ORNL REPORT

ORNL/TM-2013/283
Unlimited Release
Printed August 2013

Numerical Analysis of Fixed Point Algorithms in the Presence of Hardware Faults

M. Stoyanov, C. Webster

Prepared by
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.

DOCUMENT AVAILABILITY

Reports produced after January 1, 1996, are generally available free via the U.S. Department of Energy (DOE) Information Bridge.

Web site http://www.osti.gov/bridge

Reports produced before January 1, 1996, may be purchased by members of the public from the following source.

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
Oak Ridge, TN 37831
Telephone 703-605-6000 (1-800-553-6847)
TDD 703-487-4639
Fax 703-605-6900
E-mail info@ntis.gov
Web site http://www.ntis.gov/support/ordernowabout.htm

Reports are available to DOE employees, DOE contractors, Energy Technology Data Exchange (ETDE) representatives, and International Nuclear Information System (INIS) representatives from the following source.

Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831
Telephone 865-576-8401
Fax 865-576-5728
E-mail reports@osti.gov
Web site http://www.osti.gov/contact.html

NOTICE

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

ORNL/TM-2013/283

Computer Science and Mathematics Division

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

M. Stoyanov * C. G. Webster †

Date Published: August 2013

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831-6283
managed by
UT-BATTELLE, LLC
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

*Computer Science and Mathematics Division, Oak Ridge National Laboratory, One Bethel Valley Road, P.O. Box 2008, MS-6367, Oak Ridge, TN 37831-6164 (stoyanovmk@ornl.gov).

†Computer Science and Mathematics Division, Oak Ridge National Laboratory, One Bethel Valley Road, P.O. Box 2008, MS-6164, Oak Ridge, TN 37831-6164 (webstercg@ornl.gov)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The computing power of modern hardware has been increasing exponentially for several decades and this trend is expected to persist. However, the increase of computational throughput has also led to a decrease in reliability to the point where hardware faults are now the norm rather than the exception. Historically, numerical algorithms have been developed under the assumption that all of the associated basic algebraic operations can be computed to within round-off error. This assumption has made modern solvers very susceptible to a particular class of silent errors that may modify the result of a numerical operation without any external indication (e.g. bit-flip in the CPU cache). Those silent faults, introduce additional error that cannot be detected or corrected via conventional software resilience techniques (e.g. restart or checksums). Numerical solvers compute approximations that should not deviate from an analytic solution by more than a prescribed error tolerance, however, in the presence of silent hardware faults, even if the computer code successfully completes execution and returns a result, the produced approximation may fall outside of the desired accuracy.

We propose a new analytic approach for improving algorithms' resilience with respect to silent faults encountered in modern extreme-scale supercomputers. We extend the current framework of numerical analysis by removing the assumption that all arithmetic operations can be computed accurately. We introduce the concept of "hardware error" added to our numerical approximations by potentially unreliable hardware. We also propose an fault model that is agnostic with respect to the underlying hardware architecture. Using rigorous analysis, we develop new algorithms that minimize the propagation of hardware error and therefore guarantee convergence even when faults are encountered.

Iterative solvers are associated with the bulk of computations in many extreme scale applications and hence we first focus our attention to those methods. In particular, we consider the class of fixed point solvers. Utilizing existing resilience techniques, we split out algorithm into two components that we label "safe" and "fast". The "safe" section is assumed to be executed without hardware errors (or with negligible probability of failure) and the fast section is assumed to experience failure. We derive a test condition that would be executed in safe mode, so that errors introduced in the fast computations with large magnitude will be detected and rejected, while small errors will be automatically corrected. We prove convergence of the enhanced algorithm and provide a numerical example where we obtain reliable approximations to the solution even if we execute the algorithm with a very high rate of simulated hardware faults.

ACKNOWLEDGEMENTS

The ORNL is operated by UT-Battelle, LLC, for the United States Department of Energy under Contract DE-AC05-00OR22725.

1 Introduction

In the past few decades we have witnessed the rise of computer simulations as the third pillar of science, next to theory and experiments. The advances in the field of computational science have created and ever increasing demand for more and more powerful supercomputers. To answer the challenge, modern extreme scale computers are build to push the hardware to its limits. Fabrication size shrinks exponentially and constraints on the power consumption require that circuits operate at near critical voltage, thus even a small voltage fluctuation may have an adverse effect on operation. In addition, the ever increasing number of components offers more opportunity for hardware failure. Thus, the increase of computational power has led to a decrease in reliability. Hardware failure is now the norm rather than the exception and in the near future it is expected that the mean failure time of a supercomputer component to be of the order of an application runtime [5, 9, 10, 12, 14].

The problem of hardware faults is not a new phenomena and extensive measures have been taken to ensure that the machines operate reliably. Memory comes with error detection and correction [14, 15] and CRC checksums are used to guard against data corruption in network traffic. However, processing cores remain largely unprotected [5, 8] and adding redundancy to all chips in a cluster is prohibitive in terms of cost and energy consumption. Faults in a single component are rare, however, in an extreme scale machine, small chances of failure are multiplied to where they become a common occurrence. A small failure rate of one in a million becomes a near certainty in a machine of millions of components. Hardware level error detection and correction is indispensable, however, insufficient to solve the problem of hardware faults.

Software based resilience techniques complement the hardware approach. Software has to address the three possible outcomes of a code execution:

- The system crashes and no result is returned. In this case, we have no choice but to either rerun the code or preferably do a full or partial recovery from an earlier saved state. Code that executes on multiple nodes of a supercomputer for many hours needs to be able to recover from a crash in a one or more nodes.
- The code computes an invalid result. An invalid result is as good as no result and the code needs to be rerun or restarted from a previous saved check-point. The most obvious examples of invalid result are inf and nan. Other example include mathematically or physically impossible quantities, such as negative areas, however, those may be due to numerical stability or missing physics in the model.
- Execution completes and a result it returned that is within expected ranges. However, even if the result appears to be correct, it is still possible that the machine encountered an internal error (i.e. "silent fault"). Traditionally, numerical algorithms have been developed under the assumption that all basic algebraic operations can be computed reliably according to machine specifications, this assumption makes modern numerical code particularly vulnerable to silent faults.

The main focus of our work is the third type of errors that provide a unique set of challenges.

Consider the following example, suppose two variables with values 2 (binary 010) and 4 (binary 100) need to be added. First they are moved from the protected DRAM into the unprotected CPU cache. Then a voltage fluctuation occurs, causing the cache to fail and flip one bit making the second value 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 that

$$2 + 4 = 8.$$

If either 2 or 4 gets corrupted in main memory, we can run a posterior checksum on the data structures and identify that a fault had occurred. However, in the above example both values 2 and 4 remain intact, but the answer is incorrect. In the recent work of Hoemmen and Heroux [6] those type of errors are classified as "soft-transient" faults. In order to emphasize the main challenge associated with those errors we label them as "silent" faults.

One of the most common software resilience technique is the checkpoint/restart [1, 11, 16, 17]. Virtually all large scale software comes with capability to periodically save the execution state and revert back to an earlier checkpoint if an error is detected. This approach can completely negate any and all effects of a hardware fault (other than the added run time), however, checkpoint/restart relies heavily on a mechanism to detect the failure and hence it is unsuited to handle silent faults.

Checksums are a powerful tool that guards against data corruption in network communication as well as data storage in RAM. A well designed checksum scheme can detect and even correct errors as small as single bit-flips, however, not all memory storage can be protected in this way.

Checksum schemes can even be used to predict the outcome of a computation and thus identify a silent fault [7]. However, the approach is restricted to linear operations (i.e. matrix vector solve) and cannot be trivially generalized to non-linear problems. Furthermore, in modern heterogeneous computing environment different types of computing units (e.g. CPU or GPU) may use slightly different rounding precision. The result of floating point operations cannot be predicted bit-by-bit and hence checksumming floating point data would create a very large number of false-positive errors.

Redundant computation is yet another approach to guard against hardware failure that is commonly applied for machines working in unfriendly environment (e.g. higher radiation levels at aircraft altitudes). Using redundancy one performs the same computations two or more times to ensure consistent result. However, redundancy cannot be used on an entire extreme scale application [2, 4], at best redundancy can be used on small sections of the code.

Another viable technique is to use post-processing to verify the correctness of the computed result. While this approach is viable, with the increasing rate of hardware faults we can no longer have the expectation that a large scale simulation will not encounter silent faults. Therefore, identifying a problem at the very end of the execution is too late; the only way to correct a faulty result would be to redo the simulation, which is likely to encounter yet another error. Any silent fault identification and correction has to be done on-the-fly.

Hoemmen and Heroux [6] propose an approach to split the program execution into two modes,

"safe" and "fast" modes (called respectively "host" and "guest" modes in the paper). The computations in "safe mode" are assumed to be performed reliably (i.e. zero or near zero 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 faults. Hoemmen and Heroux propose a modified version of the restarted GMRES algorithm, where a convergence test is performed in safe mode between every two inner GMRES iterations. The convergence test rejects the next iterate unless it reduces the residual error, thus large errors are rejected and only small ones are accepted by the algorithm.

The approach of selective reliability has a lot of potential in dealing with silent faults, however, it also presents a number of new challenges. It is observed that the convergence of the algorithm deteriorates as the frequency of the faults increased, due to the additional work needed to correct a 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 vector, the statistics of which remain to be quantified.

Traditionally numerical algorithms have been developed under the assumption that the underlying algebraic operations (e.g. matrix vector product) can be carried out accurately. As the reliability of supercomputers decreases, we can no longer afford to make this assumptions. Numerical algorithms compute approximations to the solutions of mathematical problems and thus every algorithm has associated numerical error. Silent faults introduce additional error into the approximation and hence numerical error and hardware error have to be addressed together in the same context. *The problem of silent hardware faults has to be answered by numerical analysis.*

In this work, we establish a framework of numerical analysis that is free from the assumption that computations can always be performed accurately. We assume that occasionally the result of computations will be randomly perturbed. We want our analysis to be agnostic to hardware architecture and hence we make no assumptions on the structure of the perturbation. Similar to the approach of Hoemmen and Heroux, we want to reject error of large magnitude and only allow the introduction of faults that are indistinguishable from standard numerical error. In the new framework, we provide rigorous error bound for the hardware error and we prove convergence of the algorithms even if they are implemented on unreliable hardware.

As a demonstration of our approach, we present the analysis of the widely used Fixed Point family of iterative solvers. We demonstrate how the resilience of the methods can be dramatically improved and we prove convergence of the methods with respect to hardware error. This is the first step towards building a suite of resilient numerical algorithms that provide reliable results even in the presence of silent hardware faults.

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 define "hardware error" and establish the meaning of "convergence". In addition, we provide a fault model that is agnostic with respect to hardware architecture or specific software implementation.

2.1 Hardware Error

Numerical algorithms approximate the solutions to mathematical problems and one of the main focuses of analysis is the problem of error and convergence. Here are some examples:

- When differential equations are approximated by Finite Element or Finite Difference schemes, there is discretization error associated with the density of the mesh. A discretization scheme is convergent if the error goes to zero as the mesh density is increased (i.e. $\Delta x \to 0$).
- Iterative solvers for linear and non-linear equations create a sequences of approximations that converge to the exact solution, then the sequence is terminated after a finite number of steps. The difference between the final step and the exact solution is the iteration error, which can be reduced by additional iterations.

In all cases, analytic convergence is defined as: for every $\epsilon > 0$, there is a finite amount of work (e.g. solving a problem on a denser mesh or doing more iterations) that will drive the numerical error to less than ϵ.

Following the same analogy, we present the following definition:

Definition 1 *Hardware Error*

Error introduced into numerical computations by silent hardware malfunction.

For the remaining of the paper we will denote the hardware error as e^h, which is fundamentally a random variable. Hence convergence needs to be defined in statistical terms. In this work we consider the expected value and variance of e^h, however, higher order moments follow analogously.

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 that will make $E[e^h] < \epsilon$ and $V[e^h] < \epsilon^2$, where $E[e^h]$ and $V[e^h]$ denote the expected value and variance of e^h.

Note that hardware error is not to be confused with round off error as there are some key differences:

- When numbers are represented in finite precision, every operation introduces rounding error. Hardware error on the other hand is a relatively rare phenomena and will originate only in a small number of random operations.
- Finite precision computations are usually performed according to standards (e.g. IEEE-754 floating point). If standards are followed exactly, then the rounding error is a deterministic quantity, while the hardware error is due to random behavior that deviates from the standards.
- Round-off error is a function of the precision and conditioning of the the numerical method, usually it is a small and bounded quantity. On the other hand, hardware error may range from machine precision to the largest representable number [3].

The convergence may be contingent upon the properties of the hardware, such as rate of failure or expected magnitude of the failure, therefore a method may be only conditionally convergent and the relative resilience of two methods can be compared based on the restrictions or additional assumption that they require (i.e. an algorithm may be convergent on one machine, but not another).

2.2 Error Model

In order to analyze the effects of hardware error on a numerical algorithm, we need to introduce an analytic error model. Utilizing the fast/safe mode approach of Hoemmen and Heroux, we split the algorithm into two parts. In their work, Hoemmen and Heroux perform a numerical simulation by assigning a pattern of random bit-flips, however, this approach assumes specific floating point representation as well as specific algorithm implementation. We propose an analytic approach, where we assume that at each step, for each operation performed in fast mode, the result may be perturbed by a random quantity. For example, if a matrix vector product results in x, then there is non-zero probability of computing $x + \tilde{x}$, where the hardware error is $e^h = \|\tilde{x}\|$.

We want our approach to be agnostic with respect to the specifics of the machine, hence we make as little assumption on the structure of $\tilde{x}$ as possible. In order to keep the analysis general, we cannot assume a distribution for the magnitude $\|\tilde{x}\|$. Our earlier work [3] on the analysis of the most popular IEEE-754 floating point standard, shows that the statistics of the error introduced in a dot product by a single bit-flip is strongly influenced by the position of the flipped bit (i.e. the least or most significant bit) as well as the scaling of the corrupt vector entry. Therefore, statistics of $\|\tilde{x}\|$ are often times problem dependent. Only some generic statistical properties, such as the frequency of silent faults, may be assumed for the purpose of convergence analysis.

3 Analytic Example: Fixed Point Methods

Iterative linear and non-linear solvers are the methods of choice for large scale applications. Such methods generate a sequence of approximate solutions that converge to the exact solution and the sequence is terminated after a convergence criteria has been met. One of the most widely used class of iterative methods is the family of fixed point solvers, which includes the linear Jacobi and Gauss-Siedel methods as well as the non-linear Newton method. Those posses some natural resilience properties and hence they make for a good illustrative example of our approach.

3.1 Fixed Point Methods

Fixed point methods are based on the famous Fixed Point Theorem:

Theorem 1 *Fixed Point Theorem*

Suppose $\Gamma \subset \mathbb{R}^N$ is complete in some norm $\|\cdot\|$ and let $G : \Gamma \to \Gamma$ be a contraction operator, i.e. there is a constant $r < 1$ so that

$$\|G(v) - G(w)\| \leq r\|v - w\|, \qquad \textit{for all } v, w \in \Gamma.$$

Then, there is a unique fixed point $v^* \in \Gamma$ so that $G(v^*) = v^*$. Furthermore, for any $v_0 \in \Gamma$ the sequence $\{v_i\}_{i=0}^{\infty}$ defined by $v_{i+1} = G(v_i)$ converges to v^* (i.e. $\|v_i - v^*\| \to 0$) and the error $\|v_i - v^*\|$ is bounded by

$$\|v_i - v^*\| \leq r^i \|v_0 - v^*\|.$$

A proof of the theorem can be found in [13].

In particular, consider the system of linear equations given by

$$Ax = b, \tag{3.1}$$

where $A \in \mathbb{R}^{N\times N}$ is a given matrix, $b \in \mathbb{R}^N$ is a given right hand side and we are interested in the solution $x^* \in \mathbb{R}^N$ that satisfies (3.1). Two of the most popular fixed point methods for linear equations are the Jacobi and Gauss-Seidel methods. The contraction maps and the corresponding rates are given as

- **Jacobi Method**

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

  where $D = diag(A)$ (i.e. D is a diagonal matrix with the entries of A) and $S = A - D$.

- **Gauss-Seidel Method**

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

  where $D = lower(A)$ (i.e. L is a lower triangular matrix of the entries of A) and $U = A - L$.

The solution x^* to (3.1) is a fixed point to both $G_J(x)$ and $G_{GS}(x)$, therefore, if $r_J < 1$ or $r_{GS} < 1$, according to Theorem 1, the corresponding fixed point iteration will converge to x^* for all $x_0 \in \mathbb{R}^N$ (i.e. $\Gamma \equiv \mathbb{R}^N$).

The fixed point algorithms can be summarized in the following way

Algorithm 1 *Fixed Point Iteration*
Given G, b, initial x_0, tolerance ϵ and k_{min}
Let $k = 0$
repeat
$x_{k+1} = G(x_k)$
$e_k = \|x_{k+1} - x_k\|$
$k = k + 1$
until $(k > k_{min})\, and\, (e_k < \epsilon)$

The difference e_k between two successive iterates is a the most common indicator for convergence, when the iteration reaches asymptotic mode. However, this condition may cause premature termination in the first few iterations, hence we require that at least k_{min} number of iterations are taken[1].

3.2 Hardware Error Analysis

The most computationally expensive part of Algorithm 1 is computing the next iterate

$$x_{k+1} = G(x_k).$$

Suppose that at step k we encounter a silent hardware fault, which perturbs the result, i.e. instead of x_{k+1} we compute

$$\tilde{x}_{k+1} = G(x_k) + \tilde{x}.$$

So long as $\tilde{x}_{k+1} \in \Gamma$, the algorithm will generate a new sequence $y_0 = \tilde{x}_{k+1}$ with $y_{i+1} = G(y_i)$ so that $y_i \to x^*$. The convergence for the new sequence is bounded by

$$\|y_i - x^*\| \quad \leq \quad r^i\|y_0 - x^*\| \quad \leq \quad r^i\|x_{k+1} - x^*\| + r^i\|\tilde{x}\| \quad \leq \quad r^i r^{k+1}\|x_0 - x^*\| + r^i\|\tilde{x}\|.$$

The first term of the estimate $r^{i+k+1}\|x_0 + x^*\|$ is identical to the one associated with the error free iteration. Therefore, the hardware error associated with the Fixed Point Iteration that is caused by a single hardware failure is given by

$$e^h = r^i\|\tilde{x}\|,$$

where i is the number of iterations after the failure and $\|\tilde{x}\|$ is the magnitude of the introduced error.

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

Analogously, if we encounter j hardware faults at iterations $i_1, i_2, \cdots, i_j$ with perturbations $\tilde{x}_1, \tilde{x}_2, \cdots \tilde{x}_j$, then we have the error bound

$$\|x_k - x^*\| \leq r^k \|x_0 + x^*\| + \sum_{l=1}^{j} r^{k-i_l} \|\tilde{x}_l\|.$$

Thus the hardware error associated with multiple silent faults is $e^h = \sum_{l=1}^{j} r^{k-i_l} \|\tilde{x}_l\|$.

Next we ask whether or not the fixed point iteration is convergent with respect to hardware error. The terms $r^{k-i_l} \to 0$ as $k \to \infty$, however, so long as the probability of fault at the $k-th$ step is bounded away from zero, the statistics of r^{k-i_l} will not converge. The hardware error is also very sensitive to the magnitude of $\|\tilde{x}\|$, a single fault of large magnitude may require too many additional iterations to converge. Our earlier work shows how rescaling the entries of A can be used to minimize the statistics of $\|\tilde{x}\|$ [3], however, not all problems can be efficiently rescaled and even then the expectation of $\|\tilde{x}\|$ cannot be driven to zero. Therefore, the fixed point iteration is not convergent with respect to silent faults in the evaluation of $G(x_k)$.

Furthermore, the above analysis considers only silent faults in the evaluation of $G(x_k)$, however, the evaluation of e_k is also susceptible to hardware error. This may lead to an early exit of the iteration loop and false convergence before the criteria $e_k < \epsilon$ has been met. Therefore, the fixed point iteration is not convergent with respect to faults in computing e_k.

3.3 Resilience Enhancement Techniques and Convergence

First we note that evaluation of e_k as well as any conditional branching in the algorithm has to be evaluated in save mode. This can be implemented on dedicated hardware or via redundancy [6]. We can also note that the work associated with $\|x_{k+1} - x_k\|$ is significantly less than the evaluation of $G(x_k)$, in fact, in many applications the difference may be several orders of magnitude. Hence, if a silent fault is encountered, it is far more likely to happen during evaluations of $G(x_k)$ and this alone may be sufficient to reduce the risk associated with computing e_k in fast mode to an acceptable level.

The evaluation of $G(x_k)$ is the most computationally expensive step of the algorithm and hence it must be executed in fast mode. Since fast mode is susceptible to silent faults, we need a way to discriminate against large hardware error. Observe that due to the contraction properties of $G(x)$ we have that

$$e_k = \|x_{k+1} - x_k\| = \|G(x_k) - G(x_{k-1})\| \leq r\|x_k - x_{k-1}\| = re_{k-1}.$$

We expect to observe this monotonic behavior and it is a condition that is very easy to verify at each step. If the condition fails (i.e. we observe other than monotonic convergence), then we reject the next iterate x_{k+1} and recompute $G(x_k)$. In practice, the value of r may be unknown, hence we can use an upper estimate $\beta \in (r, 1]$. If such estimate is not available, we can start the iteration with $\beta = 1$ and consider $e_{k+1}/e_k < r$, then from several e_k we can estimate r. In addition, we also

need to chose a value of $e_{-1} = \|x_0 - x^*\|$. We could use the upper bound $e_{-1} = \|x_0\| + \|x^*\|$ or $e_{-1} = \|x_0\| + \gamma$, where we select $\gamma > \|x^*\|$ as an estimate of $\|x^*\|$.

We propose the following improved algorithm, where all steps except $x_{k+1} = G(x_k)$ are executed in safe mode:

Algorithm 2 *Improved Fixed Point Iteration*
Given G, initial x_0, $\beta \in (r, 1]$, $\gamma > \|x^*\|$, *tolerance* ϵ *and* k_{min}
Let $e_{-1} = \|x_0\| + \gamma$
$k = 0$
repeat
$x_{k+1} = G(x_k)$
$e_k = \|x_{k+1} - x_k\|$
if $e_k < \beta e_{k-1}$ ***then***
Accept 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* $(e_k < \epsilon)$

The evaluation of $G(x_k)$ is the only step of the algorithm susceptible to silent faults. We note that it is possible for a fault in $G(x_k)$ to create a false positive rejection and therefore stagnation, however, we address this issues at a late point and we first focus on the effects of the faults on the iterates x_k.

Analogous to our earlier analysis, we replace x_{k+1} by

$$\tilde{x}_{k+1} = G(x_k) + \tilde{x}.$$

First consider the case when $k = 0$. The faulty initial iterate would be rejected unless

$$\|x_1 + \tilde{x} - x_0\| < \beta \left(\|x_0\| + \gamma\right).$$

If we consider a new iteration $y_0 = \tilde{x}_1$, we have the bound

$$\begin{aligned} \|y_i - x^*\| &\leq r^i \|\tilde{x}_1 - x^*\| \\ &\leq r^i \|\tilde{x}_1 - x_0 + x_0 - x^*\| \\ &\leq r^i \left(\beta\|x_0\| + \beta\gamma + \|x_0 - x^*\|\right). \end{aligned}$$

Note that the error bound for the new iteration is independent from the magnitude $\|\tilde{x}\|$ and it depends only on the parameter β, γ and the initial iterate x_0. Therefore, the improved algorithms is convergent with respect to a fault in the first iteration.

If $k > 0$, the modified method will reject $\tilde{x}_k$ unless

$$\|x_{k+1} + \tilde{x} - x_k\| < \beta\|x_k - x_{k-1}\|.$$

Assuming that $\tilde{x}_{k+1}$ is the starting point of a new iteration $\{y_i\}$, the error at step y_i will be bounded

$$\begin{aligned}
\|y_i - x^*\| &\leq r^i \|\tilde{x}_{k+1} - x^*\| \\
&\leq r^i \|\tilde{x}_{k+1} - x_k + x_k - x^*\| \\
&\leq r^i \beta \|x_k - x_{k-1}\| + r^i \|x_k - x^*\| \\
&\leq r^i \beta r^{k-1} \|x_0 - x^*\| + r^i r^k \|x_0 - x^*\| \\
&= r^{i+k}(1 + \beta/r)\|x_0 - x^*\| \\
&= r^{i+k+1}\|x_0 - x^*\| + r^{i+k}\left(1 + \frac{\beta}{r} - r\right)\|x_0 - x^*\|.
\end{aligned}$$

Note that $i+k+1$ is the total number of iterations before and after the fault and hence, the hardware error associated with the improved iteration is bounded by

$$e^h \leq r^{i+k-1}\left(1 + \frac{\beta}{r} - r\right)\|x_0 - x^*\|.$$

The bound on the hardware error is independent from either the iteration on which the fault was encountered or the magnitude $\|\tilde{x}\|$ of the introduced perturbation. Therefore, the improved algorithm is convergent with respect to one hardware fault.

We can extend the result to multiple hardware faults. The total numerical error is bounded

$$\|x_k - x^*\| \leq r^{k-j}(1 + \beta/r)^j \|x_0 - x^*\|,$$

where j is the number of accepted faults. The statistics of the error depend only on k, j and the parameter β and x_0, where only j is a random variable.

Consider the statistics of j. Let $P(j|k)$ denote the probability of accepting j hardware faults in k iterations. Then the statistics of the hardware error can be bounded by

$$\begin{aligned}
E\left[\|x_k - x^*\|\right] &\leq \|x_0 - x^*\| \sum_{j=0}^{k} r^{k-j}(1 + \beta/r)^j P(j|k), \\
V\left[\|x_k - x^*\|\right] &\leq \|x_0 - x^*\|^2 \sum_{j=0}^{k} r^{2k-2j}(1 + \beta/r)^{2j} P(j|k).
\end{aligned}$$

Therefore, the improved iteration would be convergent if

$$\begin{aligned}
\lim_{k\to\infty} \sum_{j=0}^{k} r^{k-j}(1 + \beta/r)^j P(j|k) = 0, \\
\lim_{k\to\infty} \sum_{j=0}^{k} r^{2k-2j}(1 + \beta/r)^{2j} P(j|k) = 0,
\end{aligned}$$

and if we do not encounter any false positive rejections.

3.4 Guarding Against False Positive Rejections

Consider Algorithm 2 and suppose that at step k the result of $G(x_k)$ is perturbed so that

$$\tilde{x_k} = G(x_k) + \tilde{x} \approx x_k.$$

Even though e_k is computed in safe mode, the perturbation $\tilde{x}$ will make e_k very small, which can cause either an early termination or a fault-free $k+2$ iterate would be incorrectly rejected and the algorithm would stagnate. A number of techniques can be applied to avoid such problems.

To avoid stagnation use can do one of the following:

- We could apply a redundancy technique, we could keep track of the rejected steps and note if two or more successive x_{k+1} are rejected with similar e_k. If the probability of multiple consecutive silent faults is negligible, then we have a strong indication of stagnation. In this case, we may overwrite the accept/reject criteria and accept the next iterate.

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

- Yet another redundancy approach may be to test e_k against several of the past $e_{k-1}, e_{k-2}, \cdots$. That is

  if $(e_k < \beta e_{k-1}) or (e_k < \beta^2 e_{k-2}) or (e_k < \beta^3 e_{k-3})$ **then**
  Accept x_{k+1}
  else
  Reject x_{k+1}
  end if

- In addition to having the upper bound $e_k \leq \beta e_{k-1}$, we can also introduce a lower bound $\alpha e_{k-1} \leq e_k \leq \beta e_{k-1}$. We observe that $e_k \approx r e_{k-1}$ and chose α so that

  $$\frac{r^2}{\beta} < \alpha < r.$$

  Assume that we encounter a silent fault at step k, which results in e_k being smaller but bounded from below $e_k > \alpha e_{k-1}$. Suppose that the next iterate is fault free (i.e. $e_{k+1} < r e_k$), then

  $$e_{k+1} \leq r^2 e_{k-1} = \beta \frac{r^2}{\beta} e_{k-1} \leq \beta \alpha e_{k-1} < \beta e_k,$$

  and therefore the fault free iterate e_{k+1} will be accepted (i.e. there is no stagnation).

To avoid early termination can do one of the following:

- We could use post-processing in safe mode on the final iterate to verify that it satisfies the equation of interest. If the test fails, we can restart the fixed point iteration with the last iterate as the new initial x_0. This is a very safe, but potentially expensive approach.

- We could require that two or more of the last computed e_k simultaneously satisfy the convergence criteria $e_k < \epsilon$, i.e. we can replace the test by

  repeat

  $\cdots$

  until $(e_k < \epsilon)\, and\, (e_{k-1} < \epsilon/r)\, and\, (e_{k-2} < \epsilon/r^2) \cdots$

- If we use the accept/reject test $\alpha e_{k-1} \leq e_k \leq \beta e_{k-1}$, it will naturally guard against early termination as it will bound e_k from below.

All of above techniques are viable under different circumstance. We select the two that in the authors' opinion are most robust and most widely applicable. We present the resilient fixed point algorithm.

Algorithm 3 *Resilient Fixed Point Iteration*

Given G, initial x_0, $\beta \in (r, 1]$, $\alpha \in (r^2/\beta, r)$, $\gamma > \|x^*\|$, *tolerance* ϵ *and* k_{min}
Let $e_{-1} = \|x_0\| + \gamma$
$k = 0$
repeat
  $x_{k+1} = G(x_k)$
  $e_k = \|x_{k+1} - x_k\|$
  if $\alpha e_{k-1} \leq e_k \leq \beta e_{k-1}$ ***then***
    Accept 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* $(e_k < \epsilon)$

Where α, β and γ are tuning parameters that affect the magnitude of the hardware error.

3.5 Convergence Rate

Consider the bounds on the statistics

$$\begin{aligned} E\left[\|x_k - x^*\|\right] &\leq \|x_0 - x^*\| \sum_{j=0}^{k} r^{k-j}(1+\beta/r)^j P(j|k), \\ V\left[\|x_k - x^*\|\right] &\leq \|x_0 - x^*\|^2 \sum_{j=0}^{k} r^{2k-2j}(1+\beta/r)^{2j} P(j|k). \end{aligned}$$

In order to discuss the convergence rate, we need to assume a model for $P(j|k)$. Since the convergence depends only on the number of accepted hardware faults in k iterations, we may derive a model of $P(j|k)$ based solely on the rate of faults. Let every iteration has a probability of fault p that is independent from other iteration or from k. Then $P(j|k)$ has the binomial distribution

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

Substituting into the bounds for the statistics

$$\begin{aligned} E\left[\|x_k - x^*\|\right] &\leq \|x_0 - x^*\| \sum_{j=0}^{k} \frac{k!}{j!(k-j)!} r^{k-j}(1+\beta/r)^j p^j (1-p)^{k-j}, \\ V\left[\|x_k - x^*\|\right] &\leq \|x_0 - x^*\|^2 \sum_{j=0}^{k} \frac{k!}{j!(k-j)!} r^{2k-2j}(1+\beta/r)^{2j} p^j (1-p)^{k-j}. \end{aligned}$$

Combining like terms yields

$$\begin{aligned} E\left[\|x_k - x^*\|\right] &\leq \|x_0 - x^*\| \sum_{j=0}^{k} \frac{k!}{j!(k-j)!} (r(1-p))^{k-j}((1+\beta/r)p)^j, \\ V\left[\|x_k - x^*\|\right] &\leq \|x_0 - x^*\|^2 \sum_{j=0}^{k} \frac{k!}{j!(k-j)!} (r^2(1-p))^{k-j}((1+\beta/r)^2 p)^j. \end{aligned}$$

Using the polynomial expansion formula

$$\begin{aligned} E\left[\|x_k - x^*\|\right] &\leq \|x_0 - x^*\| \left(r(1-p) + (1+\beta/r)p\right)^k, \\ V\left[\|x_k - x^*\|\right] &\leq \|x_0 - x^*\|^2 \left(r^2(1-p) + (1+\beta/r)^2 p\right)^k. \end{aligned} \tag{3.2}$$

The right hand sides will converge to zero if

$$p < \frac{1-r^2}{(1+\beta/r)^2 - r^2},$$

which gives us a relationship between the contraction properties of $G(x)$ and the reliability of the hardware needed for the resilient fixed point iteration to converge. In particular we have been able to quantity the following:

- For perfectly reliable hardware (i.e. $p = 0$),

$$(r(1-p) + (1+\beta/r)p)^k = r^k,$$

  and we recover the rate of convergence of the classical fixed point iteration.

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

- As $r \to 1$ and the conditioning of the fixed point method deteriorates, the upper bound forces $p \to 0$, which signifies that we need more reliable hardware to solve a problem with worse conditioning.

- As $r \to 0$, p is bounded from above by 0.5, which means that even in the best scenario, we need hardware that can compute at least every second iteration without any soft faults.

4 Numerical Example

In this section, our goal is to numerically verify the theoretical results about the regular and resilient Fixed Point Iterations. We consider the particular example of 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 partial differential equation

$$\begin{aligned} \frac{d}{dt}u(t,x,y) &= -\frac{\partial^2}{\partial x^2}u(t,x,y) - \frac{\partial^2}{\partial y^2}u(t,x,y), \qquad (x,y) \in \Omega, t > 0, \\ u(t,x,y)|_{\partial\Omega} &= 0, \\ u(0,x,y) &= xy(x-1)(y-1). \end{aligned}$$

We seek a numerical approximation to the solution $u(t,x,y)$. We discretize the problem in Ω using Finite Difference scheme with uniformly distributed nodes. Define

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

and approximate

$$u(t,x_i,y_j) \approx u_{i,j}(t), \qquad u_{i,j}(0) = x_i y_j (x_i - 1)(y_j - 1).$$

We discretize the diffusion operator as

$$-\frac{\partial^2}{\partial x^2}u_{i,j}(t) - \frac{\partial^2}{\partial y^2}u_{i,j}(t) \approx \frac{u_{i-1,j}(t) - 2u_{i,j}(t) + u_{i+1,j}(t)}{\Delta x^2} + \frac{u_{i,j-1}(t) - 2u_{i,j}(t) + u_{i,j+1}(t)}{\Delta y^2},$$

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

$$\dot{v}(t) = Lv(t),$$

where the k-th component of $v(t)$ is associated with $u_{i,j}(t)$ by $v_{(i-1)n+j}(t) = u_{i,j}(t)$ and L is the matrix representation of the discretized operator.

We evolve the system in time using backward Euler method. We select a time step Δt and approximate

$$v(t + \Delta t) \approx (I - \Delta t L)^{-1} v(t).$$

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 on the first linear system (i.e. $t = 0$). We take $n = 100$ and $\Delta t = 10^{-4}$, which results in the system of linear equations

$$Ax = b,$$

where $A = I - \Delta t L$, $b = v(0)$ and $x \approx v(\Delta t)$. We use the Jacobi fixed point method with $x_0 = b$ and $\epsilon = 10^{-8}$. The spectral radius of the iteration matrix is $r \approx 0.8028$.

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 performed all the test twice to ensure consistent result. Since the hardware failure rate on our machine is too low, we need to artificially introduce faults into the computations. At every iteration, we poll a pseudo random number using MATLAB's *rand()* function, which returns a uniformly distributed number in the range $(0, 1)$. If the random number if smaller than a specified threshold 0.1, we introduce a perturbation to the evaluation of x_{k+1}. The perturbation $\tilde{x}$ is a pseudo-random vector added to x_{k+1}. To ensure that $\tilde{x}$ has magnitude that spans a wide range of possible perturbations we use the following MATLAB code:

```
function [ x_tildae ] = getXTildae( N )

    err = randn( N, 1 );
    if ( rand(1,1) > 2/3 )
        if ( rand(1,1) > 1/3 )
            err = 100 *rand( 1, 1 ) * err / norm( err );
        else
            err = 10000 *rand( 1, 1 ) * err / norm( err );
        end
    elseif ( rand(1,1) > 2/3 )
        err = 0.01 *rand( 1, 1 ) * err / norm( err );
    elseif ( rand(1,1) > 2/3 )
        err = 0.001 *rand( 1, 1 ) * err / norm( err );
    elseif ( rand(1,1) > 1/3 )
        err = 1.E-6 *rand( 1, 1 ) * err / norm( err );
    else
        err = 1.E-8 *rand( 1, 1 ) * err / norm( err );
    end
    x_tildae = err;
end
```

We set the rate of silent faults to be one in ten, which is extremely high compared to real world hardware. However, for the purpose of this study, we want to observe the effects of the simulated faults on the algorithms and thus we need such faults to be common enough. We make only one fault free simulation for comparison purposes, and in all other cases, we want to encounter at least one fault.

Fist we apply Algorithm 1 without any hardware faults. On Figure 1 we observe that the method converges in 83 iterations as well as the linear convergence rate. However, if we assume soft fault rate of one in ten iterations, the classic fixed point method fails to converge. The results from one realization of the faulty algorithm is given on Figure 2. In order to study the statistics of the error, we use Monte Carlo sampling. We take 1000 realization of Algorithm 1 executed with

simulated faults and average the resulting error as a function of the iteration. On Figure 3 we see that the expected value of the error does not converge for the first 500 iterations. It is possible for a realization to converge (e.g. there is 0.9^{83} probability that no faults would be encountered), however, we should have no expectation of convergence.

Next we perform the same set of test using the resilient fixed point iteration of Algorithm 3. We use $\alpha = 0.7$, $\beta = 1$ and $\gamma = 1$. The result of one realization is shown on Figure 4. The introduction of faults deteriorates the convergence rate, however, the resilient method does converges in 90 iterations. Furthermore, on Figure 5, we observe convergence for both the expected value and standard deviation.

The theoretical bounds on the statistics (3.2) depends on p, which is the rate of accepted soft faults which is smaller than the total rate of faults. Using Monte Carlo sampling again, we estimate the rate of accepted faults as $p \approx 0.04$. On Figure 6, we compare the observed expected value for the error and the theoretical error bound. The error bound assumes the worst case scenario and therefore it is an overestimate. Nevertheless, we observe the expected linear convergence rate, thus demonstrating that the resilient fixed point iteration is indeed convergent even in the presence of a very high rate of faults.

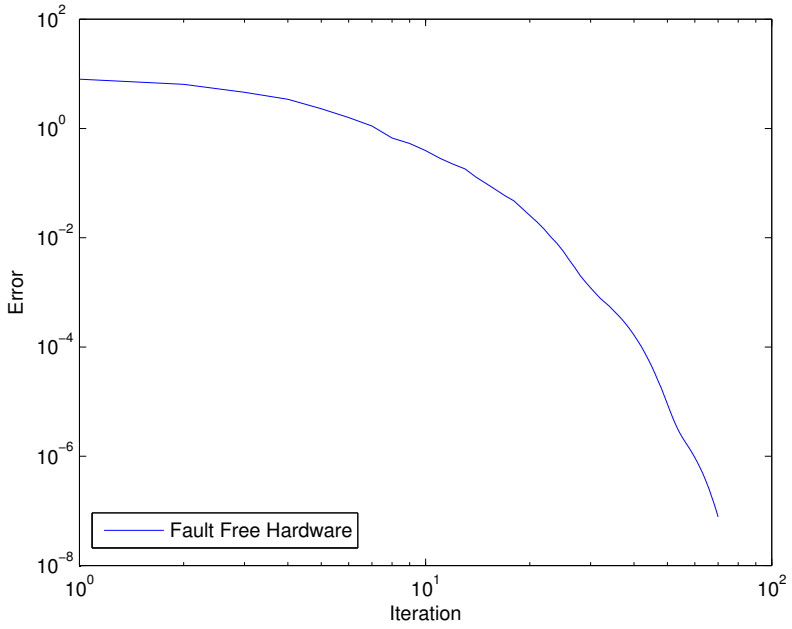

Figure 1: Convergence of the Jacobi method when executed without simulated hardware faults. We observe the expected linear convergence and the method takes 72 iterations to converge.

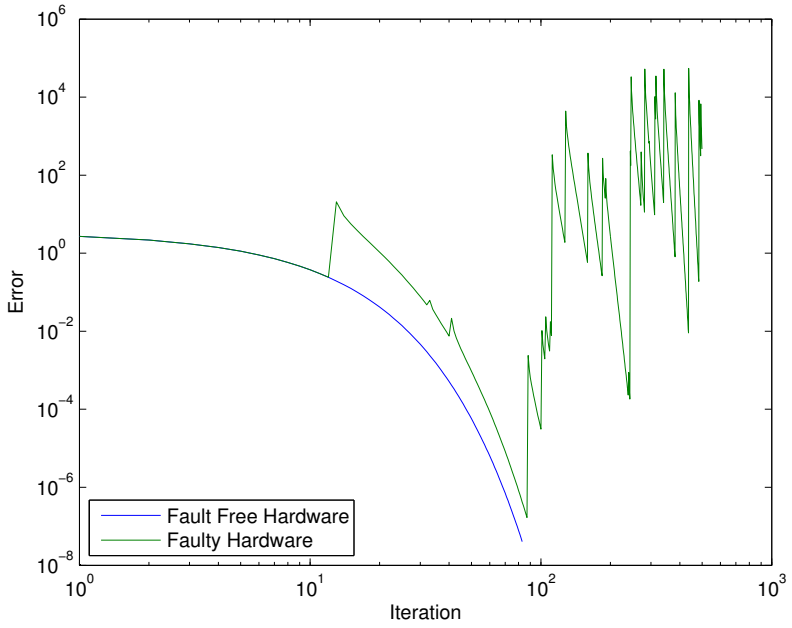

Figure 2: Convergence of the Jacobi method implemented in the classical fixed point itereration of Algortihm 1. When we introduce simulate hardware faults into the computations, the regular fixed point iterations fails to converge.

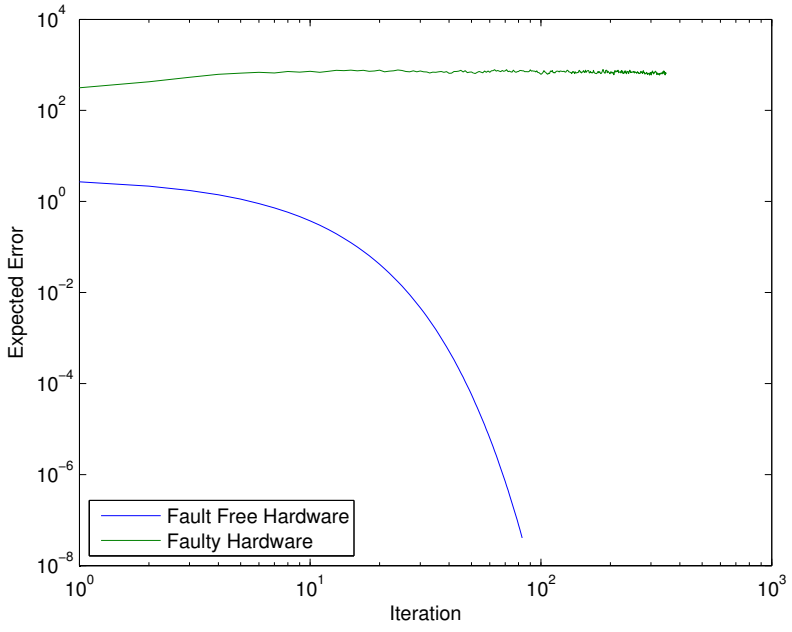

Figure 3: Expected value of the error associated with the regular fixed point iteration of (i.e. Algorithm 1), when the iteration is executed with and without simulated faults. For the first 500 iterations, the expected value of the faulty iteration does not decaly.

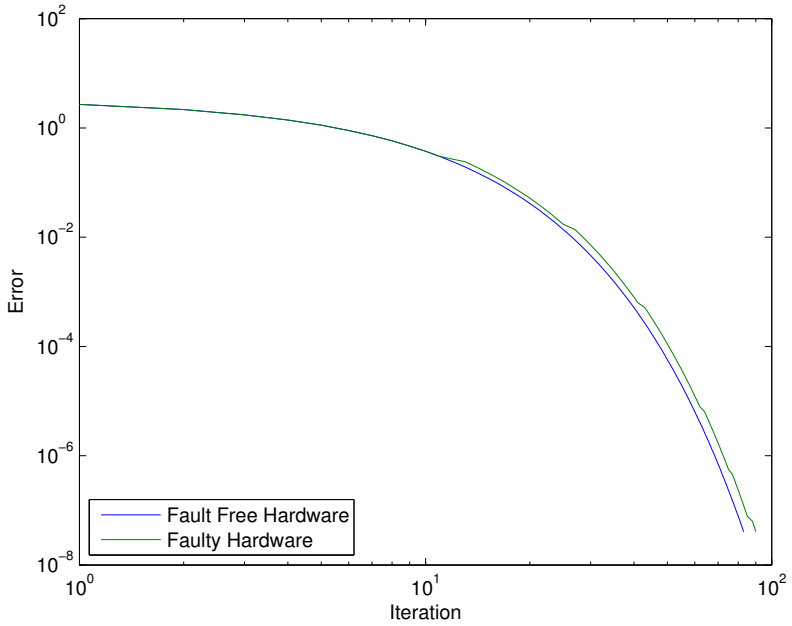

Figure 4: One realization of the resilient fixed point iteratio of Algorithm 3 executed with simulated hardware faults plotted together with the fault free iteration. We observe close match between the two convergence rates.

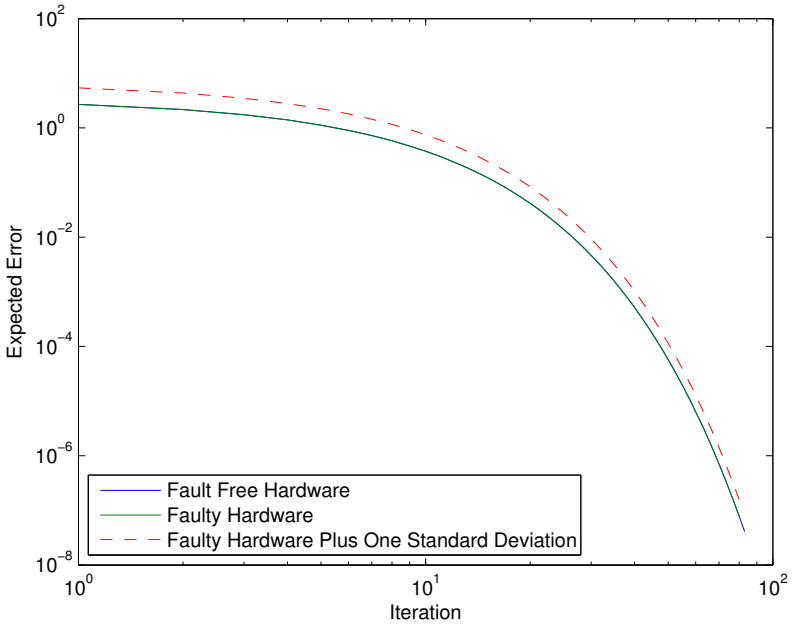

Figure 5: Expected value and standard deviation of the resilient fixed point iteratio of Algorithm 3 executed with simulated hardware faults plotted together with the fault free iteration. The statistics of the resilient algorithm converge to zero which demonstrates that the resilient algorithm is convergent with respect to silent hardware faults.

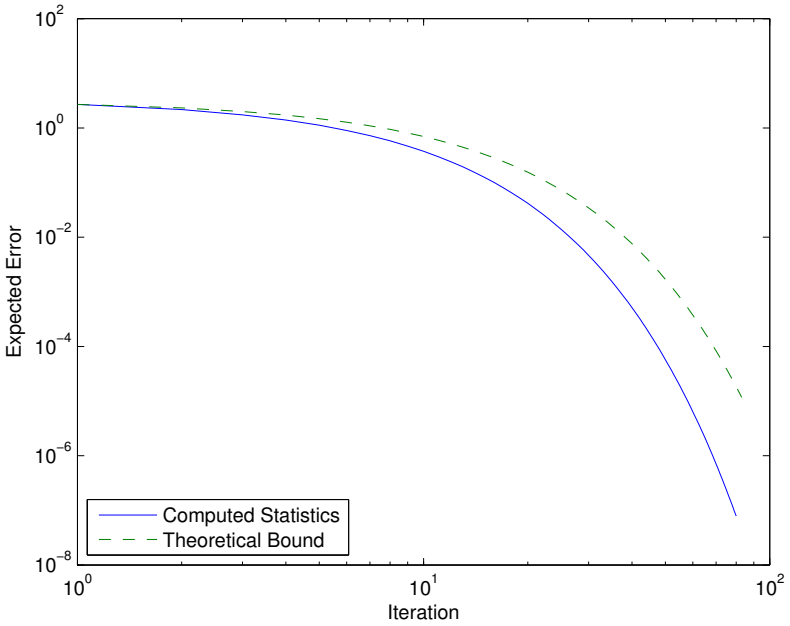

Figure 6: The expacted value of the error associated with the resilient Algorithm 3 and the theoretical error bound. The error analysis considers the worst case scenario for the error and thus the upper bound is not sharp. However, in both cases, we observe the linear convergence rate.

5 Conclusion

This work demonstrates an analytical algorithmic approach for silent hardware faults that can complement existing resilience techniques. Utilizing the hardware and software based resilience techniques, we split the 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 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 a hardware agnostic error model that can be used to analyze the hardware error propagation and algorithm's convergence properties. In particular, we analyzed the classical fixed point iteration and proposed several modifications that can dramatically improve resilience. We perform a numerical comparison between the regular and resilient fixed point methods. We demonstrate that the regular iteration is not convergent, while the resilient one is convergent even when it encounters a very high rate of faults.

REFERENCES

[1] J. DUELL, *The design and implementation of berkeley lab's linux checkpoint/restart*, tr, Lawrence Berkeley National Laboratory, 2000. 3

[2] J. ELLIOT, K. KHARBAS, D. FIALA, F. MUELLER, C. ENGELMANN, AND K. FERREIRA, *Combining partial redundancy and checkpointing for HPC*, in International Conference on Distributed Computing Systems, 2012. 3

[3] J. ELLIOT, F. MULLER, 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. 6, 9

[4] K. FERREIRA, J. STEARLEY, J. H. L. III, R. OLDFIELD, K. PEDRETTI, R. BRIGHTWELL, R. RIESEN, P. BRIDGES, AND D. ARNOLD, *Evaluating the viability of process replication reliability for exascale systems*, in Supercomputing, nov 2011. 3

[5] 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. 2

[6] M. HOEMMEN AND M. A. HEROUX, *Fault-tolerant iterative methods via selective reliability*, in Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC). IEEE Computer Society, 2011. 3, 9

[7] K.-H. HUANG AND J. A. ABRAHAM, *Algorithm-based fault tolerance for matrix operations*, IEEE Transactions on Computers, C-33 (1984), pp. 518–528. 3

[8] A. A. HWANG, I. A. 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. 2

[9] T. KARNIK AND P. HAZUCHA, *Characterization of soft errors caused by single event upsets in cmos processes*, Dependable and Secure Computing, IEEE Transactions on, 1 (2004), pp. 128–143. 2

[10] N. MISKOV-ZIVANOV AND D. MARCULESCU, *Soft error rate analysis for sequential circuits*, in Proceedings of the conference on Design, automation and test in Europe, EDA Consortium, 2007, pp. 1436–1441. 2

[11] A. MOODY, G. BRONEVETSKY, K. MOHROR, AND B. DE SUPINSKI, *Design, modeling, and evaluation of a scalable multi-level checkpointing system*, in Supercomputing, Nov. 2010. 3

[12] E. PINHEIRO, W.-D. WEBER, AND L. A. BARROSO, *Failure trends in a large disk drive population*, in USENIX Conference on File and Storage Technologies, 2007. 2

[13] W. RUDIN, *Principles of mathematical analysis*, vol. 3, McGraw-Hill New York, 1964. 7

[14] 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 ofComputer Systems, 2009, pp. 193–204. 2

[15] V. SRIDHARAN AND D. LIBERTY, *A study of dram failures in the field*, in Supercomputing, Nov. 2012. 2

[16] C. WANG, F. MUELLER, C. ENGELMANN, AND S. SCOTT, *A job pause service under LAM/MPI+BLCR for transparent fault tolerance*, in International Parallel and Distributed Processing Symposium, Apr. 2007. 3

[17] ——, *Proactive process-level live migration in hpc environments*, in Supercomputing, 2008. 3


OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE DEPARTMENT OF ENERGY