to construct a sequence of vectors $\{\boldsymbol{x}^{(k)}\}_{k=0}^{\infty}$ that enjoy the property of convergence $\boldsymbol{x}^{(k)} \to \boldsymbol{x}_G$, where $\boldsymbol{x}_G$ is a fixed point that satisfies the following linear or nonlinear system $\boldsymbol{G}(\boldsymbol{x}_G) = \boldsymbol{x}_G$. In practice, the fixed point iterative process is stopped at the minimum value n, such that $\|\boldsymbol{x}^{(n)} - \boldsymbol{x}\| < \epsilon$, where ϵ is a fixed tolerance and $\|\cdot\|$ is any convenient vector norm. However, since the exact solution is obviously not available, it is necessary to introduce a suitable stopping criteria. The increment $e^{(k)} = \|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\|$ between two successive iterates is the most common indicator for convergence, when the iteration reaches the asymptotic regime. Yet, this condition may cause premature termination in the first few iterations, therefore, we require that at least k_{min} number of iterations are taken[1]. We summarize the fixed point iteration in the following algorithm:

[1]Depending on the structure of $\boldsymbol{G}(\boldsymbol{x})$, in many applications it is sufficient to take $k_{min} = 1$.

Algorithm 1 (Fixed point iteration).

```
Given $\boldsymbol{G}$, $\boldsymbol{b}$, initial $\boldsymbol{x}^{(0)}$, tolerance $\epsilon$ and $k_{min}$
Let $k = 0$
repeat
  $\boldsymbol{x}^{(k+1)} = \boldsymbol{G}(\boldsymbol{x}^{(k)})$
    $e^{(k)} = \|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\|$
      $k = k + 1$
until $(k > k_{min})$ and $\left(e^{(k)} < \epsilon\right)$
```

Convergence of Algorithm 1 is contingent upon the properties of $G(\boldsymbol{x})$ and the initial iterate $\boldsymbol{x}^{(0)}$. One sufficient condition for convergence is based on the famous fixed point Theorem, a proof of which can be found in [20].

Theorem 1 (Fixed point theorem). *Suppose $\Gamma \subset \mathbb{R}^N$ is complete in some norm $\|\cdot\|$ and let $\boldsymbol{G} : \Gamma \to \Gamma$ be a contraction map, i.e., there is a constant $r < 1$ so that*

$$\|\boldsymbol{G}(\boldsymbol{x}) - \boldsymbol{G}(\boldsymbol{y})\| \leq r\|\boldsymbol{x} - \boldsymbol{y}\|, \qquad \textit{for all } \boldsymbol{x}, \boldsymbol{y} \in \Gamma.$$

Then, there is a unique fixed point $\boldsymbol{x}_G \in \Gamma$ so that $\boldsymbol{G}(\boldsymbol{x}_G) = \boldsymbol{x}_G$. Furthermore, for any $\boldsymbol{x}^{(0)} \in \Gamma$ the sequence $\{\boldsymbol{x}^{(k)}\}_{k=0}^{\infty}$ defined by $\boldsymbol{x}^{(k+1)} = \boldsymbol{G}(\boldsymbol{x}^{(k)})$ converges to $\boldsymbol{x}_G$ (i.e., $\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_G\| \to 0$) and the error $\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_G\|$ is bounded by

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_G\| \leq r^k\|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\|.$$

Theorem 1 is a strong result that not only grantees convergence to the desired solution, but also gives the linear r^k decay of the error. Some fixed point methods exhibit even faster convergence rate, e.g., Newton's method converges quadratically, however, in this work, we focus on methods with linear rate.

Example 3.1 (Linear solvers). *Here we focus on using fixed point methods to solve linear systems of type:*

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{3.1}$$

where $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ is a given matrix, $\boldsymbol{b} \in \mathbb{R}^N$ is a given right hand side vector and we are interested in the solution $\boldsymbol{x}_G \in \mathbb{R}^N$ that satisfies (3.1). Two of the most common fixed point methods for linear systems are known as the Jacobi and Gauss-Siedel methods. The contraction maps and the corresponding contraction coefficients are given by:

- **Jacobi Method**

  $$\boldsymbol{G}_J(\boldsymbol{x}) = \boldsymbol{D}^{-1}\left(\boldsymbol{b} - \boldsymbol{S}\boldsymbol{x}\right), \qquad r_J = \rho(\boldsymbol{D}^{-1}\boldsymbol{S}),$$

  where $\boldsymbol{D} = diag(\boldsymbol{A})$ (i.e., $\boldsymbol{D}$ is a diagonal matrix with the diagonal entries of $\boldsymbol{A}$), $\boldsymbol{S} = \boldsymbol{A} - \boldsymbol{D}$, and ρ indicates the spectral radius of the matrix $\boldsymbol{D}^{-1}\boldsymbol{S}$.

- **Gauss-Seidel Method**

  $$\boldsymbol{G}_{GS}(\boldsymbol{x}) = \boldsymbol{L}^{-1}\left(\boldsymbol{b} - \boldsymbol{U}\boldsymbol{x}\right), \qquad r_{GS} = \rho(\boldsymbol{L}^{-1}\boldsymbol{U}),$$

  where $\boldsymbol{L} = lower(\boldsymbol{A})$ (i.e., $\boldsymbol{L}$ is a lower triangular matrix of the entries of $\boldsymbol{A}$), $\boldsymbol{U} = \boldsymbol{A} - \boldsymbol{L}$, and ρ indicates the spectral radius of the matrix $\boldsymbol{L}^{-1}\boldsymbol{U}$.

The exact solution $\boldsymbol{x}_G$ to (3.1) is a fixed point of both $\boldsymbol{G}_J(\boldsymbol{x})$ and $\boldsymbol{G}_{GS}(\boldsymbol{x})$, therefore, if either $r_J < 1$ or $r_{GS} < 1$, the corresponding fixed point iteration will converge to $\boldsymbol{x}_G$ in l^2 or any equivalent norm, for any $\boldsymbol{x}^{(0)} \in \mathbb{R}^N$ (i.e., $\Gamma \equiv \mathbb{R}^N$).

3.1. Hardware error analysis. The most computationally expensive step of Algorithm 1 is finding the next iterate

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{G}(\boldsymbol{x}^{(k)}), \qquad k = 0, 1, 2, \cdots \tag{3.2}$$

In fact, in most applications this step is orders of magnitude more expensive then computing the increment $e^{(k)}$. However, hardware fault in the evaluation of $e^{(k)}$ may cause premature termination of the algorithm, hence, we assume that the last two steps of the algorithm together with the conditional branching are computed in *safe* mode.

To reduce the computational cost associated with (3.2), we assume that the $k+1$ iterate is computed in fast mode. Then we must allow for the possibility to encounter a silent hardware fault at step k, i.e., instead of $\boldsymbol{x}^{(k+1)}$ we could compute

$$\tilde{\boldsymbol{x}}^{(k+1)} = \boldsymbol{G}(\boldsymbol{x}^{(k)}) + \tilde{\boldsymbol{x}}.$$

If $\tilde{\boldsymbol{x}}^{(k+1)} \in \Gamma$, Algorithm 1 will generate a new sequence $\{\boldsymbol{y}^{(k)}\}_{k=0}^{\infty}$ where $\boldsymbol{y}^{(0)} = \tilde{\boldsymbol{x}}^{(k+1)}$ and $\boldsymbol{y}^{(i+1)} = \boldsymbol{G}(\boldsymbol{y}^{(i)})$, and from Theorem 1 $\boldsymbol{y}^{(i)} \to \boldsymbol{x}_G$. The total numerical error for the new sequence is bounded by

$$\|\boldsymbol{y}^{(i)} - \boldsymbol{x}_G\| \leq r^i \|\boldsymbol{y}^{(0)} - \boldsymbol{x}_G\| \leq r^i \|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}_G\| + r^i \|\tilde{\boldsymbol{x}}\| \leq r^i r^{k+1} \|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\| + r^i \|\tilde{x}\|.$$

The first term of the bound $r^{i+k+1}\|\boldsymbol{x}^{(0)} + \boldsymbol{x}_G\|$ is identical to the one associated with the error free iteration. Therefore, the hardware error associated with the fixed point method that is caused by a single hardware failure is bounded by $r^i\|\tilde{\boldsymbol{x}}\|$, where i is the number of iterations after the failure and $\|\tilde{\boldsymbol{x}}\|$ is the magnitude of the hardware induced perturbation. Analogously, if we encounter j hardware faults at iterations $i_1, i_2, \cdots, i_j$ with perturbations $\tilde{\boldsymbol{x}}^{(i_1)}, \tilde{\boldsymbol{x}}^{(i_2)}, \cdots, \tilde{\boldsymbol{x}}^{(i_j)}$, then we have the following bound

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_G\| \leq r^k \|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\| + \sum_{l=1}^{j} r^{k-i_l} \|\tilde{\boldsymbol{x}}^{(i_l)}\|. \tag{3.3}$$

Thus, the hardware error associated with j silent faults is bounded by $\sum_{l=1}^{j} r^{k-i_l}\|\tilde{\boldsymbol{x}}^{(i_l)}\|$.

Next we consider the convergence of the method with respect to the hardware error as in Definition 2. The terms $r^{k-i_l} \to 0$ as $k \to \infty$, however, for any non-zero r, the supremum in (2.1) will contain the unbounded terms $\|\tilde{\boldsymbol{x}}^{(i_l)}\|$ and hence the expectation and variance of the hardware error will be unbounded. In practice, we expect to encounter perturbations of finite magnitude, however, a sufficiently large perturbation may require a prohibitive number of additional iterations to converge. Our earlier work shows how rescaling the entries of $\boldsymbol{A}$ can be used to reduce the expected value of $\|\tilde{\boldsymbol{x}}^{(i_l)}\|$ [10–12], however, not all problems can be efficiently rescaled and scaling cannot drive the expectation of $\|\tilde{\boldsymbol{x}}^{(i_l)}\|$ to zero. Therefore, the fixed point iteration is not convergent with respect to silent faults if the evaluation of $\boldsymbol{G}(\boldsymbol{x}^{(k)})$ is done in *fast* mode.

3.2. Resilience enhancement techniques and convergence. We assume that the evaluation of $e^{(k)}$ and the conditional branching are computed in *safe* mode and we require that the expensive step that computes $\boldsymbol{x}^{(k)}$ be executed in *fast* mode. The fixed point method has self correcting properties that can negate the effects of hardware error with small magnitude, however, a novel approach is needed to discriminate and reject iterates $\boldsymbol{x}^{(k+1)}$ that are contaminated by hardware error with large magnitude.

First, note that the increments $e^{(k)}$ defined in Algorithm 1 decay monotonically, i.e.,

$$e^{(k)} = \|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\| = \|\boldsymbol{G}(\boldsymbol{x}^{(k)}) - \boldsymbol{G}(\boldsymbol{x}^{(k-1)})\| \leq r\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\| = re^{(k-1)}. \qquad (3.4)$$

The monotonic property (3.4) (i.e., $e_k \leq re_{k-1}$) can be easily verified at each step and deviation from this behavior is an indication of hardware malfunction. If the property fails, i.e., we observe something other than monotonic convergence, then we reject the next iterate $\boldsymbol{x}^{(k+1)}$ and recompute $\boldsymbol{G}(\boldsymbol{x}^{(k)})$. In practice, the value of r may be unknown, hence we can introduce a tuning parameter α, that is an upper estimate such that $\alpha \in (r, 1]$. If such an estimate is not available, we can start the iteration with $\alpha = 1$ and from several increments $e^{(k)}$ we can estimate α as $\alpha \approx e^{(k+1)}/e^{(k)}$. Effectively, this will provide an upper bound on the hardware error introduced by the computations.

The monotonic condition (3.4) can be tested for any iterate $k > 0$, however, the first iterate needs a different choice for $e^{(-1)}$. An upper bound for $e^{(0)}$ is given by

$$\begin{aligned} e^{(0)} = \|\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}\| &= \|\boldsymbol{x}^{(1)} - \boldsymbol{x}_G + \boldsymbol{x}_G - \boldsymbol{x}^{(0)}\| \\ &\leq r\|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\| + \|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\| \\ &= (r+1)\|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\| \\ &\leq (\alpha+1)\beta, \end{aligned}$$

where β is a second tuning parameter that is an upper bound for $\|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\|$. Thus, we set $e^{(-1)} = (\alpha+1)\beta$, which is only needed for the first step of the iteration.

With all this in mind, we propose the following improved fixed point algorithm, where all steps except $\boldsymbol{x}^{(k+1)} = \boldsymbol{G}(\boldsymbol{x}^{(k)})$ are executed in safe mode:

Algorithm 2 (Improved fixed point iteration).

```
Given $\boldsymbol{G}$, initial $\boldsymbol{x}^{(0)}$, tolerance $\epsilon$ and $k_{min}$
Select $\alpha \in (r, 1]$ and $\beta > \|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\|$
Let $e^{(-1)} = (\alpha+1)\beta$
$k = 0$
repeat
  $\boldsymbol{x}^{(k+1)} = \boldsymbol{G}(\boldsymbol{x}^{(k)})$
  $e^{(k)} = \|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\|$
  if $e^{(k)} < \alpha e^{(k-1)}$ then
      Accept $\boldsymbol{x}^{(k+1)}$ (i.e., $k = k+1$)
  else
    Reject the step (i.e., make no modifications to $k$)
  end if
until $(k < k_{min})$ or $\left(e^{(k)} < \epsilon\right)$
```

The evaluation of $\boldsymbol{G}(x^{(k)})$ in Algorithm 2 is the only step that is susceptible to silent faults. We note that it is possible for a silent fault encountered at iteration k to result in failure of the monotonic condition $e^{(k+2)} < \alpha e^{(k+1)}$ at step $k+2$, even if step $k+1$ was computed without any faults. This delayed rejection can result in stagnation of Algorithm 2 and we address this issue in §3.4. First, we focus on an upper bound for the sum of numerical error and hardware errors.

We take an approach similar to [12]. Suppose we encounter a fault at iteration k, then we replace $\boldsymbol{x}^{(k+1)}$ by

$$\tilde{\boldsymbol{x}}^{(k+1)} = \boldsymbol{G}(\boldsymbol{x}^{(k)}) + \tilde{\boldsymbol{x}}.$$

For example, when $k = 0$, faulty initial iterate will be rejected and completely corrected unless

$$\|\boldsymbol{x}^{(1)} + \tilde{\boldsymbol{x}} - \boldsymbol{x}^{(0)}\| < (\alpha+1)\beta.$$

On the other hand, if we accept a faulty iterate $\tilde{\boldsymbol{x}}^{(1)}$, we consider a new iteration $\boldsymbol{y}^{(0)} = \tilde{\boldsymbol{x}}^{(1)}$ such that

$$\begin{aligned} \|\boldsymbol{y}_i - \boldsymbol{x}_G\| &\leq r^i \|\tilde{\boldsymbol{x}}^{(1)} - \boldsymbol{x}_G\| \\ &\leq r^i \|\tilde{\boldsymbol{x}}^{(1)} - \boldsymbol{x}^{(0)} + \boldsymbol{x}^{(0)} - \boldsymbol{x}_G\| \\ &\leq r^i \left((\alpha+1)\beta + \|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\| \right). \end{aligned} \tag{3.5}$$

It is important to note that the error bound for the new iteration, given by (3.5), is independent of the magnitude of the error $\|\tilde{\boldsymbol{x}}\|$ introduced; and depends only on the parameters α, β and the accuracy of the initial iterate $\|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\|$. Furthermore, since $r^i \to 0$, the improved algorithm is convergent with respect to a fault in the first iteration (as $i \to \infty$).

Next, we consider faults encountered at iterations $k > 0$. In order to account for the possibility of multiple faults, we employ an inductive argument that we base on the following lemma.

Lemma 1 (Magnification of error). *Let $\{\boldsymbol{x}^{(s)}\}_{s=0}^{k}$ be a sequence of iterates generated by Algorithm 2 and suppose that for some constant $C > 0$ and some integer $\tilde{k} \leq k$ the following holds*

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_G\| \leq C r^{\tilde{k}} \quad \text{and} \quad \|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\| \leq C r^{\tilde{k}} \tag{3.6}$$

If a fault is encountered during computation of $\boldsymbol{x}^{(k+1)}$, then for $i > k$ (assuming no further faults are encountered) the following estimate hold

$$\|\boldsymbol{x}^{(i)} - \boldsymbol{x}_G\| \leq (1+\alpha) C r^{i-k+\tilde{k}-1} \quad \text{and} \quad \|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(i-1)}\| \leq C r^{i-k+\tilde{k}-1} \tag{3.7}$$

Proof: If we encounter a fault at iteration $k > 0$, Algorithm 2 will reject any perturbation $\tilde{\boldsymbol{x}}^{(k)}$ unless

$$\|\boldsymbol{x}^{(k+1)} + \tilde{\boldsymbol{x}} - \boldsymbol{x}^{(k)}\| < \alpha \|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\|.$$

Rejection of the fault results in the fixed point iteration simply being delayed by one iteration, hence (3.7) holds trivially. If the fault is accepted, then we obtain a fault

$\tilde{\boldsymbol{x}}^{(k+1)} = \boldsymbol{x}^{(k+1)} + \boldsymbol{x}$ if the iteration proceeds without any further faults for $i > k$

$$\begin{aligned}
\|\boldsymbol{x}^{(i)} - \boldsymbol{x}_G\| &\leq r^{i-k-1}\|\tilde{\boldsymbol{x}}^{(k+1)} - \boldsymbol{x}_G\| \\
&\leq r^{i-k-1}\|\tilde{\boldsymbol{x}}^{(k+1)} - \boldsymbol{x}^{(k)} + \boldsymbol{x}^{(k)} - \boldsymbol{x}_G\| \\
&\leq r^{i-k-1}\alpha\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\| + r^{i-k-1}\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_G\| \\
&\leq Cr^{i-k-1}\alpha r^{\tilde{k}} + Cr^{i-k-1}r^{\tilde{k}} \\
&= C(1+\alpha)r^{i-k-1+\tilde{k}}
\end{aligned}$$

Similarly,

$$\begin{aligned}
\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(i-1)}\| &\leq r^{i-k-1}\|\tilde{\boldsymbol{x}}^{(k+1)} - \boldsymbol{x}^{(k)}\| \\
&\leq \alpha r^{i-k-1}\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\| \\
&\leq \alpha C r^{i-k-1+\tilde{k}} \quad \Box
\end{aligned}$$

Combining (3.5) and Lemma (1) we now present the main result of this section

Theorem 2 (Multiple faults). *Suppose that Algorithm 2 is executed for k iterations and it encounters $j \leq k$ faults at iterations $k_1 < k_2 < \cdots < k_j$. Then, at iteration k*

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_G\| \leq C_{max} r^{k-j}(1+\alpha)^j, \tag{3.8}$$

where

$$C_{max} = \max\left\{\frac{(1+r)\|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\|}{r}, \frac{(\alpha+1)\beta + \|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\|}{r}\right\}$$

Proof: If $k_1 > 0$ (i.e., the first iteration doesn't encounter faults), then for $k \leq k_1$ assumption (3.6) holds with $C = r^{(-1)}(1+r)\|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\|$ and $\tilde{k} = k$, and therefore, for $k \geq k_1$ and $k < k_2$ we have that

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_G\| \leq (1+\alpha)C_{max}r^{k-1}, \quad \text{and} \quad \|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\| \leq \alpha C_{max} r^{k-1}.$$

Conversely, if $k_1 = 0$, then according to (3.5), for $0 < k \leq k_2$ assumption (3.6) holds with $C = r^{-1}\left((\alpha+1)\beta + \|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\|\right)$ and $\tilde{k} = k - 1$, and therefore, for $k \geq k_2$ and $k < k_3$

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_G\| \leq (1+\alpha)C_{max}r^{k-2}, \quad \text{and} \quad \|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\| \leq C_{max} r^{k-2}.$$

Inductively, suppose that for $k \geq k_{j-1}$ and $k < k_j$, we have that assumption (3.6) holds with $C = C_{max}(1+\alpha)^{j-1}$ and $\tilde{k} = k - j + 1$, then according to Lemma 1 we have that for $k \geq k_j$

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_G\| \leq (1+\alpha)^j C_{max}r^{k-j}, \quad \text{and} \quad \|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\| \leq \alpha(1+\alpha)^{j-1} C_{max} r^{k-j} \quad \Box$$

The result in Theorem 1 gives an estimate on the error associated with the resilient fixed point algorithm (i.e., Algorithm 2). More importantly, the estimate is independent of the magnitude of the introduced perturbations $\tilde{\boldsymbol{x}}$ and the iterations (k_j) where the faults occur. Every time we encounter a fault, we waste one iteration (i.e., we decrement the exponent of r by one) and the fault introduces a magnification factor of $1 + \alpha$. This allows us to bound the statistics of the error independent of $\tilde{\boldsymbol{P}}$ and we need only focus our attention to the number of faults encountered in k iterations.

Each iteration of the fixed point scheme has approximately the same computational complexity, hence, we expect that the rate of faults, i.e., the Bernoulli parameters, associated with each step to be essentially constant. Let $p_k = p$ for all iterations k, then the number of faults j encountered in k iterations follows a binomial distribution. Let $\rho(j|k)$, for $0 \leq j \leq k$, indicate the binomial probability measure, i.e., the probability of encountering j faults in k total iterations with fault rate p. Then, according to Theorem 2, for any distribution $\tilde{\boldsymbol{P}}$ of the hardware induced perturbations, the statistical moments of the total numerical error can be bounded by:

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{p},\tilde{\boldsymbol{P}}}\left[\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_G\|\right] &\leq C_{max}\sum_{j=0}^{k} r^{k-j}(1+\alpha)^j \rho(j|k), \\
\mathbb{V}_{\boldsymbol{p},\tilde{\boldsymbol{P}}}\left[\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_G\|\right] &\leq C_{max}^2\sum_{j=0}^{k} r^{2k-2j}(1+\alpha)^{2j} \rho(j|k).
\end{aligned} \tag{3.9}$$

Therefore, the improved iteration is convergent if

$$\begin{aligned}
&\lim_{k\to\infty}\sum_{j=0}^{k} r^{k-j}(1+\alpha)^j \rho(j|k) = 0, \\
&\lim_{k\to\infty}\sum_{j=0}^{k} r^{2k-2j}(1+\alpha)^{2j} \rho(j|k) = 0.
\end{aligned} \tag{3.10}$$

Condition (3.10) is sufficient for convergence, so long as Algorithm 2 does not encounter a false positive rejection of a fault free iterate. If multiple faults are accepted in consecutive iterations, it is possible for Algorithm 2 to stagnate. We address this issue in §3.4, however, we first look at the rate of convergence of the modified algorithm.

3.3. Convergence Rate. The classical fixed point algorithm converges linearly, i.e., $O(r^k)$, where r is the contraction constant (e.g., the spectral radius of the iteration matrix) and k is the number of iterations. However, in the presence of hardware faults, even if the algorithm produces an accurate approximation to the desired solution, the rate of convergence often times deteriorates [11, 15]. Furthermore, the slowdown is strongly influenced by the number of introduced hardware faults. Therefore, we are interested in the rate of convergence as a function of the rate of faults p.

The statistical moments of the hardware error are bounded in (3.9), where $\rho(j|k)$ is the binomial measure

$$\rho(j|k) = \frac{k!}{j!(k-j)!}p^j(1-p)^{k-j}. \tag{3.11}$$

Then, by substituting (3.11) into (3.9), and combining like terms we obtain the estimates

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{p},\tilde{\boldsymbol{P}}}\left[\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_G\|\right] &\leq C_{max}\sum_{j=0}^{k}\frac{k!}{j!(k-j)!}r^{k-j}(1+\alpha)^j p^j (1-p)^{k-j} \\
&\leq C_{max}\sum_{j=0}^{k}\frac{k!}{j!(k-j)!}(r(1-p))^{k-j}((1+\alpha)p)^j, \\
\mathbb{V}_{\boldsymbol{p},\tilde{\boldsymbol{P}}}\left[\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_G\|\right] &\leq C_{max}^2\sum_{j=0}^{k}\frac{k!}{j!(k-j)!}r^{2k-2j}(1+\alpha)^{2j} p^j (1-p)^{k-j} \\
&\leq C_{max}^2\sum_{j=0}^{k}\frac{k!}{j!(k-j)!}(r^2(1-p))^{k-j}((1+\alpha)^2 p)^j.
\end{aligned}$$

Using the binomial theorem the expectation and variance are given by

$$\begin{aligned}
\sup_{\tilde{\boldsymbol{P}}}\mathbb{E}_{\boldsymbol{p},\tilde{\boldsymbol{P}}}\left[\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_G\|\right] &\leq C_{max}\left(r(1-p)+(1+\alpha)p\right)^k, \\
\sup_{\tilde{\boldsymbol{P}}}\mathbb{V}_{\boldsymbol{p},\tilde{\boldsymbol{P}}}\left[\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_G\|\right] &\leq C_{max}^2\left(r^2(1-p)+(1+\alpha)^2 p\right)^k
\end{aligned} \tag{3.12}$$

If $(r(1-p)+(1+\alpha)p) < 1$ and $\left(r^2(1-p)+(1+\alpha)^2 p\right) < 1$, then the right hand side of (3.12) can be made arbitrarily small by increasing k, i.e., performing a sufficient number of iterations (extra work). Thus, Algorithm 2 will be convergent in terms of (2.1) if the following condition is satisfied

$$p < \frac{1-r}{(1+\alpha)-r} \qquad \text{and} \qquad p < \frac{1-r^2}{(1+\alpha)^2 - r^2}. \tag{3.13}$$

From (3.13) we get a relationship between the contraction properties of $\boldsymbol{G}(\boldsymbol{x})$ and the reliability of the hardware, required to guarantee convergence of the resilient fixed point iteration. In particular, the convergence condition (3.13) allows us to quantity the following:

(i) For perfectly reliable hardware, i.e., $p = 0$, the exponential term of (3.12) becomes,

$$(r(1-p)+(1+\alpha)p)^k = r^k$$

Furthermore, the r^{-1} coefficient and the $(\alpha+1)\beta + \|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\|$ terms in the definition of C_{max} manifest only at the occurrence of a fault, hence, when $p = 0$ we can take $C_{max} = \|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\|$ and thus recover the rate and constant associated with the convergence of the classical fixed point iteration.

(ii) As the hardware fault rate p increases, then $r(1-p) + (1+\alpha)p$ increases and the convergence deteriorates accordingly.

(iii) As $r \to 1$ and the conditioning of the fixed point method deteriorates, the convergence condition forces $p \to 0$, which implies problems with worse conditioning require more reliable hardware.

(iv) The constant C_{max} is of order $1/r$, however, Theorem 1 states that error of magnitude $1/r$ can be correct with a single (fault free) iteration. Thus, for a fixed p, as $r \to 0$, the C_{max} diverges but Algorithm 2 remain convergent.

Therefore, the modified fixed point iteration performs just as well as the classical method when executed on reliable hardware, while at the same time Algorithm 2 allows for convergence, even in the presence of silent hardware faults. Unreliable hardware would deteriorate the rate of convergence, which can be rigorously quantified by (3.12) and (3.13).

3.4. Guarding Against False Positive Rejections. The accept/reject criteria discussed above gives an upper bound for the statistics of the total numerical error. However, it is possible for a hardware fault to create a false-positive rejection of a following fault-free iteration. For example, suppose that at step k the result of $\boldsymbol{G}(\boldsymbol{x}^{(k)})$ is perturbed so that

$$\tilde{\boldsymbol{x}} \approx -\boldsymbol{G}(\boldsymbol{x}^{(k)}), \qquad \tilde{\boldsymbol{x}}^{(k+1)} = \boldsymbol{G}(\boldsymbol{x}^{(k)}) + \tilde{\boldsymbol{x}} \approx \boldsymbol{x}^{(k)}.$$

Even though $e^{(k)}$ is computed in safe mode, the perturbation $\tilde{\boldsymbol{x}}$ will make $e^{(k)}$ very small, which can cause either an early termination or any fault-free $k+2$ iterate would no longer be accepted and Algorithm 2 will stagnate.

We propose a number of ways to address stagnation. The first possible approach is to apply redundancy techniques, where we assume that the probability of encountering two consecutive hardware failures that result in identical hardware error is negligible. This assumption is based on the consideration that every component of a supercomputer has a relatively low rate of failure and faults in different components are likely to result in very different hardware error. Suppose iteration k was rejected, then we store the computed presumably faulty $\boldsymbol{x}^{(k+1)}$ and repeat the last iteration. If we reject the repeated iteration as well, then we compare the two rejected iterates, if they are within ϵ of each other, then we overwrite the accept/reject criteria and force the acceptance of $\boldsymbol{x}^{(k+1)}$ as the next iterate.

Another approach is to reject the last two iterates, upon encountering a fault, i.e., we reject both x_{k+1} and x_k. This is a very conservative approach that will guard against stagnation, however, every faulty iterate may result in the rejection of a fault free iterate as well. This is a feasible approach only if the rate of hardware faults is low and the increased storage cost of keeping two consecutive iterates is not of consideration.

Another redundancy approach would be to keep the history of the norms of the increments and test e_k against several of the past $e_{k-1}, e_{k-2}, \cdots$. That is

```
if ($e^{(k)} < \alpha e^{(k-1)}$) or ($e^{(k)} < \alpha^2 e^{(k-2)}$) or ($e^{(k)} < \alpha^3 e^{(k-3)}$) then
    Accept $\boldsymbol{x}^{(k+1)}$
else
    Reject $\boldsymbol{x}^{(k+1)}$
end if
```

While computationally and storage wise cheap, this approach will still fail if we encounter multiple consecutive faults causing stagnation, hence this method is only feasible if the probability of such event is negligible. Furthermore, if α is an overestimate of r, then α^3 is an even bigger overestimate of r^3 and hence this approach allows the acceptance of hardware error with larger magnitude.

In addition, there are multiple approaches to guard against early termination. Postprocessing in safe mode will guarantee that the residual associated with the final computed iterate is indeed within ϵ. If the test fails, we can restart the fixed point iteration with the last iterate as the new initial guess $\boldsymbol{x}^{(0)}$. This is a very safe, but potentially expensive approach, as it requires one evaluation of (3.1) in safe mode.

Another approach is to require that two or more of the last computed increments $e^{(k)}$ simultaneously satisfy the convergence criteria $e^{(k)} < \epsilon$, i.e., we replace the convergence test by

```
repeat
    ...
until $\left(e^{(k)} < \epsilon\right)$ and $\left(e^{(k-1)} < \epsilon/r\right)$ and $\left(e^{(k-2)} < \epsilon/r^2\right) \cdots$
```

So long as encountering multiple consecutive faults that cause early termination has negligible probability, this approach will guarantee convergence.

All of above techniques are viable under different circumstances. We select the two that in the authors' opinion are most robust and most widely applicable. Specifically we consider the following Resilient fixed point Iteration.

Algorithm 3 (Resilient fixed point Iteration).

```
Given $\boldsymbol{G}$, initial $\boldsymbol{x}_0$, tolerance $\epsilon$ and $k_{min}$
Let $\alpha \in (r, 1]$, $\beta > \|\boldsymbol{x}^{(0)} - \boldsymbol{x}_G\|$
Let $e^{(-1)} = (\alpha + 1)\beta$
$k = 0$
repeat
    $\boldsymbol{x}^{(k+1)} = \boldsymbol{G}(\boldsymbol{x}^{(k)})$
    $e^{(k)} = \|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\|$
    if $e^{(k)} \leq \alpha e^{(k-1)}$ then
        Accept $\boldsymbol{x}^{(k+1)}$, i.e., $k = k + 1$
    else
        if Last iterate was rejected and $\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}_f\| \leq \epsilon$ then
            Accept $\boldsymbol{x}^{(k+1)}$ (i.e., $k = k + 1$)
        else
            Reject the step, i.e., make no modifications to $k$
            $\boldsymbol{x}_f = \boldsymbol{x}^{(k+1)}$
        end if
    end if
until $\left(e^{(k)} < \epsilon\right)$ and $\left(e^{(k-1)} < \epsilon/\alpha\right)$
```

4. Numerical Example.

In this section, we verify the theoretical results by comparing the standard and resilient fixed point methods, when solving a parabolic partial differential equation (PDE). In particular, we consider the Jacobi method applied to the linear system of equations associated with one step of the implicit time integration of the heat equation. Let $\Omega = [0,1] \times [0,1] \subset \mathbb{R}^2$ and consider the PDE

$$\begin{aligned}
\frac{d}{d\tau} u(\tau,\xi,\eta) &= \frac{\partial^2}{\partial \xi^2} u(\tau,\xi,\eta) + \frac{\partial^2}{\partial \eta^2} u(\tau,\xi,\eta), \qquad (\xi,\eta) \in \Omega, \tau > 0, \\
u(\tau,\xi,\eta)|_{\partial\Omega} &= 0, \\
u(0,\xi,\eta) &= \xi\eta(\xi-1)(\eta-1).
\end{aligned}$$

We seek a numerical approximation to the solution $u(\tau, \xi, \eta)$. We discretize the problem in Ω using a finite difference scheme with uniformly distributed nodes. Let

$$\{\xi_i\}_{i=1}^n, \quad \xi_i = \frac{i}{n+1}, \qquad \{\eta_j\}_{j=1}^n, \quad \eta_j = \frac{j}{n+1},$$

and approximate

$$u(\tau, \xi_i, \eta_j) \approx u_{i,j}(t), \qquad u_{i,j}(0) = \xi_i \eta_j (\xi_i - 1)(\eta_j - 1).$$

In this setting, the diffusion operator is discretized as

$$\begin{aligned} \frac{\partial^2}{\partial \xi^2} u_{i,j}(\tau) &\approx \frac{u_{i-1,j}(\tau) - 2u_{i,j}(\tau) + u_{i+1,j}(\tau)}{\Delta \xi^2}, \\ \frac{\partial^2}{\partial \eta^2} u_{i,j}(\tau) &\approx \frac{u_{i,j-1}(\tau) - 2u_{i,j}(\tau) + u_{i,j+1}(\tau)}{\Delta \eta^2}, \end{aligned}$$

where $\Delta\xi = \Delta\eta = \frac{1}{n+1}$, and $u_{0,j}(\tau) = u_{n+1,j}(\tau) = u_{i,0}(\tau) = u_{i,n+1}(\tau) = 0$. The spacial discretization results in n^2 ordinary differential equations that can be written in a matrix form

$$\frac{d}{d\tau} \boldsymbol{v}(\tau) = \boldsymbol{L}\boldsymbol{v}(\tau),$$

where $u_{i,j}(\tau)$ is given by $v_m = u_{i,j}(\tau)$, with $m = (i-1)n + j$, and $\boldsymbol{L}$ is the matrix representation of the discretized Laplacian operator.

Finally, we evolve the system in time with the use of a backward Euler method. That is, we select a time step $\Delta\tau$ and approximate

$$\boldsymbol{v}(\tau + \Delta\tau) \approx (\boldsymbol{I} - \Delta\tau \boldsymbol{L})^{-1} \boldsymbol{v}(\tau),$$

where $\boldsymbol{I}$ is the identity matrix. At each time step we need to solve a system of linear equations. In order to study the resilience properties of the fixed point iteration, we consider only the first linear system (i.e., $\tau = 0$), such that

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},$$

where $\boldsymbol{A} = \boldsymbol{I} - \Delta\tau\boldsymbol{L}$, $\boldsymbol{b} = \boldsymbol{v}(0)$ and $\boldsymbol{x} \approx \boldsymbol{v}(\Delta\tau)$.

4.1. Simulating Hardware Faults. We perform all computations on a desktop computer with Intel Sandy-Bridge-E 6-core CPU, and we implement our algorithms using MATLAB. The probability of a silent fault on the machine for the short duration of the computation is negligible. Nevertheless, we perform all tests twice to ensure consistency in the results. Since the hardware failure rate on this machine is low, we need to artificially introduce faults into the computations. At every iteration, we poll a pseudo random number using MATLAB's function `rand(1,1)`, which returns a uniformly distributed number in the range $(0, 1)$. If the random number is smaller than a specified threshold p, we introduce a perturbation to the evaluation of $\boldsymbol{x}_{k+1}$. We consider two distribution of the perturbations.

The first case, denoted by $\tilde{\boldsymbol{P}}_u$, uses the following formula

$$\tilde{\boldsymbol{x}} = 10^z \frac{1}{\|\boldsymbol{g}\|}\boldsymbol{g},$$

where $\boldsymbol{g} \in \mathbb{R}^{n^2}$ is a vector with each component sampled from unit Gaussian distribution, i.e., MATLAB's function `randn(n*n,1)` and $z \in [-9, 10]$ is uniformly distributed over the range, i.e., MATLAB's function `20*rand(1,1) - 9`. We have chosen this form of the perturbation because all directions in $\mathbb{R}^{n^2}$ are well described, i.e., $\frac{1}{\|\boldsymbol{g}\|}\boldsymbol{g}$ is a unit vector with uniform distribution on the unit ball, and we have a representation of all the magnitudes in the range 10^{-9} to 10^{10}. While it is possible to encounter hardware fault that results in bigger error, 10^{10} is sufficiently large to show the shortcomings of the classical fixed point method described in Algorithm 1, and larger error would be easily rejected by our resilient approach, i.e., Algorithm 3. On the other hand, hardware error that is significantly less than ϵ will not have an effect on either algorithm. In what follows, we use $\epsilon = 10^{-8}$ and hence 10^{-9} is a suitable lower cutoff bound for the perturbations.

In the second case, we consider the worst possible scenario denoted by $\tilde{\boldsymbol{P}}_w$. In the case of linear fixed point methods (e.g., Jacobi method) the worst perturbation $\tilde{\boldsymbol{x}}_w$ is aligned with the dominant characteristic direction of the iteration matrix and the magnitude $\|\tilde{\boldsymbol{x}}_w\|$ is the largest magnitude that will be accepted by the test criteria, i.e., $\tilde{\boldsymbol{P}} = \delta_0(\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}_w)$. Thus, the occurrence of faults remains random, however, the perturbations $\tilde{\boldsymbol{x}}$ are the solutions to the deterministic optimization problems that maximizes $\|\tilde{\boldsymbol{x}}\|$ in the direction of the dominant eigenvector of the iteration matrix, subject to the constraint that the perturbation has to be accepted by the test criteria of Algorithm 3.

4.2. Classical fixed point Iteration. We first consider the classical fixed point method described in Algorithm 1. We take $n = 100$, which results in a system of $10,000$ linear equations, i.e., $\boldsymbol{A} \in \mathbb{R}^{10,000\times 10,000}$. We consider the Jacobi fixed point method with $\Delta t = 10^{-4}$, $\boldsymbol{x}_0 = \boldsymbol{b}$ and $\epsilon = 10^{-8}$ and the spectral radius of the resulting iteration matrix is $r \approx 0.8028$. We apply Algorithm 1 without any simulated hardware faults. In Figure 1 (dashed line on the right panel) we observe that the method converges linearly and it takes 83 iterations.

Next we want to observe the effects of the simulated hardware faults. We select the rate of silent faults to be one in ten, i.e., $p = 0.1$ and we use $\tilde{\boldsymbol{P}}_u$. This rate is extremely high compared to real world hardware, however, for the purpose of this study we want the expected number of faults to be significant in every realization.

The results from 4 simulations are shown in Figure 1 (left). In tests 1 and 2, the error does not drop below 10^{-7} in the first 1500 iterations and the method fails to return an accurate answer. During test 3, the method reaches the tolerance in little over 500 iterations, after encountering a large number of consecutive fault-free iterations. Due to the large fault rate, encountering such a lucky run of no faults is a possible but improbable event and hence Algorithm 1 is unreliable.

In addition to the individual tests, we are also interested in the statistics of the total numerical error. We perform $10,000$ tests and observe the average numerical error as a function of the number of iterations. Figure 1 (right) shows the lack of convergence for the first 1500 iterations. The expected value of the total numerical error depends on the rate of faults and for a sufficiently small p, it is possible that the expectation can drop below $\epsilon = 10^{-8}$ (e.g., see test case 3). Furthermore, if p is sufficiently small, the method will have a significant chance of not encountering any faults and returning an accurate approximation to the solution $\boldsymbol{x}_G$. However, the mean error will be bounded away from zero for any non-negative p. A resilient algorithm should provide results that are accurate in expectation for any arbitrary ϵ, for as large of fault rate p as possible, therefore, according to Definition 2 the classical fixed point iteration does not converge with respect to hardware induced error.

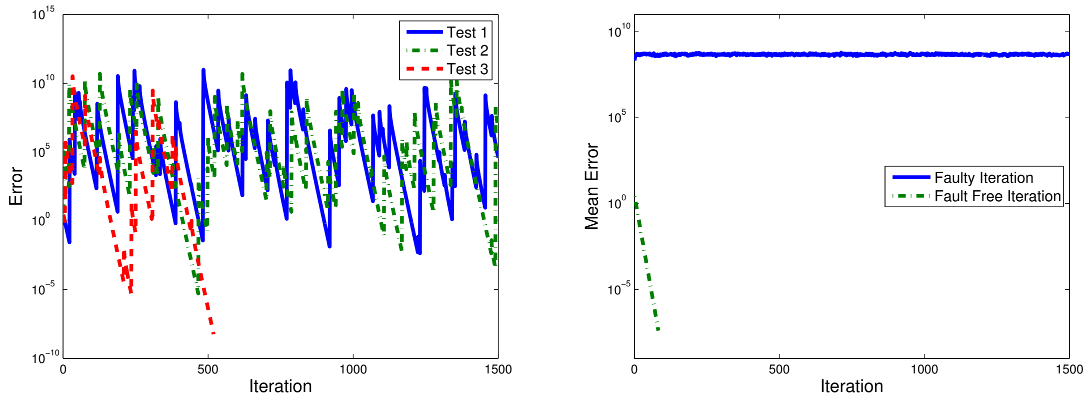

Figure 1: The Jacobi method implemented using the classical fixed point itereration of Algortihm 1 with simulated hardware faults at rate $p = 0.1$. **Left**: the result of four sample tests, where only Test 3 converged to desired tolerance but even that case took over 500 iterations. **Right**: (solid line) the expected value of the error as a function of the iteration computed by taking the average of 10,000 samples, i.e., with the use of Monte Carlo sampling. (dashed line) the method implemented without simulated faults converges in 82 iterations.

4.3. Resilient fixed point Iteration. We repeat the same set of experiments described in the previous section by keeping all of the parameters as described in section 4.2, except we replace the classic fixed point iteration with the resilient fixed point Algorithm 3. The tuning parameters that we use are $\alpha = 1$ and $\beta = 2\|\boldsymbol{b}\|$, both of which are overestimate and hence conservatively safe. Figure 2 (left) shows the result of 3 tests compared to an error free iteration. In contrast to the classical method, the resilient iteration is much more reliable and despite the large number of simulated hardware faults, all tests converge in less than 100 iterations.

The test were performed with $p = 0.2$, which is in fact in violation of the convergence condition (3.13), which implies that the bound on p is conservative and hence (3.13) is sufficient but not necessary. This is due to the fact that the magnification factor $(1 + \alpha)$ assumes that all hardware faults result in accepting a perturbation of maximum possible magnitude. In practice, many of the faulty iterates are either rejected or have a smaller magnitude and thus $(1+\alpha)$ is an overestimate. In Figure 2 (right), we plot the expected total numerical error as a function of the iteration for three different fault rates $p = 0.05, 0.2, 0.3$. While larger p implies that the algorithm would require more iterations to converge, the resilient algorithm returns a reliable result even if 30% of the iterations are computed with faults. In Figure 3 (left), we plot just the mean for the $p = 0.3$ case and the mean plus one standard deviation, demonstrating that we see convergence in both expectation and variance. Thus, according to Definition 2, the resilient fixed point method converges even in the presence of hardware faults.

Additionally, Figure 3 (right) shows the mean number of faults that are detected (i.e., rejected), allowed (i.e., faults resulting in perturbations that is not rejected) and false-positive (i.e., rejections without a fault). The number of accepted and rejected faults increases with p, however, the number of false-positive remains very small for any p. Thus, the criteria defending against false-positive rejection rarely comes to use.

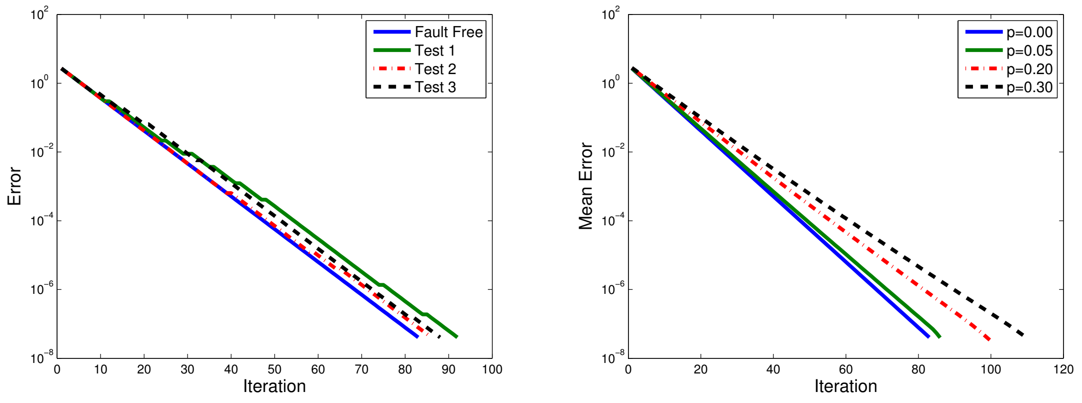

Figure 2: The Jacobi method implemented using the resilient fixed point iteration of Algortihm 3 with simulated hardware faults. **Left**: superimposed fault free iteration compared to three iterations using fault rate of $p = 0.2$. **Right**: average of the results using $10,000$ simulations and observe convergence of the expected value of the total numerical error.

4.4. Considering the Worst Case Scenario. In this section we test the performance of the resilient Algorithm 3 using the worst case faults defined by $\tilde{\boldsymbol{P}}_w$. The purpose of the tests is to demonstrate the limitations of the method as well as the sharpness of the estimate (3.13).

We take $N = 100$, $\Delta\tau = 10^{-4}$ (resulting in $r \approx 0.8028$), $\alpha = 1$, $\beta = 2\|\boldsymbol{b}\|$ and $\epsilon = 10^{-8}$. According to (3.13), the variance of the error associated with Algorithm 3 will converge for $p < 0.1060$, for $p > 0.1060$ but $p < 0.1647$ the mean error will tend to zero but the variance will not, for $p > 0.1647$ the method will not be convergent. In Figure 4, we show the average results of $10,000$ realization of Algorithm 3 with different values of p. Indeed, when $p = 0.08$ we see convergence of mean and variance. For $p = 0.11$, we see convergence of the average error, but also large fluctuations of the mean indicating large variance, the problem with the variance becomes even more pronounced for $p = 0.13$. Note that on the semi-log plot we need large accuracy to obtain a *smooth* line, however, Monte Carlo accuracy is $\sigma/\sqrt{S}$, where σ is the standard deviation of the error for a given k and S is the total number of samples, thus, unless σ tends to zero as a function of k, we require a prohibitive number of samples to make a smooth plot. In the case of $p = 0.14$, the convergence is too slow for the

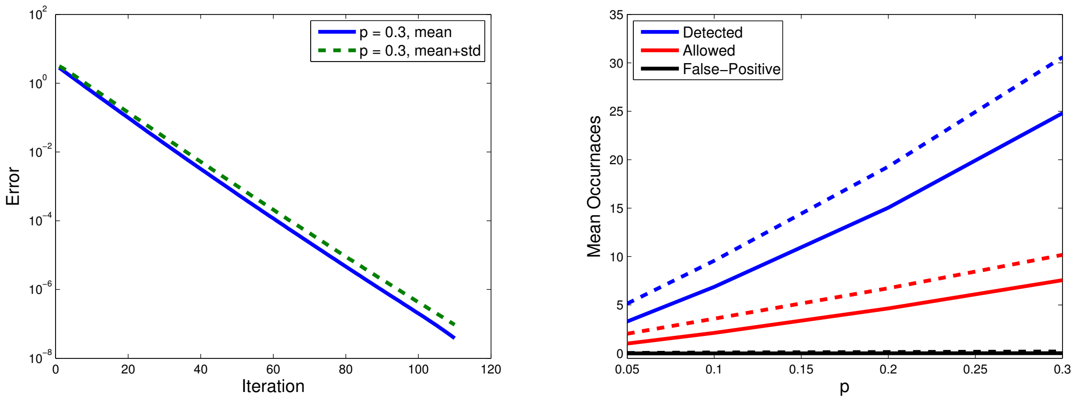

Figure 3: The Jacobi method implemented using the resilient fixed point iteration of Algortihm 3 with simulated hardware faults. **Left**: mean and mean plus one standard deviation of the error for the Jacobi method with $p = 0.3$ **Right**: average number of detected, allowed, and false-positive faults observed in $10,000$ simulations.

method to be practical. As soon as we violate condition (3.13) for the mean, at $p = 0.17$, Algorithm 3 is no longer convergent. This demonstrates the sharpness of our results.

Next, we fix $N = 100$, $p = 0.4$, $\alpha = 0.4$, $\beta = 2\|\boldsymbol{b}\|$ and $\epsilon = 10^{-8}$, and we vary $\Delta\tau$ making it smaller and smaller, which results in $r \to 0$. In the classical fixed point iteration, as $r \to 0$, the convergence rate increases and the method takes fewer and fewer iterations. However, according to (3.12), as $r \to 0$ the term under the exponent will converge to $(1+\alpha)p$ (or $(1+\alpha)^2 p$ for the variance), hence, at some point, we will not observe faster convergence even if we decrease r. Figure 4 (right) shows four examples corresponding to small r, and indeed we see very little difference between the convergence rate when $r = 0.0392$ and $r = 0.0041$ (corresponding to $\Delta\tau = 5 \times 10^{-7}$ and $\Delta\tau = 10^{-7}$). On the other hand, at $k = 1$ there is little difference in the expected error and hence we do not observe the predicted $1/r$ behavior of the constant C_{max}, which suggests that the estimate of the constant can be improved in future work.

5. Conclusion.

This work demonstrates an analytic approach improving fixed point methods in the presence of silent hardware faults. Utilizing hardware and software based resilience techniques, we split our algorithm into *safe* and *fast* modes. While most of the computations are performed in the *fast* mode, a well chosen accept/reject criteria executed in *safe* mode can control the introduction and propagation of hardware induced error.

We extend the current framework of numerical analysis by removing the assumption that computations in fast mode can be performed reliably. We propose an architecture agnostic error model that can be used to analyze the hardware error propagation and an algorithm's convergence properties. In particular, we analyze the classical fixed point iteration and proposed several modifications that can dramatically improve resilience. We perform a numerical comparison between the standard and resilient fixed point methods. We demonstrate that the classical approach fails to converge in expectation, while the resilient method converges even when encountering a very high rate of silent hardware faults.

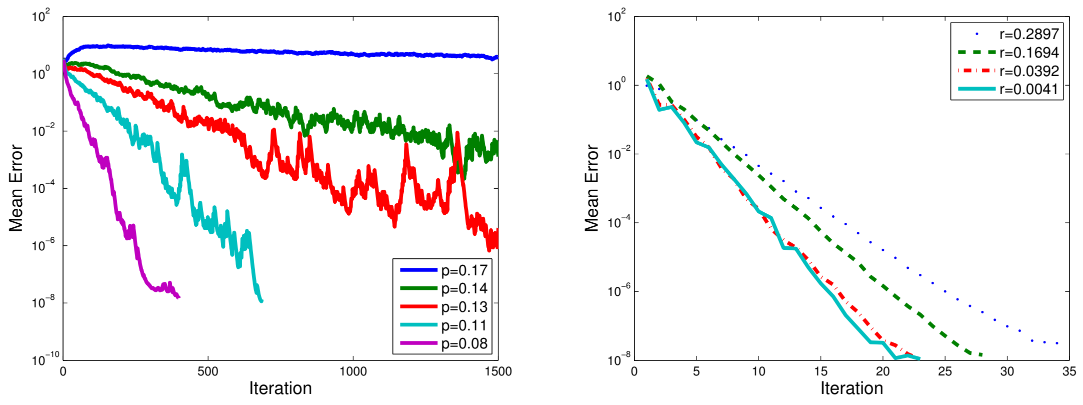

Figure 4: The Jacobi method implemented using the resilient fixed point iteration of Algortihm 3 with simulated hardware faults. **Left**: mean error for different values of p correspoding (bottom to top) to convergent behavior ($p = 0.08$), convergent mean but not variance ($p = 0.11, 0.13, 0.14$), and divergent behavior $p = 0.17$. **Right**: mean error for different values of $r \to 0$.

Acknowledgements. The authors were supported by the Office of Science of the US Department of Energy under grant number ERKJ245.

REFERENCES

[1] C. Anfinson and F. Luk, *Linear algebraic model of algorithm-based fault tolerance*, IEEE Transactions on Computers, 37 (1988), pp. 1599–1604.

[2] A. Avižienis, *Fault-tolerance and fault-intolerance: Complementary approaches to reliable computing*, in ACM SIGPLAN Notices, vol. 10, ACM, 1975, pp. 458–464.

[3] P. Bridges, M. Hoemmen, K. Ferreira, M. Heroux, P. Soltero, and R. Brightwell, *Cooperative application/OS DRAM fault recovery*, in Proceedings of the 2011 International Conference on Parallel Processing - Volume 2, Euro-Par'11, Berlin, Heidelberg, 2012, Springer-Verlag, pp. 241–250.

[4] G. Bronevetsky and B. de Supinski, *Soft error vulnerability of iterative linear algebra methods*, in Proceedings of the 22Nd Annual International Conference on Supercomputing, ICS '08, New York, NY, USA, 2008, ACM, pp. 155–164.

[5] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, *Toward exascale resilience*, International Journal of High Performance Computing Applications, (2009).

[6] M. Casas, B. R. de Supinski, G. Bronevetsky, and M. Schulz, *Fault resilience of the algebraic multi-grid solver*, in Proceedings of the 26th ACM International Conference on Supercomputing, ICS '12, New York, NY, USA, 2012, ACM, pp. 91–100.

[7] J. Dongarra, J. Hittinger, J. Bell, L. Chacon, R. Falgout, M. Heroux, P. Hovland, E. Ng, C. Webster, and S. Wild, *Applied mathematics research for exascale computing*. US Department of Energy Report, March 2014.

[8] P. Du, A. Bouteiller, G. Bosilca, T. Herault, and J. Dongarra, *Algorithm-based fault tolerance for dense matrix factorizations*, in Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '12, New York, NY, USA, 2012, ACM, pp. 225–234.

[9] J. Elliott, M. Hoemmen, and F. Mueller, *Evaluating the impact of SDC on the GMRES iterative solver*, in Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium, IPDPS '14, Washington, DC, USA, 2014, IEEE Computer Society, pp. 1193–1202.

[10] ——, *Exploiting data representation for fault tolerance*, in Proceedings of the 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ScalA '14, Piscataway, NJ, USA, 2014, IEEE Press, pp. 9–16.

[11] J. Elliott, F. Mueller, M. Stoyanov, and C. Webster, *Quantifying the impact of single bit flips on floating point arithmetic*, Tech. Rep. ORNL/TM-2013/282, Oak Ridge National Laboratory, One Bethel Valley Road, Oak Ridge, TN, 2013.

[12] J. Elliott, F. Mueller, M. Stoyanov, and C. Webster, *Quantifying the impact of single bit flips on floating point arithmetic*, Tech. Rep. TR 2013-2, Dept. of Computer Science, North Carolina State University, Mar. 2013.

[13] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, *A survey of rollback-recovery protocols in message-passing systems*, ACM Computing Surveys (CSUR), 34 (2002), pp. 375–408.

[14] A. Geist, *What is the monster in the closet?* Invited Talk at Workshop on Architectures I: Exascale and Beyond: Gaps in Research, Gaps in our Thinking, Aug. 2011.

[15] M. Heroux and M. Hoemmen, *Fault-tolerant iterative methods via selective reliability*, Tech. Rep. SAND2011-3915, Sandia National Laboratories, 2011.

[16] K.-H. Huang and J. Abraham, *Algorithm-based fault tolerance for matrix operations*, IEEE Transactions on Computers, C-33 (1984), pp. 518–528.

[17] A. Hwang, I. Stefanovici, and B. Schroeder, *Cosmic rays don't strike twice: Understanding the nature of DRAM errors and the implications for system design*, in Architectural Support for Programming Languages and Operating Systems, 2012, pp. 111–122.

[18] J.-Y. Jou and J. Abraham, *Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures*, Proceedings of the IEEE, 74 (1986), pp. 732–741.

[19] A. Roy-Chowdhury, N. Bellas, and P. Banerjee, *Algorithm-based error-detection schemes for iterative solution of partial differential equations*, Computers, IEEE Transactions on, 45 (1996), pp. 394–407.

[20] W. Rudin, *Principles of mathematical analysis*, vol. 3, McGraw-Hill New York, 1964.

[21] P. Sao and R. Vuduc, *Self-stabilizing iterative solvers*, in Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ScalA '13, New York, NY, USA, 2013, ACM, pp. 4:1–4:8.

[22] B. Schroeder, E. Pinheiro, and W.-D. Weber, *DRAM errors in the wild: A large-scale field study*, in SIGMETRICS Conference on Measurement and Modeling of Computer Systems, 2009, pp. 193–204.

[23] J. Sloan, R. Kumar, and G. Bronevetsky, *Algorithmic approaches to low overhead fault detection for sparse linear algebra*, in Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on, IEEE, 2012, pp. 1–12.

[24] M. Snir, R. Wisniewski, J. Abraham, S. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, et al., *Addressing failures in exascale computing*, International Journal of High Performance Computing Applications, (2014), p. 1094342014522573.

[25] V. Sridharan and D. Liberty, *A study of DRAM failures in the field*, in Supercomputing, Nov. 2012.

[26] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, *Feng shui of supercomputer memory positional effects in DRAM and SRAM faults*, in High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for, IEEE, 2013, pp. 1–11.

OAK RIDGE NATIONAL LABORATORY

MANAGED BY UT-BATTELLE FOR THE DEPARTMENT OF ENERGY