Tuning Stationary Iterative Solvers for Fault Resilience

Hartwig Anzt
Innovative Computing Lab
University of Tennessee
Knoxville, Tennessee, USA
hanzt@icl.utk.edu

Jack Dongarra
Innovative Computing Lab
University of Tennessee
Knoxville, Tennessee, USA
dongarra@icl.utk.edu

Enrique S. Quintana-Ortí
Depto. Ingeniería y Ciencia de Computadores
Universidad Jaume I
Castellón, Spain
quintana@icc.uji.es

ABSTRACT

As the transistor's feature size decreases following Moore's Law, hardware will become more prone to permanent, intermittent, and transient errors, increasing the number of failures experienced by applications, and diminishing the confidence of users. As a result, resilience is considered the most difficult under addressed issue faced by the High Performance Computing community.

In this paper, we address the design of error resilient iterative solvers for sparse linear systems. Contrary to most previous approaches, based on Krylov subspace methods, for this purpose we analyze stationary component-wise relaxation. Concretely, starting from a plain implementation of the Jacobi iteration, we design a low-cost component-wise technique that elegantly handles bit-flips, turning the initial synchronized solver into an asynchronous iteration. Our experimental study employs sparse incomplete factorizations from several practical applications to expose the convergence delay incurred by the fault-tolerant implementation.

CCS Concepts

•**Mathematics of computing** → **Solvers;**

Keywords

Sparse linear systems; stationary (and asynchronous) iterative solvers; resilience; fault tolerance; high performance computing

1. INTRODUCTION

As the number of transistors in CMOS integrated circuits continues to grow at the pace dictated by Moore's Law, the rate of faults (e.g., due to the impact of alpha particles or radiation from chip packaging,) will become significant [18]. In addition, with the end of Dennard's scaling [7], power is the key constraint for the performance of current high performance computing (HPC) systems [11, 17]. Near-threshold voltage computing (NTVC) [15] can help increase the levels of hardware concurrency and throughput of these systems, but will do so at the expense of more transistor area and larger volumes of systems faults [14].

ScalA15 November 15-20, 2015, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2832080.2832081

To challenge this background, fault resilience has been identified as a critical component for future supercomputers [16]. In particular, the expected increase in error rates in large HPC systems will render current checkpoint+restart methods too costly. However, adding some fault tolerance mechanism that ensures correctness will be necessary to yield *useful* exascale machines [18]. Without this protection, applications will not run to completion, or even worse, they will return an incorrect answer without any indication.

In this paper, we propose and evaluate a low-cost technique that enhances the resilience of component-wise relaxation (i.e., stationary) solvers for sparse linear systems [19] with respect to transient bit-flips (a class of soft errors or silent data corruption). By nature, this type of iterative method features a certain degree of error resilience in case of hardware outage or a low bit-flip rate [2]. In this work, we move one step forward by integrating an affordable fault tolerance technique that also preserves the convergence of the iteration in case high bit-flip rates occur during the sparse matrix-vector product (`SpMV`) –an operation which typically dominates the computational cost of the solver. More precisely, we propose a light-weight soft error protection mechanism which inspects each iteration approximation of the solution against some bit-flip threshold, accepting or rejecting the updates at the component level. For a benchmark associated with the finite difference discretization of the Laplace problem, we optimize the only threshold parameter of the method; we exercise the strategy against an increasing bit-flip rate; and we investigate the influence of the bit-flip location in the 64-bit IEEE number representation. For a more extensive evaluation, we leverage sparse incomplete factorizations for several cases taken from the University of Florida Matrix Collection[1] (UFMC). The relative convergence delay indicating the overhead incurred by the fault-tolerant implementation, compared with an error-free implementation, reveals the efficiency of the approach.

The rest of the paper is structured as follows. In Section 2 we review a number of related works that have also analyzed fault tolerance/resilience in the solution of sparse linear systems. In Section 3 we offer a brief introduction to stationary solvers, and in Section 4 we describe our fault tolerance technique for low-cost error resilience in this type of method. The paper is closed with an experimental analysis of the technique, in Section 5, followed by concluding remarks in Section 6.

2. RELATED WORK

Several recent works have considered the effect of soft errors, in most cases focusing on data corruption and iterative Krylov subspace methods [19].

[1]Available at https://www.cise.ufl.edu/research/sparse/matrices/. Last accessed: July 2015.

Bronevetsky and de Supinski [3] analyze the vulnerability to soft errors of the Krylov-based solvers in SparseLib [8]. Their study concludes that simple soft error detectors can render low overhead mechanisms with few false positives (FPs), but these techniques do not yield an acceptable reduction in application vulnerability.

Chen [5] introduces an on-line verification of orthogonality and residual to detect soft errors during the execution of Krylov subspace solvers. This technique is then combined with check-pointing to obtain correct results in an unreliable computing scenario.

Hoemmen and Heroux [13] apply selective reliability to assemble a fault-tolerant version of GMRES, furnished with an inner-outer iteration, that converges at a rate that degrades with the fault rate. When faults are rare, this algorithm operates most of the time in unreliable mode, avoiding expensive restarts from checkpoints.

Elliott, Hoemmen, and Mueller [9] present a low cost fault detection mechanism for GMRES, expanding this to limit the magnitude of the error that the inner solve may return. Related with this, the same authors [10] investigate the connection between errors in the IEEE 754 representation of real numbers, the dot product kernel, and the effect of normalizing the data.

Sao and Vuduc [20] depart from previous work by adopting "self-stabilization" to obviate the need for full state saving and fault detection. When a fault occurs, their stabilized Krylov subspace solvers can, by design, reach a correct state in a finite number of steps, while ensuring convergence in the absence of additional faults.

Compared with these efforts, we also address the iterative solution of sparse linear systems, but consider the error resilience of relaxation methods instead of Krylov subspace-based solvers. Despite the often superior convergence characteristics of Krylov methods, component-wise iterations remain an important building block as smoothers for multigrid methods [21], and for the approximate solution of sparse triangular systems within an incomplete factorization preconditioner [6, 1].

Krylov subspace methods are typically combined with an incomplete factorization preconditioner to accelerate convergence. However the application of this preconditioner requires solving two sparse triangular systems per iteration. These solves are traditionally implemented using forward/backward substitution, a procedure which is not only difficult to parallelize but also error prone. Concretely, a single bit-flip rapidly propagates into the complete system damaging the preconditioner as well as the top-level solver iteration. As the sparse triangular factors arising in this context fulfill the convergence condition for asynchronous component-wise iteration, fault-tolerant relaxation methods are a highly attractive alternative to the "exact" triangular solves in an error-prone scenario.

3. STATIONARY METHODS

Consider the linear system $Ax = b$, where $A \in \mathbb{R}^{n\times n}$ is sparse, $b \in \mathbb{R}^n$ is the right-hand side vector, and $x \in \mathbb{R}^n$ is the sought-after solution. Stationary solvers apply component-wise relaxation principles, iteratively updating each individual component of an approximated solution $x^{\{k\}}$, using the matrix A, combined with values of other components, and vector b. A popular example is the Jacobi iteration, which can be formulated as

$$\begin{aligned} x^{\{k\}} &:= D^{-1}\left(b-(A-D)x^{\{k-1\}}\right) = D^{-1}b + Mx^{\{k-1\}} \\ &= y + Mx^{\{k-1\}}, \quad k = 1,2,\ldots, \end{aligned} \tag{1}$$

where $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix containing the diagonal entries of A, and $x^{\{0\}}$ corresponds to a starting solution guess [19].

An appealing property of the "Jacobi update" $y + Mx^{\{k-1\}}$ is that all components of $x^{\{k\}}$ can be obtained in parallel, as the arithmetic operations on any component $x_i^{\{k\}}$ only involve values from $x^{\{k-1\}}$. In contrast, the Gauss-Seidel method [19] requires values from both the last and the current iterate, imposing a specific updating order that strongly constrains its concurrency.

If an implementation of the Jacobi iteration does not update all components, but some of them keep the value from the previous iteration, the method becomes asynchronous (or "chaotic") in the sense that, at a certain point during the iteration process, some components may differ in the number of times they have been updated. This is equivalent to an asynchronous iteration where each component is modified in an arbitrary manner, but always using the latest available values for the remaining components [4, 12]. (The synchronized Jacobi and Gauss-Seidel methods can be viewed as specialized cases of this general class.) Asynchronous iterations require stronger convergence conditions than the synchronized Jacobi and Gauss-Seidel variants [12]. Nonetheless, these criteria are, for example, fulfilled for the sparse triangular factors arising in incomplete factorization preconditioners.

The soft error protection we propose in this work builds upon an implementation of the Jacobi method, rejecting the update of those locations (components) where the error detection test indicates that a bit-flip occurred during the computation. In case of rejected updates, this turns the initial synchronized Jacobi solver into an asynchronous iteration. As this imposes stronger convergence conditions on the iteration matrix, and the number of updates for the distinct components will likely differ during the iteration process, we will refer to the bit-flip protected solver with the term *Fault-Tolerant Jacobi* (`FTJacobi`). This intends to indicate that, for the premise that all updates are accepted, the method becomes a plain implementation of the Jacobi solver.

4. BIT-FLIP PROTECTION FOR JACOBI

Synchronous relaxation methods in general, and the Jacobi method in particular, carry the convenient property of a monotonic residual decay [19]. Consider the relation between the residual $r^{\{k\}} = b - Ax^{\{k\}} \in \mathbb{R}^n$, at an iterate k, and the distance (error) between the approximation $x^{\{k\}}$ at that step and the exact solution $x^{\{*\}}$:

$$r^{\{k\}} = b - Ax^{\{k\}} = A\left(x^{\{*\}} - x^{\{k\}}\right). \tag{2}$$

For Jacobi, the residual decreases linearly with the iteration count, up to convergence in appropriate floating–point format. Hence, a straight-forward strategy for detecting errors arising during the Jacobi iteration consists of checking for a monotonic decrease of the residual vector, accepting the new solution vector only if it passes this test. Unfortunately, this approach has the drawback of being computationally very expensive, as the residual computation (which can also be affected by errors) has about the same cost as the iteration itself. Furthermore, the method will not make any progress in cases of high bit-flip rates.

4.1 Component-wise bit-flip protection

In addition to the monotonic convergence, the Jacobi iteration also fulfills the contraction property of fixed-point iterations at the component level, which states that the difference between the current and last iterate decreases component-wise:

$$\begin{aligned} &\forall i \in [1,n], \quad \exists\, 0 < \theta_i < 1 \ : \\ &\left|x_i^{\{k\}} - x_i^{\{k-1\}}\right| \le \theta_i \left|x_i^{\{k-1\}} - x_i^{\{k-2\}}\right| \le \theta_i^2 \left|x_i^{\{k-2\}} - x_i^{\{k-3\}}\right| \ldots \\ &\equiv \quad z_i^{\{k\}} \qquad \le \theta_i \quad z_i^{\{k-1\}} \qquad\quad \le \theta_i^2 \quad z_i^{\{k-2\}} \ldots . \end{aligned}$$

This allows replacing the costly residual test with a component-wise convergence test, based on the expected difference to the next iterate. For relaxation methods with linear convergence rate for a specific problem, the component-wise ratio

$$c_i := z_i^{\{k-1\}} \,/\, z_i^{\{k\}}, \quad k = 2, 3 \ldots \tag{3}$$

remains constant up to convergence in the respective format. This suggests one should compute c_i in some reliable mode, and exploit the tolerance-based estimation for the difference

$$\left| \frac{z_i^{\{k-1\}}}{z_i^{\{k\}}} - c_i \right| \leq c_i \cdot \delta, \tag{4}$$

for some user-defined threshold δ, as (part of) a component-wise error detection mechanism. Thus, if this condition is not fulfilled, it might be an indicator that an error has occurred, and the update to obtain $x_i^{\{l\}}$ should be rejected. Hereafter we will refer to the test (4) as the *threshold condition* (`T-COND`).

Despite being a good indicator, in particular for the practical case of limited precision, the test `T-COND` is not perfect. On the one hand, the condition is not able to detect all errors since bit-flips with small effect on the component value can "slip" through the threshold. This is acceptable, as errors of such magnitude will likely cause no serious harm to the convergence rate of the relaxation method. On the other hand, a failure to pass the test due to causes other than an actual bit-flip error can result in FPs. These may not cause the divergence of the iteration, but can still result in stagnation of the convergence. The threshold parameter δ provides a mechanism to balance these two effects.

A central question is how the bit-flip protection strategy handles a negative evaluation of `T-COND`. If the strategy rejects the complete Jacobi update, and (almost) any iteration carries a corruption, an FP, or both, this option causes stagnation. Hence, it is beneficial to update only those components where the condition is fulfilled, while rejecting the changes to locations violating `T-COND`. However, accepting some component updates, but rejecting others, destroys the synchronism of the relaxation method. As a consequence, if a component update is rejected, the linear decay of the difference between the last and the new iteration solution approximations will no longer be guaranteed for the remaining components. Instead, components strongly depending on those entries which were not updated may exhibit a small increase of this quantity in the current iteration. These will expose themselves as FPs during the next iteration. Therefore, to also ensure progress in the case of FPs, we propose to extend the strategy by a second condition that becomes relevant if a component was not updated during the previous iteration(s). Concretely, a flag ($flag_i^{fp}$) accumulates how many consecutive times a certain component has not been updated, and the component-wise update is accepted if the *false-positive condition* (`FP-COND`) indicates that the component value does not explode:

$$\frac{z_i^{\{k-1\}}}{z_i^{\{k\}}} > 10^{-flag_i^{fp}} \tag{5}$$

with a certain $\phi \geq flag_i^{fp}$. This bound should be chosen according to the condition number of the system matrix and the sought-after relative residual reduction. Specifically, a higher bound in (5) permits convergence to a very small relative residual, with the danger of residual explosion for ill-conditioned systems. A tighter bound prevents residual explosion at the cost of potential stagnation after a certain residual reduction. In practice, we identified $flag_i^{fp} \leq \phi = 10$ to be a good choice for many problems, as this allows satisfying convergence to machine precision at the risk of a residual increase of —at most— ϕ orders of magnitude multiplied by the last difference $z_i^{\{k-1\}}$.

4.2 Implementation details

Figure 1 offers a practical implementation of the bit-flip protection mechanism implemented in MATLAB®. The function inputs correspond, in that order, to the problem dimension n, the threshold δ, the convergence component-wise ratio c, the values $x^{\{k-1\}}$ and $z^{\{k-1\}}$ from the *prev*ious iterate (`xprev` and `zprev`), the *cur*rent $x^{\{k\}}$ (`xcur`), and values for flags associated with `T-COND` and `FP-COND`. After performing the appropriate operations, the function updates only those entries that pass the test, overwriting the "previous" `xprev` and `zprev` in preparation for the next iteration. This particular implementation exposes the favorable properties of the bit protection mechanism for vectorization and execution on an SIMD unit or a data-parallel architecture. The code involves 12 component-wise operations on vectors of length n, plus a few masked array updates (e.g., `flag_fp(fpcond) = 0;`). Note in particular that this is also true for the `max` and `min` computations, which apply this operator component-wise instead of performing a reduction.

We emphasize that our approach assumes a reliable computation of c, to ensure that the iteration has proceeded beyond the increase in the relative residual that can occur at the beginning of an asynchronous iteration. In practice, we can ensure the precision of c by, for example, repeating (as needed) the first three iterations of the Jacobi method, or performing them on reliable hardware/mode.

To conclude this discussion of practical aspects, let us inspect the update of $flag_{fp}$. Define $d_i = z_i^{\{k\}}$ and assume that, in the next iteration, $z_i^{\{k+1\}}$ does not pass the test and the corresponding component is not be updated. In iteration $k+2$, the update is accepted provided $z_i^{\{k+2\}} < 10^1 \cdot d$. If rejected again, in iteration $k+3$, the update will be accepted if $z_i^{\{k+3\}} < 10^2 \cdot d$; and so on.

5. EXPERIMENTAL EVALUATION

5.1 Setup and solvers

All the experiments in this section were performed using MATLAB (release R2014a) and IEEE 754 double-precision (64-bit) arithmetic, on a server equipped with two Intel Xeon E5-2670 sockets ($2\times$ 16 cores, operating at 2.6 GHz) and 64 GB of RAM.

The baseline for the evaluation of our fault tolerance technique is a plain implementation of the `Jacobi` iteration in (1) using MATLAB. Obviously, this iterative scheme can be expected to experience slow convergence or even divergence in the presence of bit-flips. The fault-tolerant variant `FTJacobi` integrates the soft-error protection defined by (4)–(5), in the form of the bit-flip protection function given in Figure 1.

In both the original and the fault-tolerant codes, the iteration is stopped when the residual satisfies $\|r^k\|_2 < \tau \|x\|_2$, with τ a user-defined threshold.

5.2 Bit-flip injection methodology

At the beginning of each iteration of the Jacobi method, single bit-flips are inserted into ε random positions of the uncorrupted positive iteration matrix M (different at each iteration k), before computing the `SpMV` involving this matrix; see (1). After this, the original values of the matrix are restored, in preparation for the next iteration. These transient errors, therefore, reflect a scenario where, at iteration k, the results of κ floating-point operations involving elements of M get corrupted, but the source of the data (e.g., in

```
function [...] =
Check_cond( n, delta, c, xprev, zprev, xcur,
            flag_t, flag_fp )

% Constant vector
phi = 10;
vk  = 10.^[0:-1:-phi];

% Compute component-wise quotient
zcur   = max( abs(xcur - xprev), eps );
zratio = zprev ./ zcur;

% Evaluate T-COND
t1     = abs( zratio - c );
t2     = delta * c;
tcond  = ( t1 < t2 );

% Update FLAG_FP vector and evaluate FP-COND
flag_fp          = flag_fp + 1;
t3               = min( flag_fp, phi );
t4               = vk( t3 );
fpcond           = ( zratio > t4 );
flag_fp(fpcond) = 0;

% combine T-COND or ( flag_t == 1 & FP-COND )
ccond = tcond | ( flag_t == 1 & fpcond );

% update flag_t
flag_t         = ones(n,1);
flag_t(ccond) = 0;

% Update locations fulfilling T-COND
% or flag_t == 1 and FP-COND
zprev(ccond) = zcur(ccond);
xprev(ccond) = xcur(ccond);
```

Figure 1: Bit-flip protection in MATLAB.

main memory or certain level of the cache) remains correct to be re-loaded in the next iteration $k+1$. We note that, during the k-th iteration, a single error in an element of M corrupts a single entry of x^k. However, with κ errors in M, the number of entries corrupted in x^k equals the number of different rows of M these errors affect, which can be smaller than κ. Furthermore, we assume that all other operations, and in particular the error detection technique, operate in a reliable mode.

5.3 Assessment criteria

We employ the following quantitative "metrics" to expose the properties of the fault tolerance technique:

- Detected and missed bit-flips (DBFs and MBFs, respectively). Obviously, we would like to detect κ bit-flips per iteration and miss none. In practice, because the bit-flips are detected by checking the result x^k, the maximum number of bit-flips that can be detected may be smaller than κ, as two or more errors in the same row of M propagate to the same entry of the result vector, corrupting only a single entry in it.

- False positives (FPs). Detecting a false bit-flip in an entry of x^k implies that this entry is not updated during the current iteration and may slow down convergence. Therefore, we would like to keep the number of FPs low.

- Relative convergence delay μ. We define this metric as a ratio between the number of iterations required by `FTJacobi`, to reach a relative residual bounded by τ in the presence of κ errors per iteration, and the number of iterations required by `Jacobi` to reach the same residual threshold when no errors occur. Ideally, we would like the delay ratio to be close to 1.

5.4 Detailed evaluation of Laplace benchmark

For a comprehensive evaluation of the strategy, we target a finite difference discretization of the Laplace problem in 3D. The system matrix is derived by using a 27-pt stencil on a $16 \times 16 \times 16$ discretization. This results in a regular system matrix with 97,336 nonzeros distributed over 4,096 rows and a small condition number [19] of 9.36e+01. The purpose of our initial evaluation is to assess the impact of the following three factors on the metrics defining the quality of the fault tolerance technique:

- The parameter δ controlling the update in (4).

- The number of bit-flips *per iteration* κ (i.e., per `SpMV`). To make the effects of data corruption rapidly visible, we introduce $\kappa = 40$ errors by default, which is also the scenario chosen in [20]. We note that this number can be considered quite high, as it may corrupt up to 1% of the updates in this particular benchmark.

- The position of the bit-flips. We distinguish between an error flipping any of the 64 bits of a number; or only the sign, exponent, or upper/lower "half" of the mantissa (i.e., its fractionary part). The latter corresponds, respectively, to the bit-flip occurring in positions $\rho = \text{IE}^3(63)$, $\text{IE}^3([52-62])$, or $\text{IE}^3([26-51])/\text{IE}^3([0-25])$ of the IEEE 754 representation of the number.

5.4.1 *Convergence delay and* `T-COND`

We commence with an evaluation of the delay experienced by `FTJacobi` when $\delta = 0.9$, $\kappa = 40$, and $\rho = \text{IE}^3([0-63])$. The left-hand side plot in Figure 2 visualizes the distance z_i^k, along with the threshold condition `T-COND` defined by $c_i \cdot \delta$, for a single component i. For those iterations where z_i^k is outside the boundaries imposed by the condition, the update to x_i^k is rejected. The corresponding convergence of `FTJacobi` for this particular execution is shown in the middle plot of the figure, which also reports the behaviour of an error-free run of `Jacobi` and `FTJacobi`. As expected, in a scenario prone to soft errors, `FTJacobi` exhibits a moderately slower convergence than `Jacobi` due to the rejected updates. This relative convergence delay is quantified in the right-hand side plot in the figure. These results reveal that the delay increases as we raise the threshold to declare convergence, from slightly below 1.03 (3%) for $\tau = 10^{-1}$ to around 1.17 (17%) for $\tau = 10^{-12}$ for this high bit-flip rate. The reason is the different effect of the errors as the iteration progresses and the entries of the approximated solution vector x^k incorporate more accurate digits. In the initial iterations, we can expect that only the sign/exponent and a few bits in the most significant part of the mantissa are correct. Therefore, errors flipping the less significant bits of the mantissa do not affect the convergence rate. However, as the iteration proceeds, x^k "incorporates" more and more correct flips, increasing the negative impact of undetected errors. The analysis of the effect of the bit-flip position in Section 5.4.4 sheds further light into this behavior.

We note that the left plot of Figure 2 reflects the results from a single execution of the Jacobi-based solvers. The other two plots in that figure, as well as all experiments in the remainder of the paper, report data that is averaged over the execution of 100 repetitions of the experiments, with different random seeds in order to generate different error positions.

5.4.2 *Impact of* δ

Let us now optimize the value of the threshold δ that trades off FPs against MBFs. To illustrate the effect of this parameter, we take

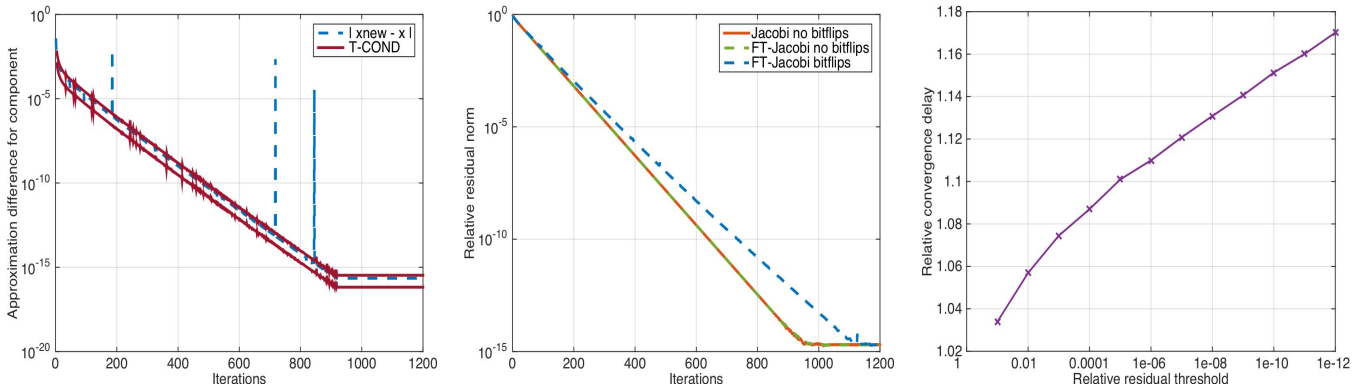

Figure 2: Convergence delay experienced by `FTJacobi` applied to the Laplace benchmark. $\delta = 0.9$, $\kappa = 40$, and $\rho = \text{IE}^3([0-63])$.

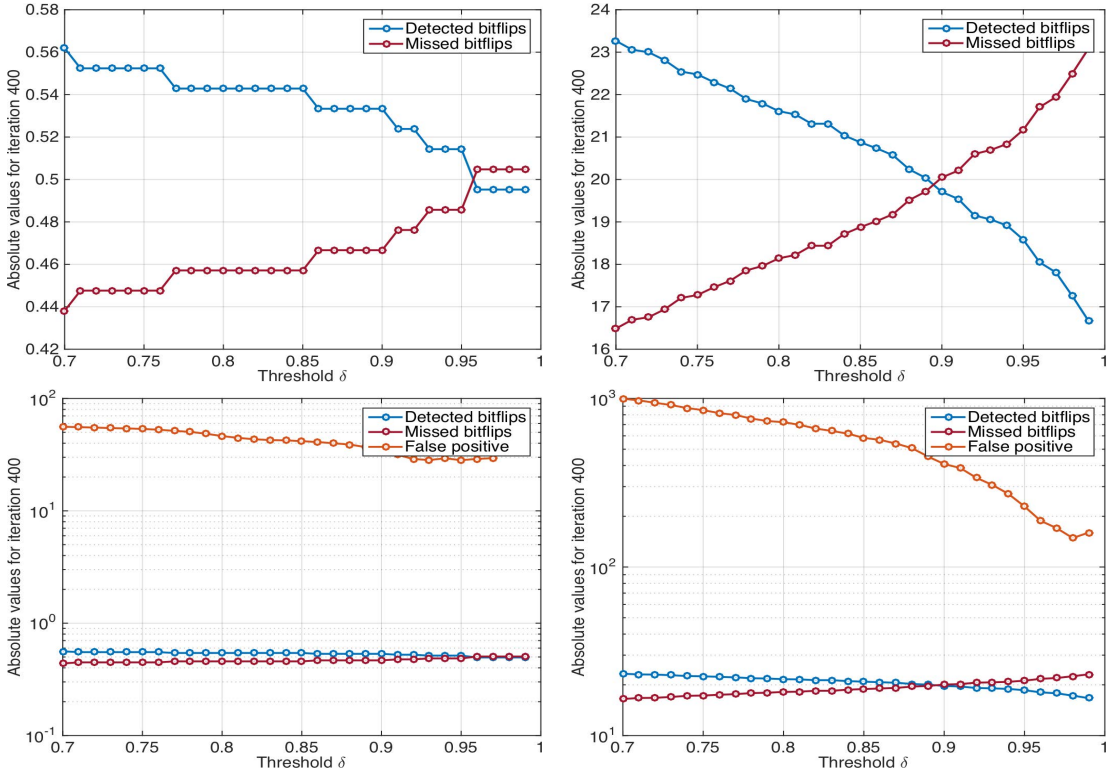

Figure 3: Detected/missed bit-flips and false positives for iteration 400 experienced by `FTJacobi` applied to the Laplace benchmark. $\delta \in [0.7, 1]$, $\kappa = 1$ (left) and $\kappa = 40$ (right), and $\rho = \text{IE}^3([0-63])$.

a "snapshot" of the iteration process at iteration 400 which, for this particular example, is somewhere in the middle of the convergence process. We analyze how many bit-flips are detected/missed and how many FPs appear at this specific iteration. The plots in the first row in Figure 3 show the number of DBFs/MBFs with a single and 40 bit-flips per `SpMV` (left and right plots, respectively). Both cases motivate the selection of a small threshold δ in order to catch more errors. The plots in the second row also include the number of FPs, which is about two orders of magnitude larger. This shows that accepting more updates, by choosing a larger threshold δ, decreases the number of FPs. In particular, in case of a large bit-flip rate (right-hand side plot), the number of FPs dramatically decreases as $\delta \to 1$. The convergence delay balances the effects of MBFs/FPs; see Figure 4. For the scenario we address, the optimal threshold is almost independent of the number of corruptions, in the range [0.94, 0.97]. However, we emphasize that FPs may delay the convergence and, in the worst case, result in stagnation, while MBFs can potentially cause the explosion of the residual. Especially for ill-conditioned systems, this may ask to accept a larger relative convergence delay at the advantage of a more robust bit-flip protection.

5.4.3 *Bit-flips per iteration*

We now investigate the relationship between the number of bit-flips per `SpMV` and the relative convergence delay using the experimental $\delta = 0.9$. The results in Figure 5 confirm the expectation that increasing the number errors in the computation of the updates yields higher relative convergence delays. For reasonable bit-flip

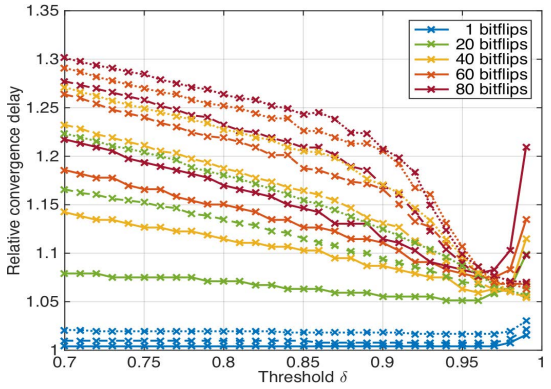

Figure 4: Convergence delay experienced by `FTJacobi` applied to the Laplace benchmark. $\delta \in [0.7,1]$, $\kappa \in \{1,20,40,60,80\}$, and $\rho = \text{IE}^3([0-63])$. The lines with different pattern correspond to $\tau = 10^{-4}, 10^{-8}, 10^{-12}$.

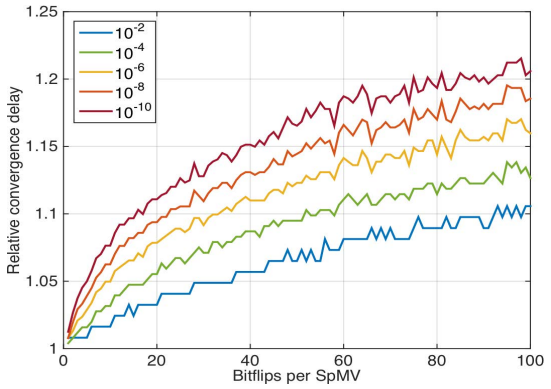

Figure 5: Convergence delay experienced by `FTJacobi` applied to the Laplace benchmark. $\delta = 0.9$, $\kappa \in [1,100]$, $\rho = \text{IE}^3([0-63])$, and $\tau \in \{10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}, 10^{-10}\}$.

rates, in the range of corrupting 0.1% of the computations in the Jacobi update (i.e., less than 5 bit-flips per `SpMV`), the relative convergence delay stays below 10% for all residual thresholds τ. Extremely high bit-flip rates can incur an up to 20% delay for `FTJacobi`.

5.4.4 *Bit-flip position*

Our last experiment with the Laplace benchmark evaluates the impact of the bit-flip position in the detection properties of the bit-flip protection mechanism and the impact of MBFs as well as FPs on the convergence delay. Figure 6 assesses these two metrics in four different scenarios, corresponding to the bit-flips affecting the sign, exponent, lower or upper part of the mantisa. The first rows show that corrupting the sign bit or the exponent of a matrix entry in A has significant impact on `Jacobi`, causing either stagnation (sign bit) or even explosion (exponent) of the residual (see left-hand side plots). The error protection mechanism of `FTJacobi` generally catches all these bit-flips. Only during the very first iterations, does this error type slip through. For the mantissa, the bit-flips become crucial — and are detected — only if the location is relevant in the convergence process. The more significant the position in the mantissa is (the higher the index of the position in the IEEE format), the earlier the bit-flip becomes relevant for the `Jacobi` convergence, and the easier it is to detect this corruption for the protection mechanism. An important takeaway from this experiment is that "small" errors (in the mantissa) are not important in the initial iterations, but become important later.

Matrix	Factor	Dimension	Nonzeros	Condition number
CHP	L	20,082	150,616	7.90e+05
	U	20,082	150,616	1.75e+11
DC	L	116,835	441,781	6.54e+10
	U	116,835	441,450	6.50e+09
STO	L	213,360	1,660,005	1.38e+07
	U	213,360	1,575,003	6.08e+07
VEN	L	62,424	890,108	1.85e+07
	U	62,424	890,108	2.51e+10

Table 1: Characteristics of the sparse triangular factors employed in the experimentation.

5.5 Benchmark problems

Finally, we turn to different benchmarks to evaluate the efficiency of the bit-flip protection in a more general setting. For this purpose, we look into the approximate solution of sparse triangular systems arising for incomplete factorization preconditioners. Particularly, we consider the same problems that are solved using relaxation without bit-flip protection in [1]. These cases arise from the incomplete LU factorization of test matrices from the UFMC. Table 1 illustrates some key characteristics of the corresponding sparse triangular factors. A key difference with respect to the previous experiments is that, for incomplete factorization preconditioning, we do not require solving the sparse triangular systems with high accuracy. Typically, an approximated solution reducing the initial residual by a moderate factor is sufficient to produce the same preconditioning benefits in the top-level solver as those attained from an "exact" triangular solve via forward-backward substitution [6, 1]. For this reason, we investigate the performance of the bit-flip protection for the relative residual stopping criteria $\rho \in \{10^{-1}, 10^{-2}\}$.

Table 2 lists the relative convergence delay (together with other statistics) for scenarios with $\kappa = 1$ and $\kappa = 40$ bit-flips per `SpMV`. For STO, convergence to $\tau = 10^{-1}$ is reached within the reliable iterations. Furthermore, it can be observed that in the case of 40 corruptions per update, the relative residual cannot be decreased by more than one order of magnitude for the upper triangular factors U of the DC and STO problem. The reason is that the nonzero entries in these factors are not evenly distributed, but lie all in one row. When inserting 40 bit-flips in random nonzero locations, the dense rows will almost always carry at least one corruption. As a result, the `FTJacobi` is also unable to make any progress towards convergence. Except for the aforementioned case, `FTJacobi` provides satisfying convergence to the target accuracy with moderate convergence delay.

6. CONCLUDING REMARKS

We have proposed a bit-flip protection mechanism that transforms a synchronized Jacobi iteration —with no inherent protection against errors— into an asynchronous fault tolerant relaxation method. The error protection scheme operates at the component level, individually accepting or rejecting updates at each iteration, depending on whether they pass certain tests based on a single user-defined threshold, δ. Furthermore, the tests are composed of a few Level-1 BLAS-like operations which can be efficiently implemented in vector (SIMD) units, yielding an affordable overhead.

Our detailed experiments with a sparse benchmark show that the optimal threshold δ is almost independent of the corruption rate, and offers a means to balance the effects of MBFs vs FPs.

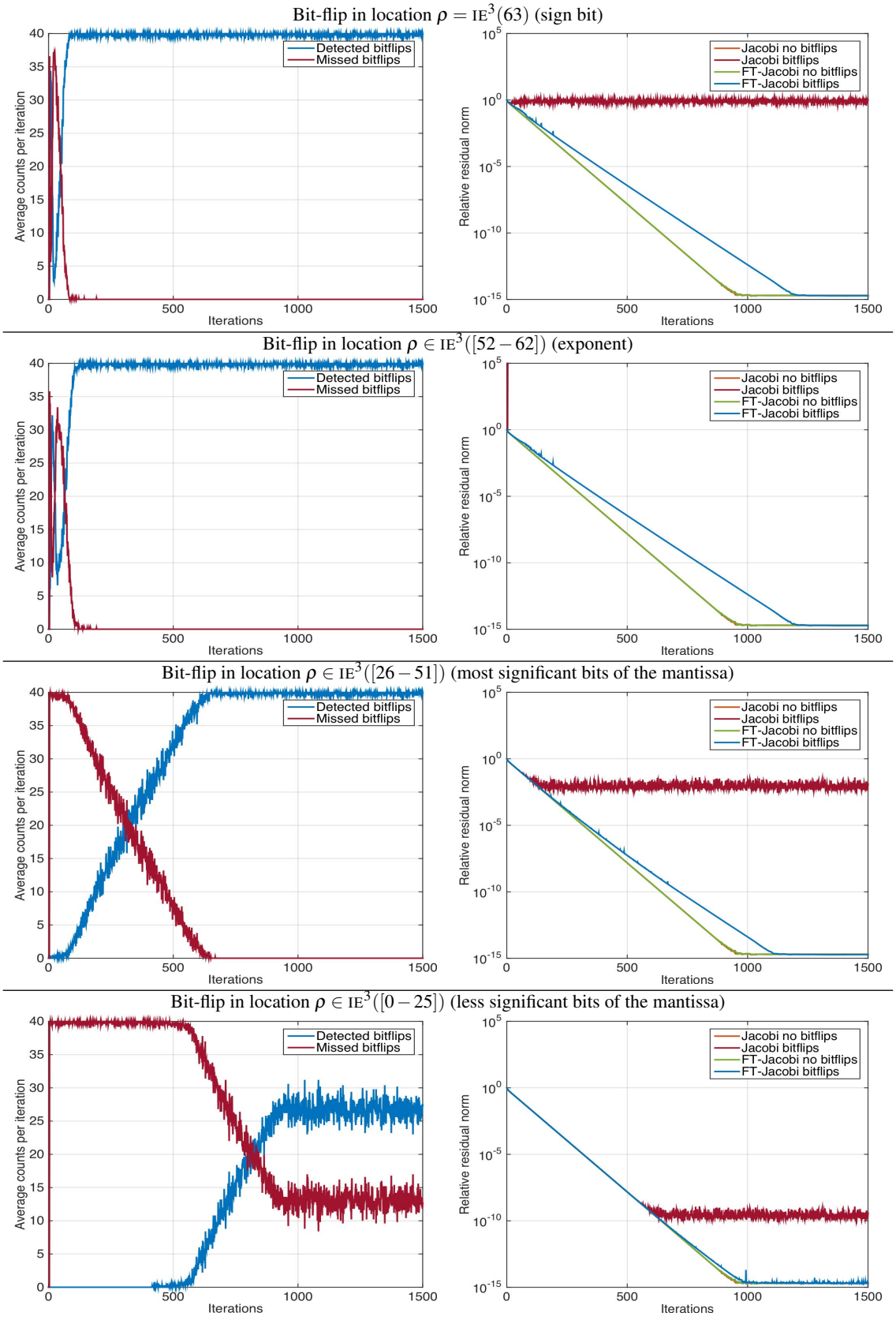

Figure 6: Detected/missed bit-flips (left) and convergence delay (right) experienced by `FTJacobi` applied to the Laplace benchmark. $\delta = 0.0$, $\kappa = 40$, and different positions of the bit-flips.

Matrix	Factor	$\kappa = 1$								$\kappa = 40$							
		$\tau = 10^{-1}$				$\tau = 10^{-2}$				$\tau = 10^{-1}$				$\tau = 10^{-2}$			
		μ	DBF	MBF	FPs	μ	DBF	MBF	FPs	μ	DBF	MBF	FPs	μ	DBF	MBF	FPs
CHP	L	1.00	10%	90%	4.16e+04%	1.00	10%	90%	9.56e+04%	1.00	11%	89%	1.05e+03%	1.10	16%	84%	2.94e+03%
	U	1.00	20%	80%	5.70e+03%	1.00	13%	87%	1.20e+04%	1.00	12%	88%	1.42e+02%	1.00	12%	88%	4.01e+02%
DC	L	1.17	53%	47%	6.97e+06%	1.14	55%	45%	5.63e+06%	1.17	66%	34%	1.74e+05%	1.14	62%	38%	1.41e+05%
	U	1.33	65%	35%	1.68e+06%	1.29	72%	28%	1.92e+06%	3.00	83%	17%	5.69e+04%	NaN	—	—	—
STO	L	1.00	*	*	*	1.00	40%	60%	4.36e+06%	1.00	*	*	*	1.00	42%	59%	1.09e+05%
	U	1.00	*	*	*	1.00	60%	40%	4.48e+06%	1.00	*	*	*	NaN	—	—	—
VEN	L	1.23	28%	72%	1.91e+06%	1.22	42%	58%	2.08e+06%	1.23	36%	64%	4.76e+04%	1.22	42%	58%	5.19e+04%
	U	1.12	28%	72%	1.78e+06%	1.19	39%	61%	1.85e+06%	1.12	31%	69%	4.45e+04%	1.19	36%	64%	4.61e+04%

Table 2: Relative convergence delay μ of `FTJacobi` applied to UFMC benchmarks for $\delta = 0.9$, $\kappa = 1$ (left) and $\kappa = 40$ (right), and $\rho = \text{IE}^3([0-63])$. The number of DBPs, MBPs, and FPs is given relative to the number of actual corruptions.

This evaluation also reveals that, for reasonable bit-flip rates, in the range of corrupting 0.1% of the computations, the relative convergence delay stays below 10% for all residual thresholds. Thirdly, this example shows the interplay among the location of bit-flips in the IEEE standard, the volume of MBF/DBF, and the convergence of the iterations. Concretely, as the error location moves towards the less significant part(s) of the number, they become easier too miss but exert a milder effect on the relative residual until that level of accuracy is reached.

Finally, we have applied the fault tolerant solver to several sparse triangular linear systems arising from the sparse incomplete factorizations of four benchmarks from the UFMC. The results show that unless the bit-flips repeatedly corrupt the same component, the protection mechanism is also able to ensure convergence for high bit-flip rates with a reasonable convergence delay.

Acknowledgments

This work was partly funded by the U.S. Department of Energy (Award Number DE-SC-0010042), and the Russian Scientific Foundation (Agreement N14-11-00190). E. S. Quintana-Ortí was supported by projects TIN2011-23283 and TIN2014-53495-R of the Spanish *Ministerio de Economía y Competitividad*.

7. REFERENCES

[1] H. Anzt, E. Chow, and J. Dongarra. Iterative sparse triangular solves for preconditioning. In J. L. Träff, S. Hunold, and F. Versaci, editors, *Euro-Par 2015: Parallel Processing*, volume 9233 of *Lecture Notes in Computer Science*, pages 650–661. Springer Berlin Heidelberg, 2015.

[2] H. Anzt, S. Tomov, J. Dongarra, and V. Heuveline. A block-asynchronous relaxation method for graphics processing units. *Journal of Parallel and Distributed Computing*, 73(12):1613–1626, 2013.

[3] G. Bronevetsky and B. de Supinski. Soft error vulnerability of iterative linear algebra methods. In *Proc. 22nd Annual Int. Conf. on Supercomputing*, ICS'08, pages 155–164, 2008.

[4] D. Chazan and W. Miranker. Chaotic Relaxation. *Linear Algebra and Its Applications*, 2(7):199–222, 1969.

[5] Z. Chen. Online-ABFT: An online algorithm based fault tolerance scheme for soft error detection in iterative methods. In *Proc. 18th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, PPoPP'13, pages 167–176, 2013.

[6] E. Chow and A. Patel. Fine-grained parallel incomplete LU factorization. *SIAM Journal on Scientific Computing*, 37(2):C169–C193, 2015.

[7] R. Dennard, F. Gaensslen, V. Rideout, E. Bassous, and A. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE J. Solid-State Circuits*, 9(5):256–268, 1974.

[8] J. Dongarra, A. Lumsdaine, X. Niu, R. Pozo, and K. Remington. Lapack working note 74: A sparse matrix library in c++ for high performance architectures. Technical report, Knoxville, TN, USA, 1994.

[9] J. Elliott, M. Hoemmen, and F. Mueller. Evaluating the impact of SDC on the GMRES iterative solver. In *Proc. 2014 IEEE 28th Int. Parallel and Distributed Processing Symp.*, IPDPS'14, pages 1193–1202, 2014.

[10] J. Elliott, M. Hoemmen, and F. Mueller. Exploiting data representation for fault tolerance. In *Proc. 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ScalA'14, pages 9–16, 2014.

[11] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proc. 38th Annual Int. Symp. Computer architecture*, ISCA'11, pages 365–376, 2011.

[12] A. Frommer and D. B. Szyld. On asynchronous iterations. *Journal of Computational and Applied Mathematics*, 123:201–216, 2000.

[13] M. Hoemmen and M. A. Heroux. Fault-tolerant iterative methods via selective reliability. In *Proc. 2011 Int. Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.

[14] D. Kanter. Intel's near-threshold voltage computing and applications, September 2012. http://www.realworldtech.com/near-threshold-voltage/.

[15] U. Karpuzcu, N. S. Kim, and J. Torrellas. Coping with parametric variation at near-threshold voltages. *Micro, IEEE*, 33(4):6–14, 2013.

[16] P. Kogge et al. ExaScale computing study: Technology challenges in achieving ExaScale systems, 2008. http://users.ece.gatech.edu/~mrichard/ExascaleComputingStudyReports/exascale_final_report_100208.pdf.

[17] J. F. Lavignon *et al*. ETP4HPC strategic research agenda achieving HPC leadership in Europe, 2013. http://www.etp4hpc.eu/.

[18] R. Lucas *et al*. Top ten Exascale research challenges, 2014. http://science.energy.gov/~/media/ascr/ascac/pdf/meetings/20140210/Top10reportFEB14.pdf.

[19] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.

[20] P. Sao and R. Vuduc. Self-stabilizing iterative solvers. In *Workshop Latest Advances in Scalable Algorithms for Large-Scale Systems*, pages 4:1–4:8, 2013.

[21] U. Trottenberg and A. Schuller. *Multigrid*. Academic Press, Inc., 2001.