informatics mathematics

Adaptive fault detection criterion for the generalized minimal residual method

Alexandre Moreau, Luc Giraud, Emmanuel Agullo

RESEARCH REPORT
N° 1
July 2016
Project-Teams HiePACS

ISSN 0249-6399 ISRN INRIA/RR--1--FR+ENG

Adaptive fault detection criterion for the generalized minimal residual method

Alexandre Moreau*, Luc Giraud*, Emmanuel Agullo

Project-Teams HiePACS

Research Report n° 1 — July 2016 — ?? pages

Abstract: We have studied the impact of transient soft errors in the application of the matrix vector product or in the preconditioner of preconditioned GMRES. We have analyzed those faults and we are able to successfully correlate those faults to the framework of inexact GMRES theory. We have derived a heuristic to detect faults. This heursitic is extremely robust to detect faults for GMRES and left-preconditioned GMRES and pretty robust in the case of right-preconitioned GMRES. An analysis on the set up of the heurisitcs furthermore identified the critical parameter $(y_{l,k})$ that we had to approximate in our heuristic and that may lead to miss a few faults.

Key-words: No keywords

* Shared foot note

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Rapport de 3 mois

Résumé : Pas de résumé

Mots-clés : Pas de motclef

Contents

1 Introduction

The current trend to improve supercomputers' performance is to increase the component count while reducing their size. Future exascale systems will gather several millions of tiny CPU cores, which will drive the error rate up to many times per day. This issue is already taking place in some present petascale systems, that experience up to 3 failures per day (according to The Computer Failure Data Repository [?], LANL). In case of a failure, an application without any fault tolerance mechanism may either stop or silently terminate with an incorrect output, which can have serious consequences in some applications. For this reason, we study the impact of faults and explore strategies to enhance the fault tolerance of particular applications in a specific context described below.

Errors can be classified into two types, hard errors and soft errors. Hard errors are permanent and unrecoverable errors, for instance a system crash or a broken component, whereas soft errors are silent data corruptions that usually do not affect the system reliability, but may lead to incorrect results. In the following, we focus on the study of soft error tolerance.

Several schemes have been designed to enhance HPC system resiliency to soft errors. On one hand, hardware mechanisms usually relying on the addition of redundancy in the circuits are being used, but may be too costly in terms of hardware and energy to be systematically applicable in the future. On the other hand, software schemes are often more flexible and provide fault tolerance at lower cost.

Software schemes may be system-wide or at the application level. The standard system-wide method is Checkpoint/Restart: the system state is periodically saved into a safe memory storage, and a rollback is performed from a previous correct state whenever an incorrect state is detected. However this approach becomes expensive as the system size increases, and may not be applicable in the long term as the mean time between failures (MTBF) gets closer and closer to the mean time to repair (MTTR). Moreover, it still requires a reliable fault detection mechanism. At the application level, this method is also being used and usually benefits from much smaller sized states, enabling more frequent checkpoints and shorter recovery times.

Other approaches at the application level include Algorithm Base Fault Tolerance (ABFT) such as check-sum based techniques, introduced by Kuang-Hua Huang and J. A. Abraham in [?] and usually provide fault tolerance at lower cost and hardware overhead than standard application level schemes. More recently, studies on the algorithms numerical properties as well as floating point analysis have been used to provide evidences and quantify the impact of any potential perturbation occurring during the algorithm execution. More information about the current state of the art of the resilience in HPC can be found in [?].

One of the most important and time-consuming kernel in numerical schemes is the solution of large sparse linear systems. One popular algorithm for solving such systems it the generalized minimum residual (GMRES) algorithm [?]. In this report, we analyze the impact of soft-errors on the convergence of GMRES and deduce a new, robust fault-detection mechanism.

The rest of the report is organized as follows. In Section 2, the GMRES algorithm is described, in particular the variants we are interested in (full-GMRES, with and without preconditioner). In Section 3, the method, definitions and assumptions used throughout the numerical experiments are detailed. In Section 4, the impact of faults in GMRES is empirically studied. In Section 5, an analysis of the error introduced by faults is proposed, to quantify their impact and predict their influence on the convergence. In Section 6, an oracle-based detection scheme is described and evaluate. In Section 7, a more practical implementation of the detection scheme, based on some approximations, is proposed and evaluated. Finally a discussion on the results and concluding remarks are given.

2 Solving linear systems of equations in HPC

Large computational problems tackled by HPC systems often arise out of complex phenomenon models. To dispose of powerful mathematical tools, most of those models are linear or are linearized so that the innermost calculation reduces to the solution of the linear system of equations of the form

$$Ax = b$$

where $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$ are known and $x \in \mathbb{R}^n$ is to be found with $n \gg 1$ up to a few tens of millions.

Several methods for computing a good solution to this equation are available, separated in two classes of algorithms. On one hand, direct solvers based on Gaussian elimination [?] are very robust and provide an accurate solution in term of backward error [?] at the end of the computation. On the other hand iterative solvers are given a first approximation x_0 of the solution x as an additional parameter, and compute a sequence (x_i) of improving approximations until the target accuracy is reached. In certain situations, the iterative methods have proved to be more practical than the direct ones, especially when the problem size is large, in which case direct methods can be prohibitively expensive. Depending on the structure and numerical properties of matrix A, some iterative methods will perform better than others. In particular, when A is a large, sparse and non-singular matrix, Krylov methods are particularly adapted. When the matrix A is non-symmetric, GMRES [?] is among the best known Krylov method to use [?].

2.1 Krylov methods

All Krylov methods designed to solve a linear system $Ax = b$ have in common the construction of the Krylov space of order i associated to A and b:

$$\mathcal{K}_i(A, b) \equiv \operatorname{span}(b, Ab, ..., A^{i-1}b).$$

The reason behind the use of such a space is derived from the Cayley-Hamilton theorem, which implies that A^{-1} can be expressed as a linear combination of powers of A. Consequently the solution $x = A^{-1}b$ belongs to $\mathcal{K}_i(A, b)$ for r large enough. The basis $\{b, Ab, A^2b...\}$ is iteratively computed using repeated matrix-vector product, until the distance between the spanned Krylov space and the solution is small enough. The solution x can then be efficiently approximated inside this small-sized space. However, the basis vectors quickly become linearly dependent, which usually requires the use of some orthogonalization scheme to enhance the numerical stability.

2.2 The GMRES algorithm

The GMRES method (Algorithm **??**) was first described in 1986 by Saad and Schultz [?] and is one of the most popular Krylov method for solving non-symmetric linear systems. It relies on the repetition of the Arnoldi iteration, which recursively generates an orthonormal basis $V_i = \{v_0, v_1, ...v_i\}$ for $\mathcal{K}_i(A, b)$ to project the initial problem $Ax = b$ onto. The first Arnoldi vector v_0 is initialized to the first residual normalized $v_0 = r_0/\|r_0\| = (Ax_0 - b)/\|Ax_0 - b\|$ (line 2-4) and the recursive relation maintained throughout the algorithm is

$$AV_i = V_{i+1}H_i$$

where H_i is an upper Hessenberg rectangular matrix of size $i+1 \times i$ produced by the orthonormalization scheme. The most widely used scheme is the modified Gram-Schmidt process (line 8-13), although variants using the classical Gram-Schmidt or the Householder process also exist but are numerically less stable or more expensive. The initial least-square problem of minimizing the residual norm $\|r\| = \|Ax - b\|$ becomes equivalent to a new least-square problem, easier to solve thanks to the structure and size of matrix H_i:

$$\underset{\mathbf{x}\in\mathcal{K}_i(A,b)}{\arg\min} \|A\mathbf{x} - b\| = \underset{\mathbf{y}\in\mathbb{R}^i}{\arg\min} \|\beta e_1 - H_i\mathbf{y}\| \text{ with } e_1 = [1, 0, ...0]^T \text{ and } \beta = \|r_0\|. \tag{1}$$

The upper-Hessenberg matrix H can be efficiently transformed into an upper-triangular matrix, using QR factorization (for instance Givens rotations or Householder), to solve the new least-square problem using back substitution [?] (line 14). The Arnoldi process is repeated until the computed residual reaches the target accuracy, which requires at most n iterations in exact arithmetic, and the initial problem solution is expressed back into the initial basis using $x = x_0 + V_i y_i$ (line 17) where y_i is the solution of Equation (??).

Algorithm 1: The Full-GMRES algorithm for iteratively solving the equation $Ax = b$.

```
1  GMRES ($A, b, x_0, \varepsilon$);
   Input     : $A$ ($n \times n$ matrix), $b$ (size n vector), $x_0$ (size n vector)
   Parameter: $\varepsilon$ (target accuracy)
   Output    : $x$ so that $\frac{\|Ax-b\|}{\|b\|} < \varepsilon$
2  $r_0 \leftarrow Ax_0 - b$;
3  $\beta \leftarrow \|Ax_0 - b\|$;
4  $v_0 \leftarrow \frac{r_0}{\beta}$;
5  $e_1 \leftarrow [1, 0, ..., 0]$;
6  for $i \leftarrow 0$ to $n-1$ do
       // Arnoldi iteration
7      $w \leftarrow A \cdot v_i$                          // Compute the next Arnoldi vector
8      for $k \leftarrow 0$ to $i$ do                       // Orthonormalize w against
9          $h_{k,i} \leftarrow w^H v_k$                     // all previous Arnoldi vectors
10         $w \leftarrow w - h_{k,i} v_k$                   // (Modified Gram-Schmidt method)
11     end
12     $h_{i+1,i} \leftarrow \|w\|$;
13     $v_{i+1} \leftarrow w / h_{i+1,i}$ ;
14     $y_i \leftarrow \arg\min(\|\beta e_1 - H_i y_i)\|$;   // Solve the inner least-square problem
15     $r'_i \leftarrow \beta e_1 - H_i y_i$;                // Compute the estimated residual
16     if $\|r'_i\| < \varepsilon$ then
17         $x_i \leftarrow x_0 + V_i y_i$;                   // Update the solution
18         break
19     end
20 end
21 return $x = x_i$;
```

Moreover, the GMRES algorithm is often used with a preconditioner, which is a matrix M designed to increase the execution convergence rate by transforming the initial $Ax = b$ problem into a simpler one. For instance, in the case of right preconditioning, the initial $Ax = b = AM^{-1}Mx = b$ problem is split into two simpler ones to solve, $AM^{-1}y = b$ and $Mx = y$. Many good preconditioners have been studied and in the following, we will consider a popular one, the Incomplete LU decomposition (ILU) of matrix A.

In the following, all execution are performed without restart, using full-GMRES. We refer to the full-GMRES algorithm without preconditioner as GMRES, and to the full-GMRES algorithm with preconditioner as preconditioned-GMRES. When not precised, the preconditioned-GMRES variant used is right-preconditioned-GMRES, and usually means that no difference can be observed between the left and right variants.

Algorithm 2: The (right) preconditioned full-GMRES algorithm with ILU preconditioner.

```
1  GMRES ($A, b, x_0, \varepsilon$);
   Input     : $A$ ($n \times n$ matrix), $b$ (size n vector), $x_0$ (size n vector)
   Parameter: $\varepsilon$ (target accuracy)
   Output    : $x$ so that $\frac{\|Ax-b\|}{\|b\|} < \varepsilon$
2  $r_0 \leftarrow Ax_0 - b$;
3  $\beta \leftarrow \|Ax_0 - b\|$;
4  $v_0 \leftarrow \frac{r_0}{\beta}$;
5  $e_1 \leftarrow [1, 0, ..., 0]$;
6  $M \leftarrow \mathrm{ILU}(A)$;
7  for $i \leftarrow 0$ to $n-1$ do
       // Arnoldi iteration
8      $z_i \leftarrow \mathrm{solve}(M, v_i)$                    // Solve the $Mz_i = v_i$ problem
9      $w \leftarrow A \cdot v_i$                              // Compute the next Arnoldi vector
10     for $k \leftarrow 0$ to $i$ do                          // Orthonormalize w against
11         $h_{k,i} \leftarrow w^H v_k$                        // all previous Arnoldi vectors
12         $w \leftarrow w - h_{k,i} v_k$                      // (Modified Gram-Schmidt method)
13     end
14     $h_{i+1,i} \leftarrow \|w\|$;
15     $v_{i+1} \leftarrow w / h_{i+1,i}$ ;
16     $y_i \leftarrow \arg\min(\|\beta e_1 - H_i y_i)\|$;     // Solve the inner least-square problem
17     $r'_i \leftarrow \beta e_1 - H_i y_i$;                  // Compute the estimated residual
18     if $\|r'_i\| < \varepsilon$ then
19         $x_i \leftarrow x_0 + \mathrm{solve}(M, V_i y_i)$;  // Update the solution
20         break
21     end
22 end
23 return $x = x_i$;
```

3 Methodology

In the following, several definitions and assumptions are given to define the models and enable the experiment reproducibility.

3.1 GMRES convergence

A GMRES execution is considered to have converged to the target accuracy ε when at a given iteration ℓ, the norms of the true residual (i.e., $\|b - Ax_\ell\|$) and the computed one (i.e., $\|\beta e_1 -$

Algorithm 3: The (left) preconditioned full-GMRES algorithm with ILU preconditioner.

```
GMRES $(A, b, x_0, \varepsilon)$;
Input     : $A$ ($n \times n$ matrix), $b$ (size n vector), $x_0$ (size n vector)
Parameter: $\varepsilon$ (target accuracy)
Output    : $x$ so that $\frac{\|M^{-1}(Ax-b)\|}{\|M^{-1}b\|} < \varepsilon$, M = spilu(A)
$M \leftarrow \mathrm{ILU}(A)$;
$r_0 \leftarrow \mathrm{solve}(M, Ax_0 - b)$;
$\beta \leftarrow \|r_0\|$;
$v_0 \leftarrow \frac{r_0}{\beta}$;
$e_1 \leftarrow [1, 0, ..., 0]$;
for $i \leftarrow 0$ to $n-1$ do
    // Arnoldi iteration
    $w \leftarrow A \cdot v_i$                          // Compute the next Arnoldi vector
    $w \leftarrow \mathrm{solve}(M, w)$                 // Apply left preconditioner
    for $k \leftarrow 0$ to $i$ do                     // Orthonormalize w against
        $h_{k,i} \leftarrow w^H v_k$                  // all previous Arnoldi vectors
        $w \leftarrow w - h_{k,i} v_k$                // (Modified Gram-Schmidt method)
    end
    $h_{i+1,i} \leftarrow \|w\|$;
    $v_{i+1} \leftarrow w / h_{i+1,i}$ ;
    $y_i \leftarrow \arg\min(\|\beta e_1 - H_i y_i)\|$;  // Solve the inner least-square problem
    $r'_i \leftarrow \beta e_1 - H_i y_i$;              // Compute the estimated residual
    if $\|r'_i\| < \varepsilon$ then
        $x_i \leftarrow x_0 + \mathrm{solve}(M, V_i y_i)$;   // Update the solution
        break
    end
end
return $x = x_i$;
```

$H_\ell y_\ell\|$) are both lower than the target accuracy; that is $\exists\, 0 < \ell < n \;/\; \|r_\ell\| < \varepsilon$ and $\|r_\ell'\| = \|\beta e_1 - H_\ell y_\ell\| < \varepsilon$.

In practice, the computed residual norm is used to determine whether the execution is likely to have converged, and the true residual norm is used as a confirmation: the computed residual is a by-product of the algorithm and does not cost any floating point operation while the true residual norm calculation requires to compute the residual vector at an extra cost of a matrix-vector product. In exact arithmetic the norm of these two vectors are equal, but not necessary in finite precision calculation where some orthonormality might be lost in V_i.

3.2 Fault model

The evaluation focuses on transient faults that numerically affects the GMRES executions. Other errors introduced by transient faults such as memory access error or instruction modifications are supposed to be separately taken care of by low overhead techniques not described here. Hence, faults are modeled by a perturbation happening in a register which contains numerical data. Several types of perturbation can occur, so the fault model we define only uses bit-flips for simplicity. Moreover, it is assumed that no more than one fault may occur during the whole GMRES execution, as the occurrence of a fault is assumed to be a rare enough event compared to the execution time, and we will leave the two or more faults situation to the discussion section. Since the most expensive operation is the sparse-matrix vector product (line 6 in Algorithm **??**), it is assumed that the fault may only occur at this moment for at least two reasons:On one hand, most of the execution time will be spent performing this operation, which directly increases the probability of a fault happening there, and on the other hand, remaining operations can be duplicated without any significant impact on the execution time, enabling easy fault detection and error correction for those parts of the algorithm. Algorithm **??** is equivalent to the faulty product used in the following experiments. To simplify, it does not use any particular data structure and is based on the standard dense matrix-vector product. Hence it consists in two nested loops and we define the location (i, k) where the fault occurred as the indexes values of those loops at the moment of the fault. The operation repeated in the inner loop involves three registers, reg_1 stores $A_{i,k}$, reg_2 stores v_k and reg_3 stores $A_{i,k} \cdot v_k$. In the numerical experiments, double precision is used so those registers contains 64 bits that could be affected by one bit-flip. A faulty execution is then characterized by the following parameters :

- The **iteration** when the fault occurred
- The **bit flipped** ($b \in [0, 63]$ in double precision IEEE 754)
- The **location** in the product ($(i, k) \in [0, n-1]^2$)
- The **register** affected (reg_1, reg_2, reg_3)

3.3 Experiments setup

The following numerical experiments are split into two parts. The first one provides qualitative and illustrative results, and consists in executions performed on two matrices with carefully chosen fault parameters while the second one provides quantitative results, using several matrices and covering a large part of the fault parameter space.

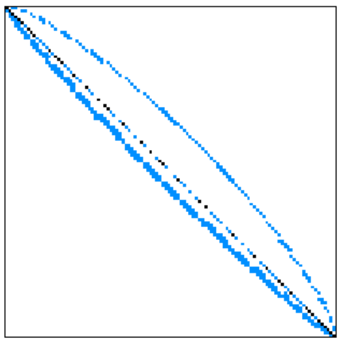
For the first part, the matrices used are HB/gre_216a (Figure **??**), chosen for its small size ($n = 216$) and handy convergence history, which will be used to illustrate the GMRES algorithm results, and HB/pores_2 (Figure **??**), which is bigger ($n = 1224$) and will be used

Algorithm 4: Faulty product $w \leftarrow \widetilde{A \cdot v}$

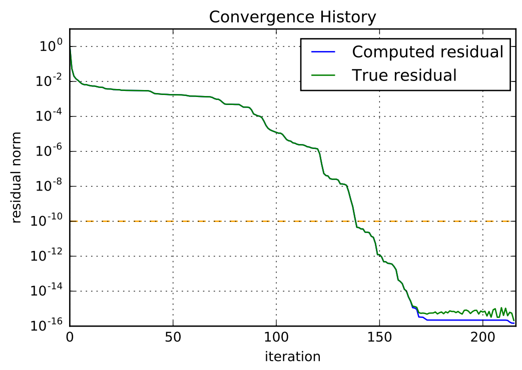
```
1  Faulty product $(A, v)$;
   Input     : $A$ ($n \times n$ Matrix), $v$ (size n vector)
   Parameter: fault parameters (location $I, K$, bit $B$, register $R$)
   Output    : $w$ so that $w = \widetilde{A \cdot v}$
   // SpMV product
2  for $i \leftarrow 0$ to $n-1$ do
3      $w_i \leftarrow 0$ ;
4      for $k \leftarrow 0$ to $n-1$ do
5          if $A_{i,k} \neq 0$ and $v_k \neq 0$ and $i == I$ and $k == K$ then
6              if $R == reg_1$ then $reg_1 \leftarrow \text{bitflip}((A_{i,k}, B)$ ;
7              else $reg_1 \leftarrow A_{i,k}$ ;
8              if $R == reg_2$ then $reg_2 \leftarrow \text{bitflip}(v_k, B)$ ;
9              else $reg_2 \leftarrow v_k$ ;
10             if $R == reg_3$ then $reg_3 \leftarrow \text{bitflip}(reg_1 \cdot reg_2, B)$ ;
11             else $reg_3 \leftarrow reg_1 \cdot reg_2$ ;
12             $w_i \leftarrow w_i + reg_3$ ;
13         end
14     end
15 end
16 return $w$;
```

with the preconditioned-GMRES algorithmes. The results will be put in parallel to illustrate the common behavior of those algorithms when they are submitted to soft errors. When left-preconditioned-GMRES and right-preconditioned-GMRES present the same behavior, only the result of right-preconditioned-GMRES are displayed. For the second part, the matrices used are gathered in Table **??**.

All matrices can be found in the University of Florida Sparse Matrix Collection [**?**].

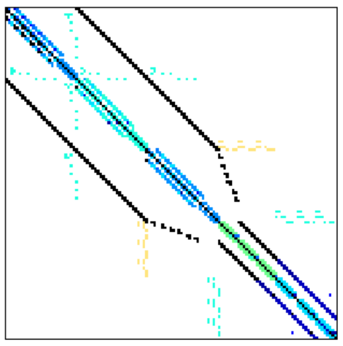


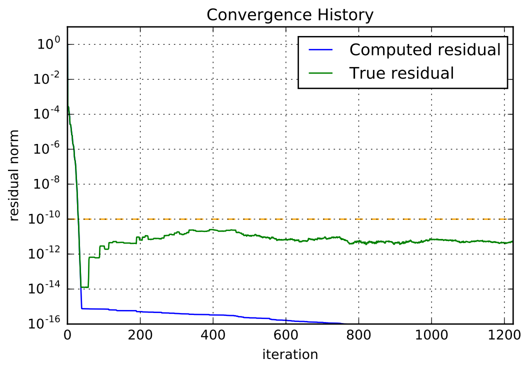
(a)



(b)

Figure 1. Matrix HB/gre_216a and its GMRES convergence history.





(a) (b)

Figure 2. Matrix HB/pores_2 and its preconditioned-GMRES convergence history.

Table 1. Matrices used

Name	Size (n)	Non-Zeros (%)	type
HB/arc130	130	0.061	real unsymmetric
Bai/lop163	163	0.035	real unsymmetric
HB/mcca	183	0.082	real unsymmetric
Rajat/rajat14	180	0.046	real unsymmetric
HB/fs_183_1	183	0.030	real unsymmetric
Bai/rdb200	200	0.028	real unsymmetric
JGD_Trefethen/Trefethen_200	200	0.072	real unsymmetric
HB/impcol_a	207	0.013	real unsymmetric
HB/gre_216a	216	0.017	real unsymmetric
HB/steam1	240	0.039	real unsymmetric

3.4 Representation of executions

Since an execution can be represented in many different ways, we propose in the following two visualizations enabling the key information to be displayed. The main purpose of the GMRES algorithm is to reduce the residual norm to the target accuracy, hence we use its convergence history, referring to the residual norm evolution through the iterations. Figure **??** plots the convergence history of a GMRES execution on gre_216a, disrupted by a transient fault, as well as the same non-faulty execution for reference. We first observe that the true residual in the non-faulty execution (dashed blue curve) converges in 150 iterations since it reaches the target accuracy ($\varepsilon = 10^{-10}$) at iteration 150. Then, the computed residual in the faulty execution also converges but much later (around iteration 195). On the contrary, the true residual in the faulty execution never reaches the target accuracy, as it remains greater than 10^{-7}, so according to the definition, the faulty execution fails to converge. The fault (occurring at iteration 70) is responsible for this failure, and it can also be seen that after the fault occurrence, the residual norm decreases more slowly in the faulty execution, which imply more iterations than the reference and potentially a delayed solution (correct or incorrect).

The second way used to represent an execution is simply by its outcome. For instance, Table **??** represents a set of outcomes used later in this report.

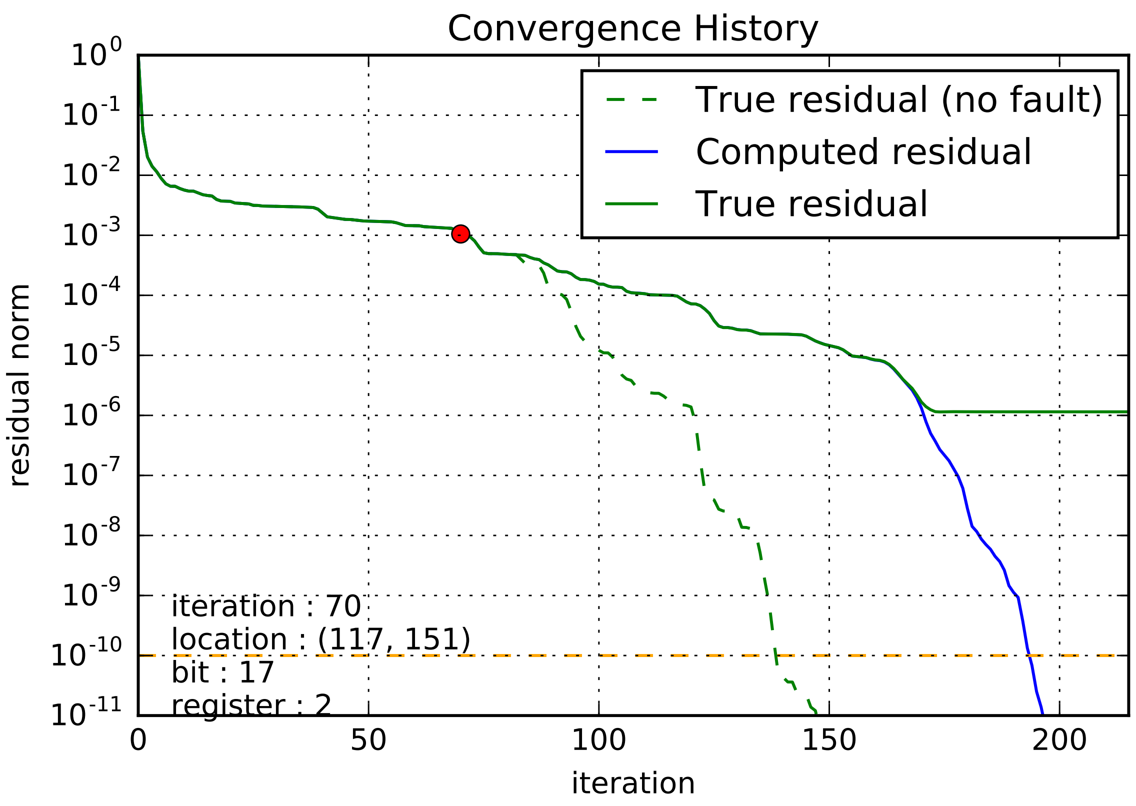


Figure 3. Convergence history of a GMRES execution on gre_216a disrupted by a transient fault (red circle). The Y axis corresponds to the residual norm, and the X axis is the iteration where it was measured. The dashed green line corresponds to the true residual norm in the reference non-faulty execution, the full green line is the true residual norm in the faulty execution (the fault parameters are displayed at the bottom left hand corner) and the blue line is the computed residual norm in the faulty execution.

Table 2. Color used for each test outcome.

Outcome
Convergence without delay
Convergence with delay
No convergence

4 Fault impact analysis

In this section, the impact of faults on the convergence of GMRES executions is studied. The impact is first observed both qualitatively and quantitatively on a few cases and according to the previously described experimental setup, and is then analysed from a more theoretical point of view.

4.1 Empirical study of executions disrupted by faults and fault parameters influence on the convergence

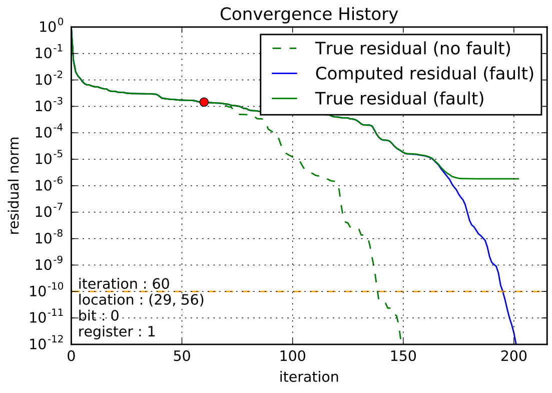
To visualize the impact of soft-errors on the convergence, different faults are injected into GMRES executions. First qualitative results on a few cases are given to illustrate in detail each fault parameter, and then, quantitative experiments using several matrices are performed to evaluate more extensively the fault effects on GMRES executions.

4.1.1 Qualitative results and fault parameters influence

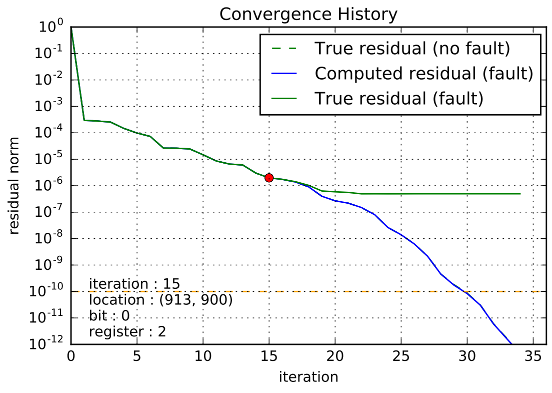
To visualize the influence of each fault parameter, the convergence history of 3 disrupted GMRES executions over the same matrices (HB/gre_216a for GMRES and HB/pores_2 for preconditioned-GMRES) are plotted for each one of the 4 fault parameters (bit flipped, iteration, register and location) while the others are fixed. Since both variants of preconditioned-GMRES give similar results, only the right-preconditioned-GMRES ones are displayed.

In Figure **??** all parameters but the bit flipped are fixed. It can be observed that bit-flips on the most significant bits have a stronger impact on the convergence than bit-flips on the least significant bits: in Figure **??**, the bit flipped is one of the register's least significant bit (bit 60) and the fault does not prevent the convergence. In Figure **??**, the 40^{th} bit is flipped and the convergence is delayed by a dozen iterations. In Figure **??** the 20^{th} bit is flipped and the execution never converges to the target accuracy. Intuitively, bit flips on the most significant bits introduce bigger errors and are more likely to disrupt the convergence than bit flips on the least significant bits.

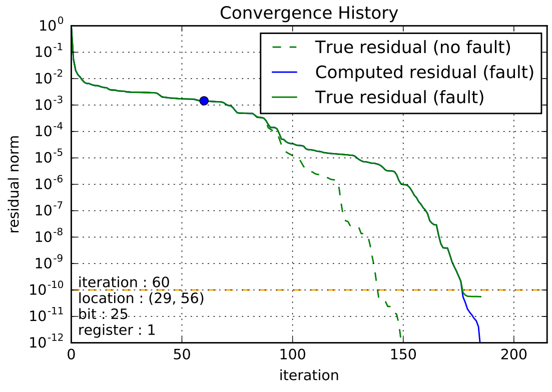
GMRES executions on **gre_216a** **preconditioned-GMRES** on **pores_2**



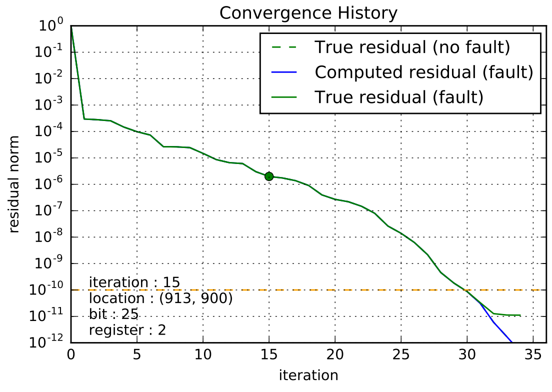
(a) bit 0



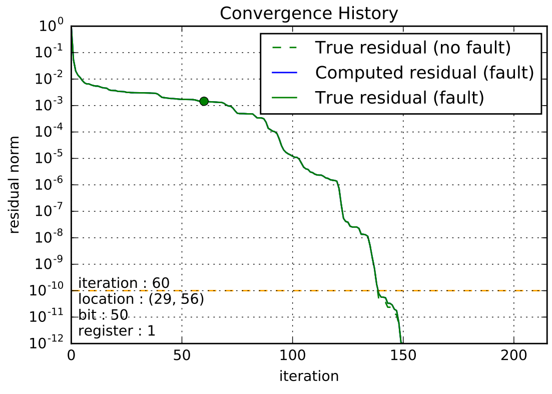
(d) bit 0



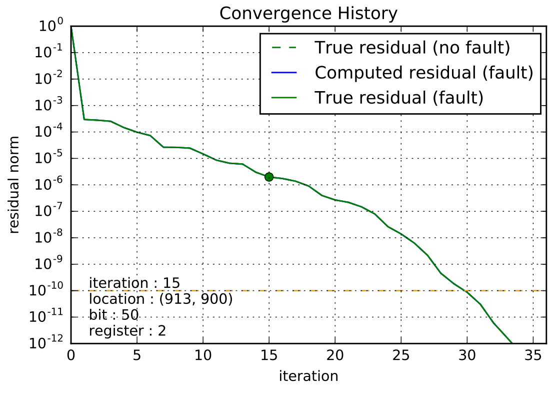
(b) bit 25



(e) bit 25



(c) bit 50

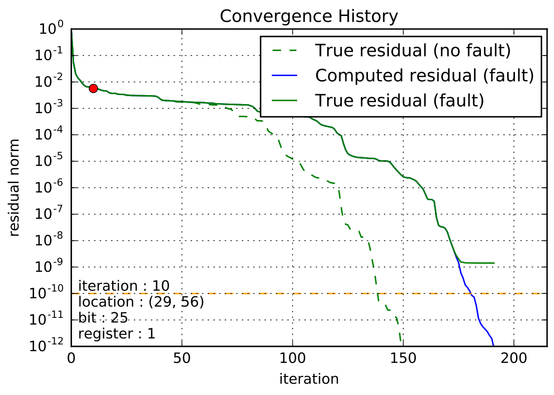


(f) bit 50

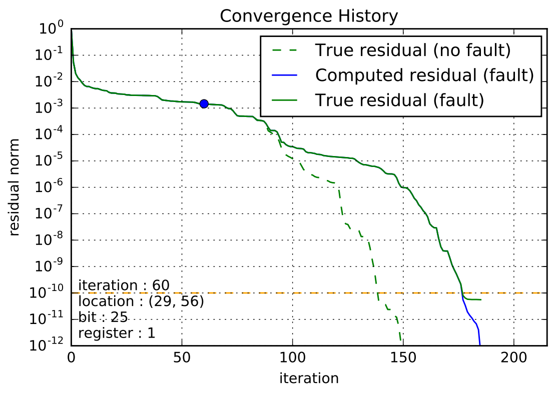
Figure 4. Convergence history of 3 GMRES executions on gre_216a and 3 preconditioned-GMRES executions on pores_2, disrupted by a transient fault. In the execution **??**, the fault (**bit 0**) disrupts the convergence, in **??**, the fault (**bit 25**) delays it, and in **??**, the fault (**bit 50**) has no effect on the convergence. In both cases, bit-flips on the most significant bits have a larger impact than bit-flips on the least significant bits.

In Figure **??**, all parameters but the iteration when the fault occurs are fixed. It can be observed in both GMRES and preconditioned-GMRES that the sooner the fault occurred, the bigger the impact on convergence seems to be: in Figure **??**, the fault occurred on the 1^{st} iteration and the execution did not converge, in Figure **??**, the fault occurs on the 61^{st} iteration and the convergence is delayed, and in Figure **??** the fault occurs on the 121^{st} iteration and the execution converges to the target accuracy. This observations complies with the intuition that the first iterations are responsible for the main part of the solution, while latter ones are responsible for its refinement, hence faults are more likely to be critical when occurring at the early stages of the execution than towards the end.

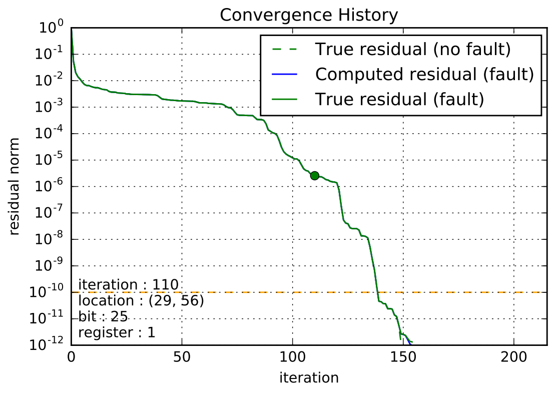
GMRES executions on **gre_216a**



(a) iteration 10

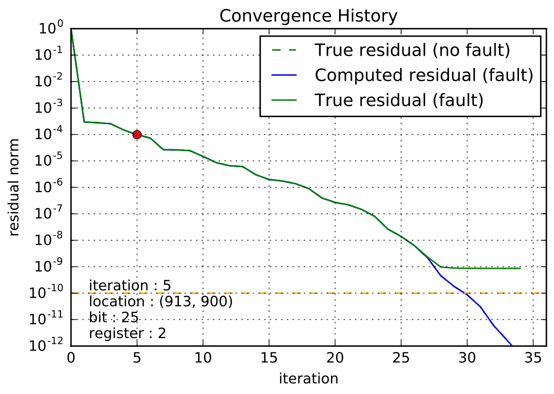


(b) iteration 60

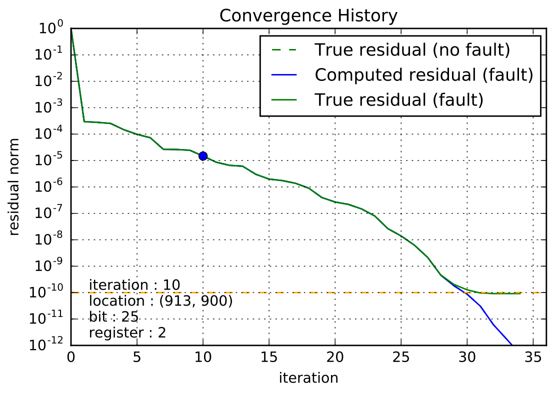


(c) iteration 110

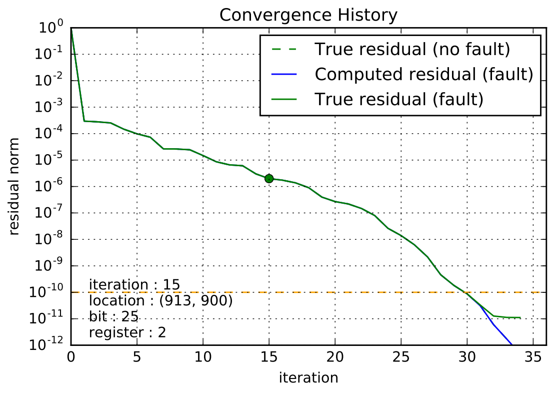
preconditioned-GMRES on **pores_2**



(d) iteration 5



(e) iteration 10

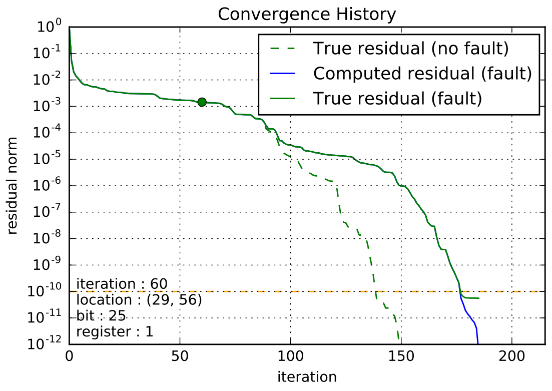


(f) iteration 15

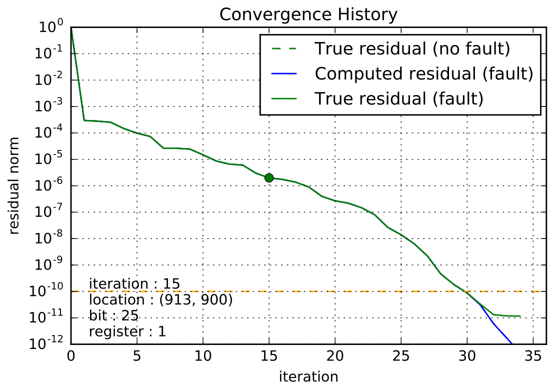
Figure 5. Convergence history of 3 GMRES executions on gre_216a and 3 preconditioned-GMRES executions on pores_2, disrupted by a transient fault. In the execution **??**, the fault (**iteration 10**) disrupts the convergence, in **??**, the fault (**iteration 60**) delays it, and in **??**, the fault (**iteration 110**) has no effect on the convergence. In both cases, the fault impact decreases as the iteration when it is injected increases.

The remaining fault parameters (register and location) do not appear to have any significant impact on the convergence, even though inputs can be constructed to contradict this observation. Figure **??**, representing the convergence history of 3 GMRES executions and 3 preconditioned-GMRES disrupted by a bit-flip on reg_1, reg_2 and reg_3, and Figure **??**, representing the convergence history of 3 GMRES executions and 3 preconditioned-GMRES disrupted by a bit-flip on three different locations, present similar behaviors in terms of convergence history.

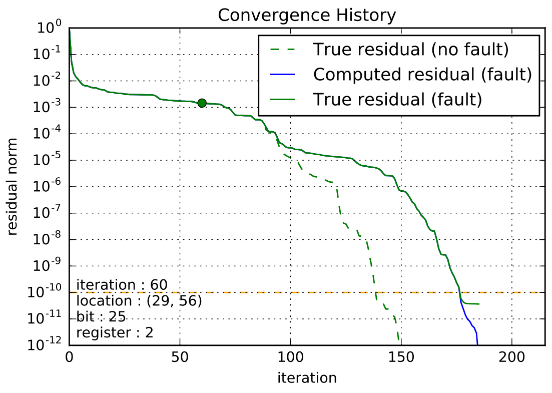
GMRES executions on **gre_216a** **preconditioned-GMRES** on **pores_2**



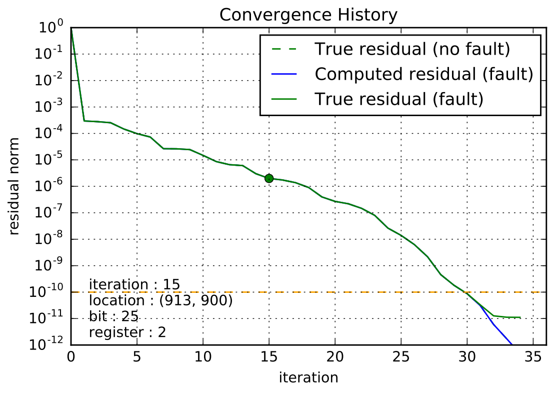
(a) reg_1



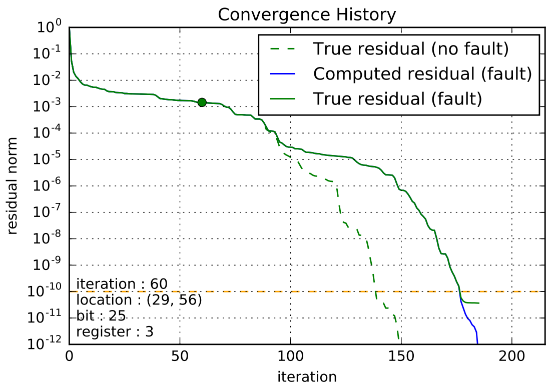
(d) reg_1



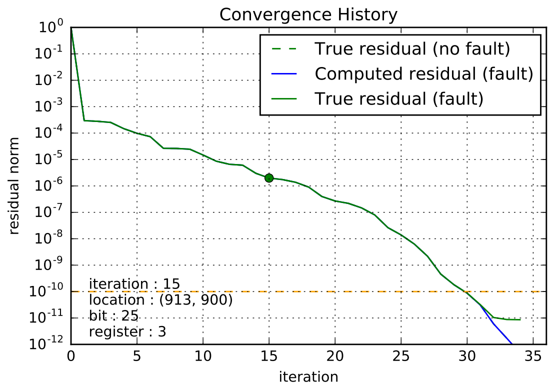
(b) reg_2



(e) reg_2



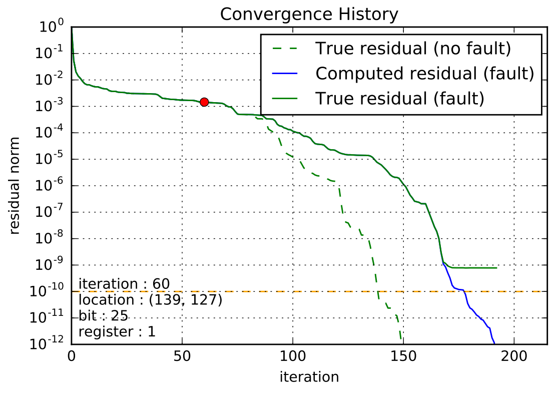
(c) reg_3



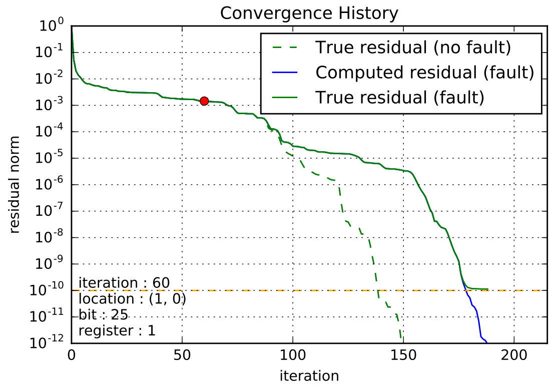
(f) reg_3

Figure 6. Convergence history of 3 GMRES executions on gre_216a and 3 preconditioned-GMRES executions on pores_2, disrupted by a transient fault. In all the executions, the faults (occurring in reg_1, reg_2 and reg_3) seem to have the same impact.

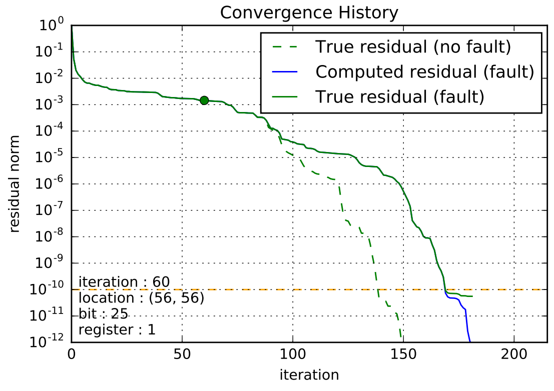
GMRES executions on **gre_216a**



(a)

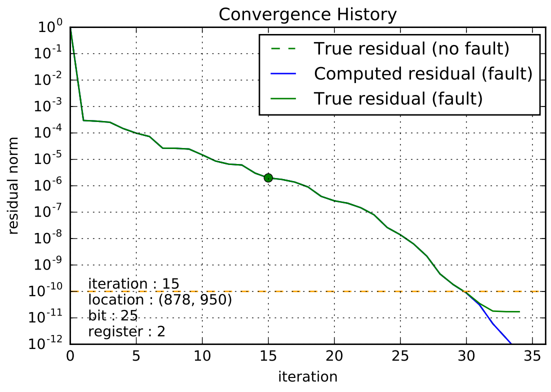


(b)

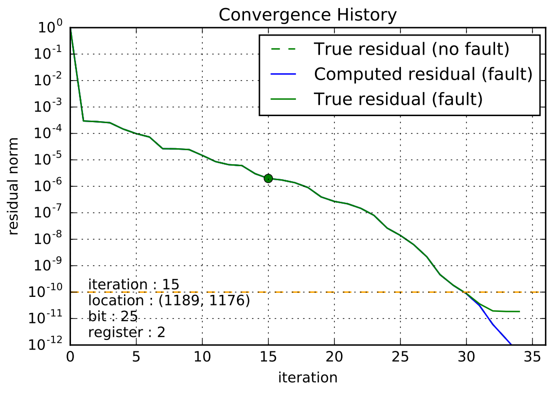


(c)

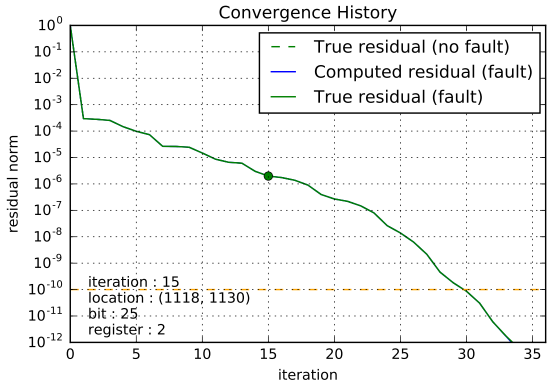
preconditioned-GMRES on **pores_2**



(d)



(e)



(f)

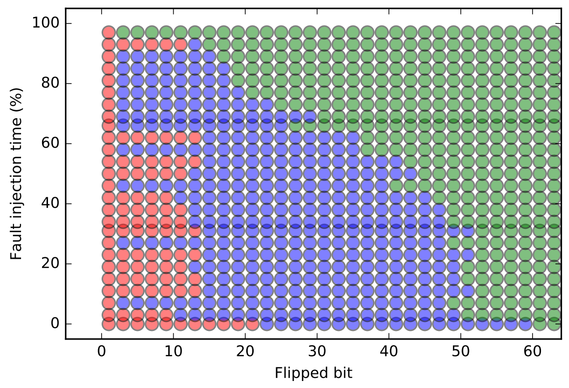
Figure 7. Convergence history of 3 GMRES executions on gre_216a and 3 preconditioned-GMRES executions on pores_2, disrupted by a transient fault. In all the executions, the faults (occurring in random locations) seem to have the same impact.

In summary, only the bit and the iteration where the bitflip occurs seems to have a significant impact on the convergence. For this reason, the fault parameter space to cover is reduced to those two parameters, simplifying the quantitative results presented in the next section.

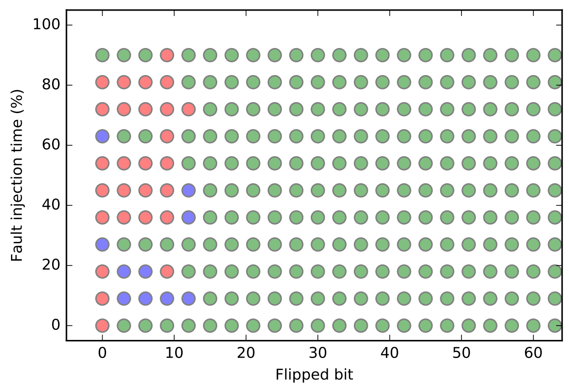
4.1.2 Quantitative results

To illustrate the previous observations, Figure **??** presents the convergence of several faulty-executions - one for each couple (iteration fault $\times$ bit flipped) while the other parameters are chosen randomly - for GMRES on HB/gre_216a and right-preconditioned-GMRES on HB/pores_2. Some faults have no effect on the convergence (in green), some may only delay it (blue), and others may prevent the execution from converging to the target accuracy (red). As observed in the previous results, faults involving the most significant bits of the registers or happening soon in the execution are more likely to cause a failure than faults involving the least significant bits or happening late in the execution.

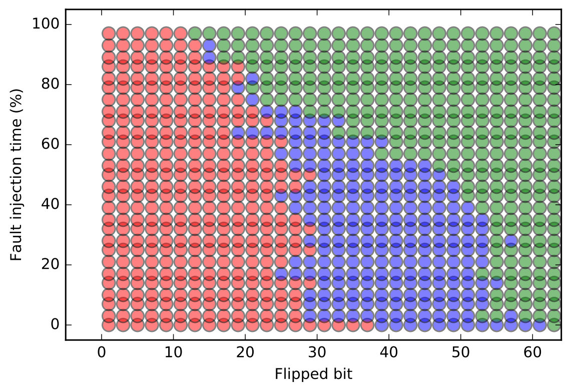
GMRES executions on **gre_216a** **preconditioned-GMRES** on **pores_2**



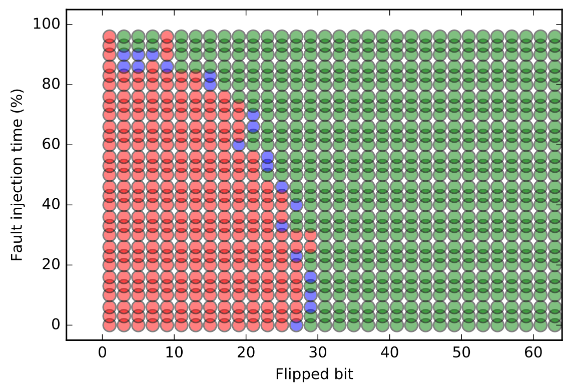
(a) $\varepsilon = 10^{-5}$



(b) $\varepsilon = 10^{-5}$



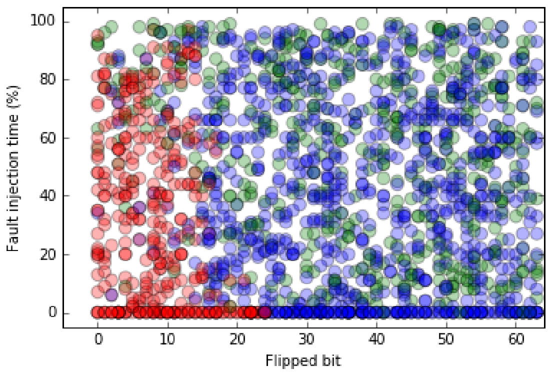
(c) $\varepsilon = 10^{-10}$



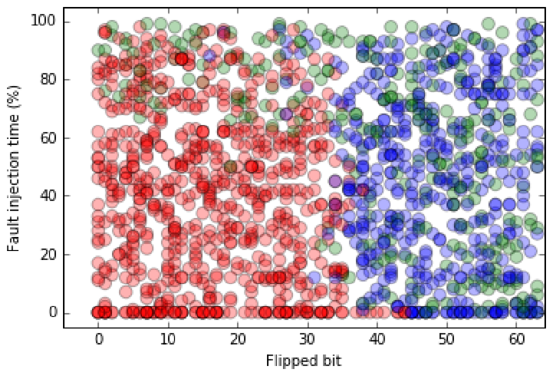
(d) $\varepsilon = 10^{-10}$

Figure 8. Diagram of executions disrupted by a fault. Each dot represents an execution of GMRES on HB/gre_216a (figures **??** and **??**) or preconditioned-GMRES on HB/pores_2 (figures **??** and **??**), its abscissa is the bit where the fault occurred, its ordinate is the moment in the execution when the fault has been injected and its color represents whether the execution has converged or not (green: no delay, red: no convergence, blue: delayed convergence).

As shown in Figure **??**, the behavior of faulty GMRES is very similar for all the selected matrices: the left-hand side of the graphs corresponding to executions disrupted by bit-flips of the most significant bit mostly fail to converge, except for the latest iterations, while the right-hand side of the graphs corresponding to executions where the least significant bits are flipped usually converges.



(a) $\varepsilon = 10^{-5}$



(b) $\varepsilon = 10^{-10}$

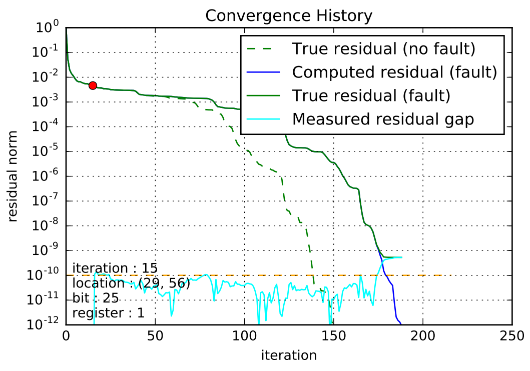
Figure 9. Diagrams of GMRES executions disrupted by a fault. Each dot represents an execution on any matrix of the dataset, its abscissa is the bit where the fault occurred, its ordinate is the moment in the execution when the fault has been injected and its color represents whether the execution has converged or not (green: no delay, red: no convergence, blue: delayed convergence).

4.2 Analytical study of the fault impact on the convergence

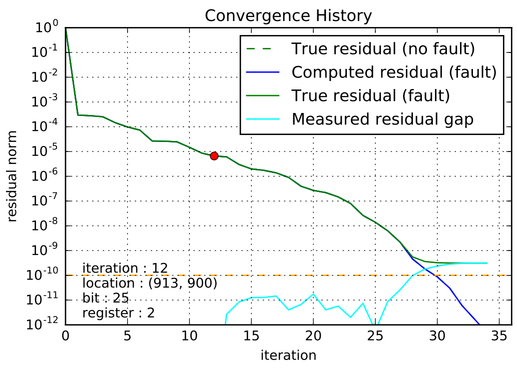
In the previous experiments, we showed that a transient fault may have a significant impact on the convergence of an execution. In the following, an analysis is proposed to evaluate this impact using the theory of inexact GMRES, in order to intent to predict the convergence behavior in **??**.

4.2.1 Error quantification

Because the sequence of least-square problems solved along the iteration are nested; that is at iteration i the leading part of H_{i+1} is H_i as well as for the right-hand sides; we know that independently of the possible occurrence of a fault during the execution, GMRES always reduces the computed residual norm down to the machine accuracy, as long as it is given enough iterations to do so. However, a fault may introduce a large residual deviation between the computed residual and the true residual (i.e., $\|\widetilde{r}_\ell - \widetilde{r}_\ell'\|$ becomes large), which may prevent the latter one from reaching the target accuracy. Since $\|\widetilde{r}_\ell\| \geq \|\widetilde{r}_\ell - \widetilde{r}_\ell'\| - \|\widetilde{r}_\ell'\|$ and $\|\widetilde{r}_\ell'\| \to 0$, we obtain that $\|\widetilde{r}_\ell\| \to \|\widetilde{r}_\ell - \widetilde{r}_\ell'\|$, so if the error remains greater than the target accuracy ε, the execution will not converge. The error measured at iteration ℓ is called the residual gap at iteration ℓ, denoted $\delta(\ell)$. We illustrate in Figure **??** the effect of a soft-error on the norm of the residuals and the measured residual gap $\|\widetilde{r}_i - \widetilde{r}_i'\|$. Before the fault, the residual gap is close to the machine accuracy. After the fault and until the end of the execution, the residual gap fluctuates and gets closer to the true residual norm (green line) as the computed residual norm (blue line) converges to the machine accuracy. The dashed green line shows the non-faulty execution residuals, to illustrate the delay induced by the error in the convergence. The residual gap provides a good quantification of the fault impact, however it does not explain nor quantifies how much the convergence is delayed (when the convergence occurs).



(a) GMRES on gre_216a



(b) preconditioned-GMRES on pores_2,

Figure 10. Two convergence histories disrupted by a fault. The light blue line represents the residual gap evolution in the faulty executions:GMRES on gre_216a in **??** and precondition-GMRES on pores_2 in **??**.

In [**?**], the authors provide a general framework for the understanding of inexact Krylov subspaces, generated by inexact matrix vector products that arise naturally in many scientific applications. In particular, they analyze the residual norm deviation assuming that a perturbation E_i is applied to the SpMV product at each iteration i.

$$\widetilde{r}_\ell = \widetilde{r}_\ell{}' + \widetilde{y}_\ell \cdot \sum_{i=1}^{\ell} E_i \cdot v_i \tag{2}$$

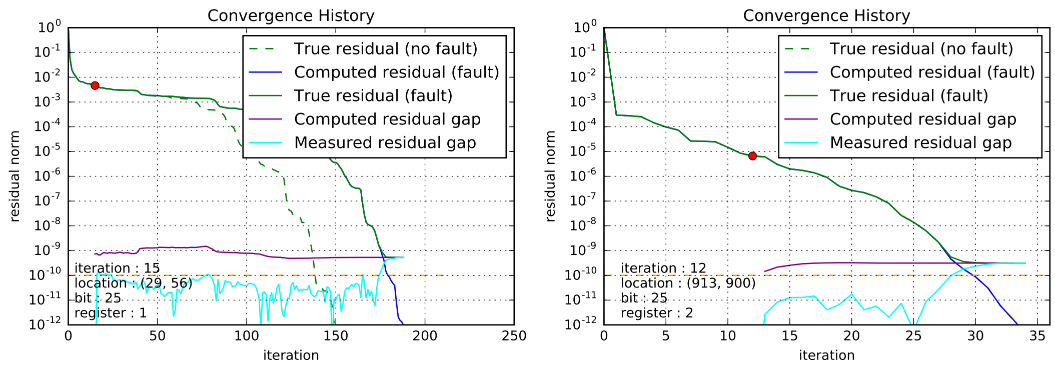
We can use this analysis in a somehow simpler situation where a transient fault is considered as a perturbation E_f that only occurs once at the f^{th} iteration, specializing the results in [**?**] to be applicable in our model.

$$\|\widetilde{r}_\ell - \widetilde{r}_\ell{}'\| = \|\widetilde{y}_\ell \cdot E_f \cdot v_f\| \tag{3}$$

This enables us to get the following equation that links the residual gap to $\widetilde{y}_{\ell,f}$ and to the error introduced by the fault $\|E_f \cdot v_f\| = \|A \cdot v_f - \widetilde{A} \cdot v_f\| = \|w_{f+1} - \widetilde{w}_{f+1}\|$:

$$\delta(\ell) = \|\widetilde{r}_\ell - \widetilde{r}_\ell{}'\| = |\widetilde{y}_{\ell,f}| \cdot \|w_{f+1} - \widetilde{w}_{f+1}\| \tag{4}$$

where w_{f+1} is the $f+1^{th}$ Arnoldi vector before orthonormalization and $\widetilde{y}_{\ell,f}$ is the f^{th} component of the ℓ^{th} least square problem solution $\widetilde{y}_\ell = \arg\min_{\mathbf{y}} \|\beta e_1 - H_\ell \mathbf{y}\|$. In the following, the quantity $|\widetilde{y}_{i,f}| \cdot \|w_{f+1} - \widetilde{w}_{f+1}\|$ is called the computed residual gap at iteration i, in constrast with the measured residual gap at iteration i $\|\widetilde{r}_i - \widetilde{r}_i{}'\|$. These quantities are equal when $i = \ell$.



(a) GMRES on gre_216a **(b)** preconditioned-GMRES on pores_2,

Figure 11. Two convergence histories disrupted by a fault. The light blue and purple lines represent the measured residual gap ($\|\widetilde{r}_i - \widetilde{r}_i'\|$) and the computed residual gap ($|\widetilde{y}_{i,f}|\cdot\|w_{f+1} - \widetilde{w}_{f+1}\|$) evolutions in the faulty executions:GMRES on gre_216a in **??** and precondition-GMRES on pores_2 in **??**.

4.2.2 Prediction of the fault impact

From Section **??**, we observed that faults may either have a critical impact when preventing the execution from converging, little impact when delaying the convergence, or no impact at all. Ideally, by using the symptoms brought by a fault we should be able to predict its impact. Theorem **??** below proposes a possible suited criterion in the form of a threshold on the error $\|w_{f+1} - \widetilde{w}_{f+1}\|$ to ensure that a fault is not critical. It is based on Equation (**??**), itself relying on the remark that a faulty matrix-vector product caused by a bit-flip can be interpreted as an inexact matrix-vector product in inexact [**?**] or relaxed [**?**] GMRES.

Theorem 1. *Let* $0 < c < 1$ *and let a GMRES execution using* $(1-c)\varepsilon$ *as the target accuracy, terminating in* ℓ *iterations and disrupted by a fault at iteration* $f < \ell$*. In exact arithmetic, if* $\|w_{f+1} - \widetilde{w}_{f+1}\| < \tau_c = (c\cdot\varepsilon)/|\widetilde{y}_{\ell,f}|$ *then the execution converges to the target accuracy* ε*.*

Proof. The purpose of the c parameter is to bound the true residual norm as a convex combination of the residual deviation norm and the computed residual norm. Since the GMRES execution terminates in ℓ iterations for the target accuracy $(1-c)\varepsilon$

$$\|\widetilde{r}'_\ell\| \leq (1-c)\varepsilon$$

Then, using Equation (**??**) derived from [**?**] and the criterion $\|w_{f+1} - \widetilde{w}_{f+1}\| < \frac{c\cdot\varepsilon}{|\widetilde{y}_{\ell,f}|}$:

$$\begin{aligned}\|\widetilde{r}_\ell\| &\leq \|\widetilde{r}_\ell - \widetilde{r}'_\ell\| + \|\widetilde{r}'_\ell\| \\ &\leq \|y_{\ell,f}(w_{f+1} - \widetilde{w}_{f+1})\| + \|\widetilde{r}'_\ell\| \\ &\leq |y_{\ell,f}|\cdot\|w_{f+1} - \widetilde{w}_{f+1}\| + \|\widetilde{r}'_\ell\| \\ &\leq c\cdot\varepsilon + (1-c)\cdot\varepsilon = \varepsilon\end{aligned} \tag{5}$$

The GMRES execution has converged with the target accuracy ε. □

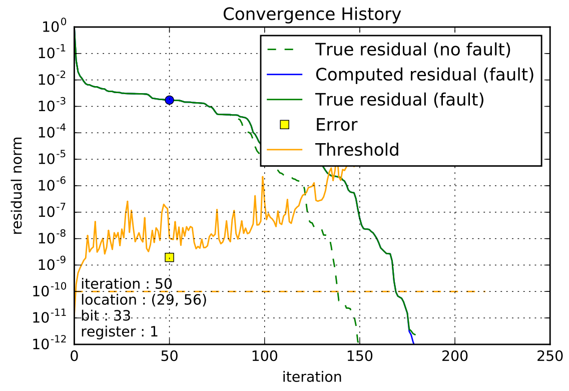
4.2.3 Numerical experiment

To verify if Theorem **??** is applicable, even in floating point arithmetic instead of exact arithmetic, the following numerical experiments are performed. Let $0 < c < 1$, and let the target accuracy be reduced downto $(1 - c) \cdot \varepsilon$. Using Theorem **??**, we can predict if the fault will not disrupt the convergence to ε by computing for all the executions the following expression at the fault iteration f:

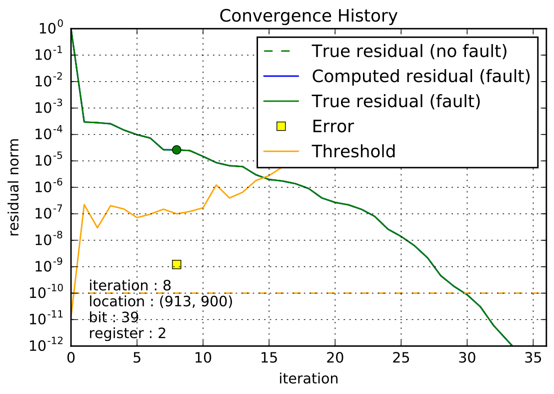
$$\text{error} = \|w_{f+1} - \widetilde{w}_{f+1}\| < \tau_c = (c \cdot \varepsilon)/|\widetilde{y}_{\ell,f}| \tag{6}$$

If the error introduced by the fault in lower than the threshold, the execution must converge (and then is predicted to converge) to the target accuracy ε according to Theorem **??**. Note that the reverse is not necessarily true so if the error is greater than the threshold, the execution may or may not converge to the target accuracy (no prediction). Figure **??** illustrates this idea.

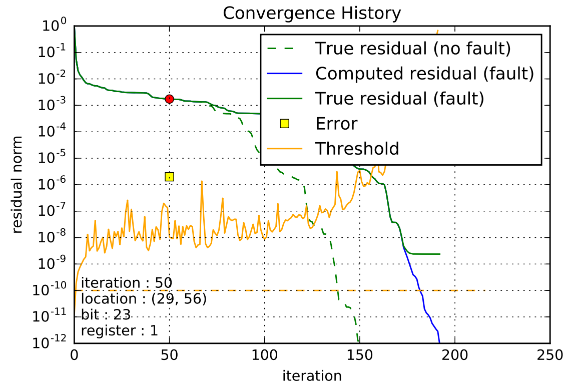
GMRES executions on **gre_216a** **preconditioned-GMRES** on **pores_2**



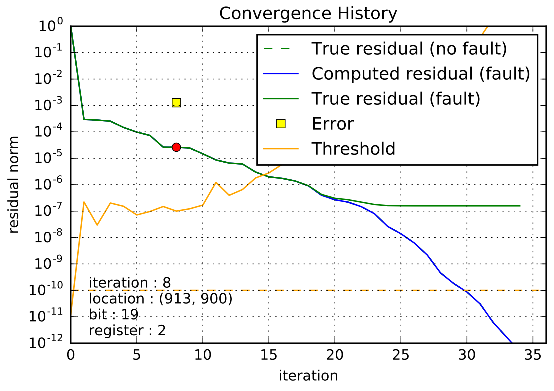
(a) Convergence prediction



(c) Convergence prediction



(b) No prediction

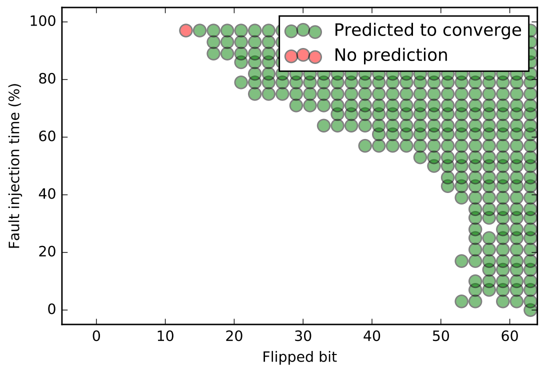


(d) No prediction

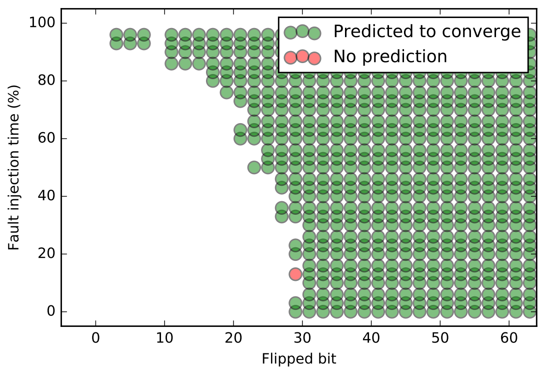
Figure 12. Prediction of the convergence in 4 executions disrupted by a fault on gre_216a and pores_2. At the fault iteration f, the error is compared to the threshold. In the upper plots, the error is lower than the threshold, hence the execution is predicted to converge. On the contrary in the bottom plots, the error is greater than the threshold so no convergence prediction can be made (although in both case the executions happen to converge anyway). For the prediction, the value $c = 0.5$ was used.

Figure **??** displays the prediction result for several executions, grouped by convergence outcome, and depending on the fault properties (bit flipped and injection time). Every execution that was predicted to converged (green) has indeed converged to the target accuracy (top and middle plots), since no execution that did not converge (bottom plots) has been predicted to. Despite the use of floating point arithmetic instead of exact arithmetic, Theorem **??** still gives the expected results. For the executions that did converge, the prediction is correct for a large part of cases, except for those closer to non-convergence. In particular, the proportion of predicted convergence is lower when the convergence was delayed.

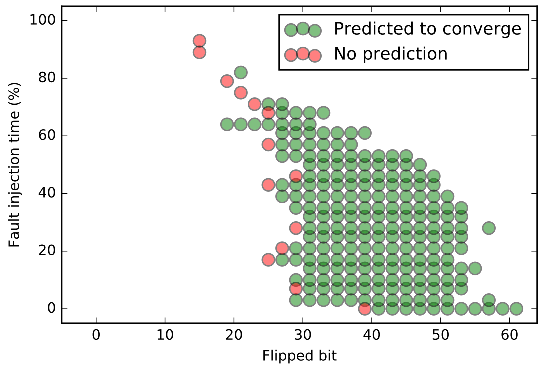
GMRES executions on **gre_216a** **preconditioned-GMRES** on **pores_2**



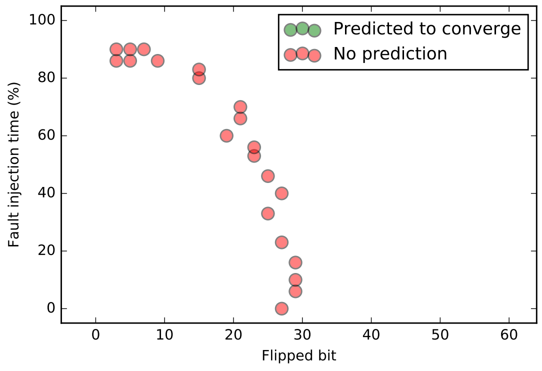
(a) Converged



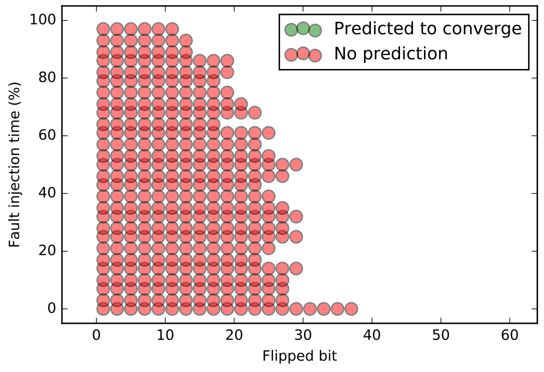
(d) Converged



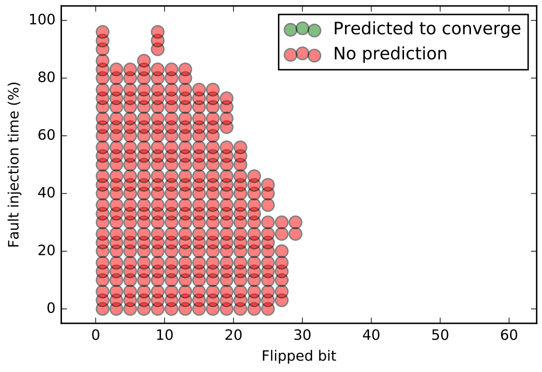
(b) Delayed



(e) Delayed



(c) Not converged



(f) Not converged

Figure 13. Prediction of the convergence in several executions disrupted by a fault. Each dot corresponds to an execution, its abscissa is the bit flipped, its ordinate is the fault injection time in the execution, and its color represents whether the execution was predicted to converge (green) or no prediction was made (red). The executions are split into 3 plots for each matrix used, one for each convergence outcome (convergence, delayed convergence and no convergence). For the prediction, the value $c = 0.5$ was used.

5 Detection of faults in GMRES

In Section **??**, we described a way to systematically predict when a fault is not critical (i.e. does not prevent the execution from converging) by assessing its symptoms on the execution variables. In this section, we now use this result to propose and evaluate a fault detection scheme. In Section **??**, the method is described, then each outcome is illustrated, and finally is evaluated extensively. In Section **??**, approximations are proposed for enabling the fault detection scheme implementation, then this new practical detection scheme is evaluated, as well as each approximation applied separately.

5.1 An oracle-based detection scheme

5.1.1 Description of the detection scheme

From the ability to systematically predict if a fault is non-critical, a fault detection scheme can easily be derived by ignoring faults that we are sure will not disrupt the convergence, and by triggering the detection otherwise.

Let $0 < c < 1$. This detection scheme would be as follow:

1. Set the target accuracy to $(1 - c) \cdot \varepsilon$.
2. At the fault iteration f, perform the following test:

$$error = \|w_{f+1} - \widetilde{w}_{f+1}\| < \tau_c = c \cdot \varepsilon / |y_{\ell,f}| \tag{7}$$

3. If the error is smaller than the threshold, the fault is ignored as it does not threaten the convergence, otherwise, a detection is triggered.

Table **??** gathers the 4 possible outcomes produced by this detection scheme. If the execution converges, the fault is said to have no impact, whereas a fault preventing the execution from converging is said to be critical.

Table 3. Colors and name used for each test outcome.

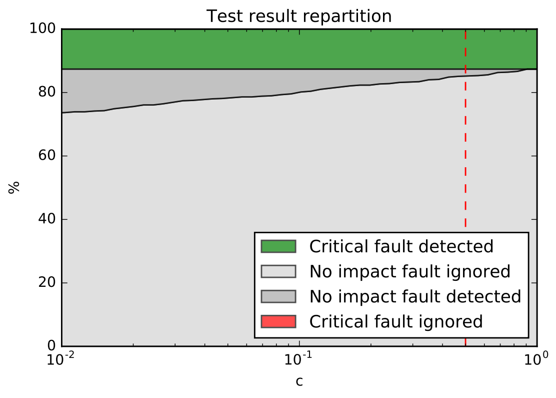
	Convergence	No convergence
Detection	**No impact fault detected**	**Critical fault detected**
No detection	**No impact fault ignored**	**Critical fault ignored**

Figure **??** plots 2 convergence histories for each algorithm. In figures **??** and **??**, no fault is detected as the error is smaller than the threshold, and the executions converges, producing a *No impact fault ignored* from Table **??**. In figures **??** and **??**, a critical fault is detected as the error is greater than the threshold $\tau_{0.5}$ at the fault iteration and the executions does not converge, producing a *Critical fault detected* from Table **??**. In both cases, the fault is properly detected.

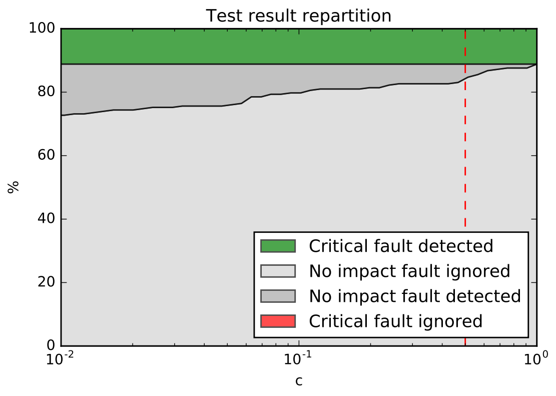
5.1.2 Evaluation of the detection scheme

The remaining parameter to be chosen is c. One intuitive way to understand this parameter's role is as follow: in order to control the accuracy of the true residual norm, we split it into two quantities easier to control. On the one hand, the computed residual norm can be adjusted by setting the target accuracy of the execution to the desired value, at the expense of additional computation, and on the other hand the residual gap can be measured and allowed to grow before triggering a detection. However those two values cannot be adjusted independently to ensure the true residual norm convergence to the target accuracy, so the c parameter aims at balancing the importance of one or the other value (computed residual or residual gap). So if one wishes to decrease the sensitivity of the detection scheme, he should increase the c value to allow the residual gap to grow larger before triggering a detection, at the expense of more iterations to reduce the computed residual norm. On the contrary if one does not want to use too much additional iterations to reduce the computed residual norm, he gets a more sensitive detection scheme that is efficient for detecting critical faults, but may also detect faults that do not have any impact on the convergence. In the following, numerical experiments are performed to evaluate the detection scheme quality. Figure **??** displays the fault detection scheme results (in percentage of executions) for several values of c. First, no critical faults are ignored, which is a direct consequence of Theorem **??**: if a fault is ignored (the error is smaller than the threshold), then the execution must converge. Second, for low values of c, an important part (between 20% and 35%) of executions disrupted by a negligible fault detect it anyway (dark gray) as the detection scheme is more sensitive, but it becomes more and more capable of ignoring those when c increases. The rest of the faulty executions (green and light gray) are correctly handled. For more details, the $c = 0.5$ cases are plotted in Figure **??**. In each case, a large majority of the faulty executions correctly detects the faults. The rare cases of unneeded detection (dark gray) occur at the limit of the convergence and non convergence domain (light gray and green respectively). Overall, the detection quality is very good as all critical faults have been properly detected, and only a few negligible faults were unnecessarily detected. Indeed, even though we do not have the reverse of Theorem **??** (which would ensure that *only* executions in which faults are ignored by the detection scheme does converge, hence removing the dark gray outcomes), this numerical experiment shows that the reverse is not far from being true.

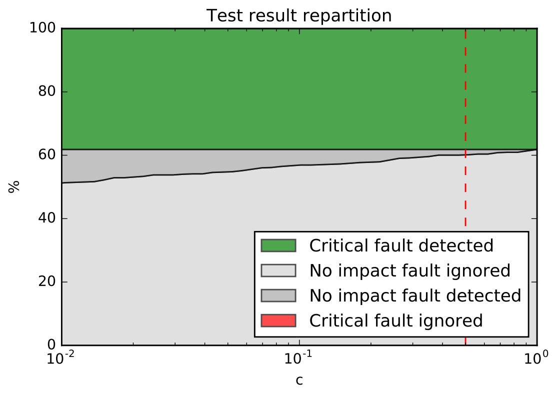
GMRES executions on **gre_216a** **preconditioned-GMRES** on **pores_2**



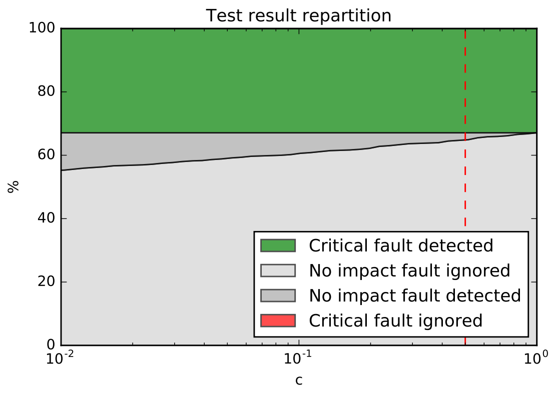
(a) $\varepsilon = 10^{-5}$



(c) $\varepsilon = 10^{-5}$



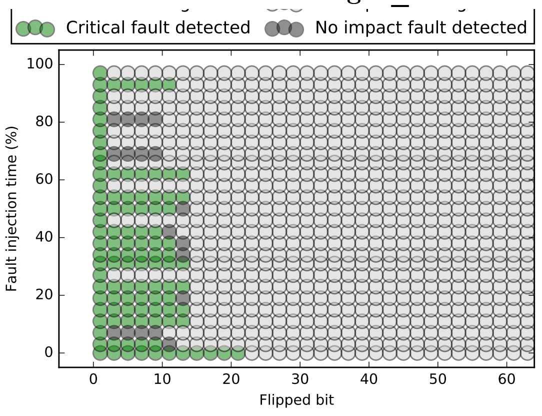
(b) $\varepsilon = 10^{-10}$



, **(d)** $\varepsilon = 10^{-10}$

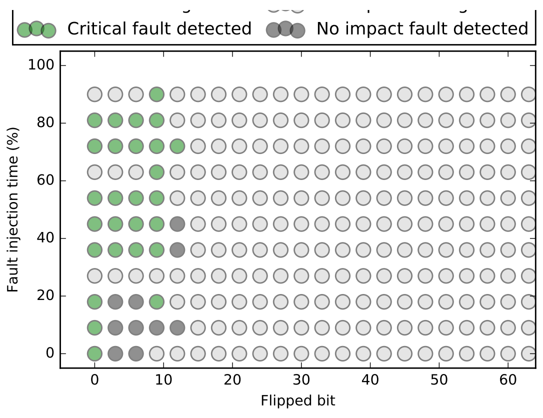
Figure 14. Diagrams representing the test outcome proportion for several values of c. To compute them, faulty executions covering a large part of the fault parameter space were performed for several values of c. The case $c = 0.5$ represented by the dashed red line is detailed in Figure **??**.

GMRES executions on **gre_216a**

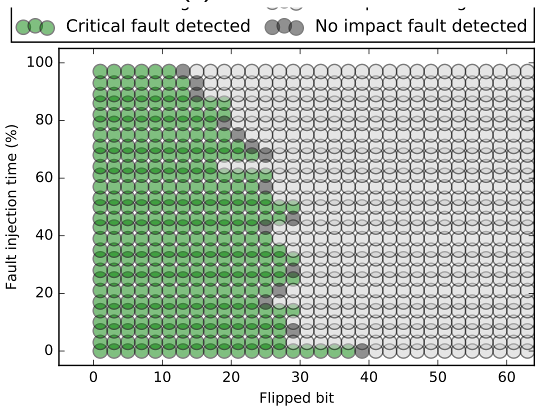


(a) $\varepsilon = 10^{-5}$

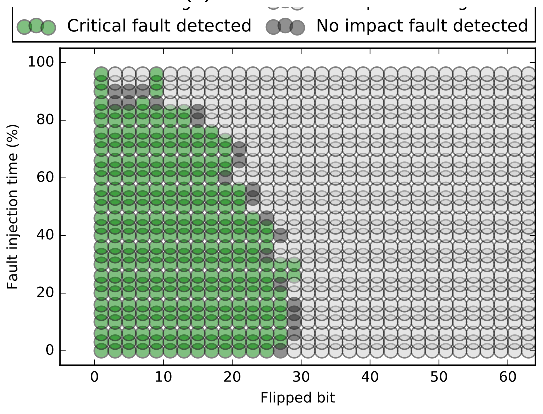
preconditioned-GMRES on **pores_2**



(c) $\varepsilon = 10^{-5}$



(b) $\varepsilon = 10^{-10}$



(d) $\varepsilon = 10^{-10}$

Figure 15. Diagram detailing the test results for $c = 0.5$. Each dot corresponds to a faulty execution, and its color represents the test outcome from Table **??**.

5.2 A practical fault detection scheme

The oracle-based detection scheme described in Section **??** gives good results, however, it cannot be implemented for several reasons. First, the error $\|w_{f+1} - \widetilde{w}_{f+1}\|$ cannot be measured directly, since w_{f+1} is not available, only $\widetilde{w}_{f+1}$ is. An approximation for the error is proposed and evaluated in **??**. Second, the threshold $\tau_c = c \cdot \varepsilon / |\widetilde{y}_{\ell,f}|$ requires the $\widetilde{y}_{\ell,f}$ vector which is computed at the last iteration ℓ of the execution, and is then not yet available at the fault iteration, so an approximation is also proposed and evaluated in **??**. Moreover, the iteration when the fault occurred is not known either, but this issue can easily be taken care of by applying the scheme at each iteration, and replacing f by the current iteration index i as described in **??**. Finally, the new implementable scheme is evaluated in **??**.

5.2.1 Estimation of the error using a check-sum approach

Traditional approaches for detecting transient faults in SpMV product include check-sums [**?**]. The main idea of such techniques is to encode some data, here the Matrix A data, with the vector $\mathbf{1^T} = (\mathbf{1}, \mathbf{1}, ..., \mathbf{1})$, to perform the same operations on the encoded data, which are usually much cheaper than the operations on the real data, and to check the result consistency (result on encoded data is equal to the encoded result on real data). Assuming exact arithmetic and no error happened during the operations on encoded data, a wrong result often implies a faulty execution. In exact arithmetic, the following equations hold :

$$\begin{pmatrix} \mathbf{A} \\ \mathbf{1^T} \cdot \mathbf{A} \end{pmatrix} \cdot v = \begin{pmatrix} \mathbf{A} \cdot \mathbf{v} \\ (\mathbf{1^T} \cdot \mathbf{A}) \cdot \mathbf{v} \end{pmatrix} = \begin{pmatrix} \mathbf{A} \cdot \mathbf{v} \\ \mathbf{1^T} \cdot (\mathbf{A} \cdot \mathbf{v}) \end{pmatrix} \tag{8}$$

In case of a transient fault in the product $\mathbf{A} \cdot \mathbf{v}$, the previous equation becomes:

$$\begin{pmatrix} \widetilde{\mathbf{A}} \\ \mathbf{1^T} \cdot \mathbf{A} \end{pmatrix} \cdot v = \begin{pmatrix} \widetilde{\mathbf{A} \cdot \mathbf{v}} \\ (\mathbf{1^T} \cdot \mathbf{A}) \cdot \mathbf{v} \end{pmatrix} \neq \begin{pmatrix} \widetilde{\mathbf{A} \cdot \mathbf{v}} \\ \mathbf{1^T} \cdot (\widetilde{\mathbf{A} \cdot \mathbf{v}}) \end{pmatrix} \tag{9}$$

The difference between the result on encoded data and the encoded result $|(\mathbf{1^T} \cdot \mathbf{A})^\mathbf{T} \mathbf{v} - \mathbf{1^T}(\widetilde{\mathbf{A} \cdot \mathbf{v}})|$ is called the check-sum, where $\mathbf{1^T} \cdot \mathbf{A}$ is a row vector computed once with its j^{th} entry is the sum of all entries of the j^{th} row of $\mathbf{A}$. Then, in exact arithmetic we have:

$$\text{check-sum} = |(\mathbf{1^T} \cdot \mathbf{A}) \cdot \mathbf{v} - \mathbf{1^T}(\widetilde{\mathbf{A} \cdot \mathbf{v}})| = |\mathbf{1^T}(\mathbf{w} - \widetilde{\mathbf{w}})| = \|\mathbf{w} - \widetilde{\mathbf{w}}\| \tag{10}$$

The last equality holds since $\mathbf{w}$ differs from $\widetilde{\mathbf{w}}$ by only one row (transient fault). If no fault occurred, $\mathbf{w} = \widetilde{\mathbf{w}}$ and the check-sum is equal to 0, whereas if a fault occurred, the check-sum is equal to the error introduced by the fault.

For right-preconditioned-GMRES, in case of a transient fault in the matrix vector product $\mathbf{A} \cdot (\mathbf{M^{-1}} \cdot \mathbf{v})$ the same method holds when $(\mathbf{A} \cdot \mathbf{M^{-1}})$ is encoded instead of A :

$$\text{check-sum} = |(\mathbf{1^T} \cdot \mathbf{A} \cdot \mathbf{M^{-1}}) \cdot \mathbf{v} - \mathbf{1^T}(\widetilde{\mathbf{A} \cdot} (\mathbf{M^{-1}} \cdot \mathbf{v}))| = |\mathbf{1^T}(\mathbf{w} - \widetilde{\mathbf{w}})| = \|\mathbf{w} - \widetilde{\mathbf{w}}\| \tag{11}$$

However for left-preconditioned-GMRES, in case of a transient fault in the matrix vector product $\mathbf{A} \cdot \mathbf{v}$, when $(M^{-1} \cdot A)$ is encoded :

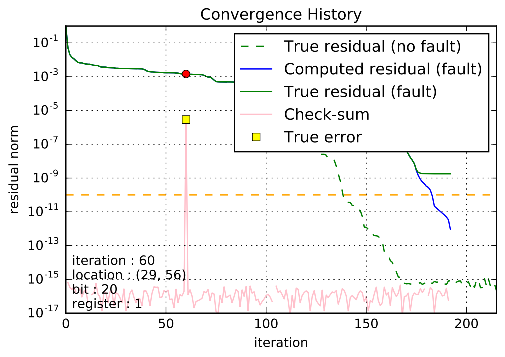
$$\text{check-sum} = |(\mathbf{1^T} \cdot \mathbf{M^{-1}} \cdot \mathbf{A}) \cdot \mathbf{v} - \mathbf{1^T}(\mathbf{M^{-1}} \cdot (\widetilde{\mathbf{A} \cdot \mathbf{v}})| = |\mathbf{1^T}(\mathbf{w} - \widetilde{\mathbf{w}})| \neq \|\mathbf{w} - \widetilde{\mathbf{w}}\| \tag{12}$$

The last equation does not hold anymore since $\widetilde{\mathbf{w}}$ no longer differs from $\mathbf{w}$ by only one row. Nevertheless, the check-sum can still be used as an approximation of the error, even thought it may overestimate the error:

$$\|\mathbf{w} - \widetilde{\mathbf{w}}\| \leq \text{checksum} \leq \sqrt{\mathbf{n}} \|\mathbf{w} - \widetilde{\mathbf{w}}\| \tag{13}$$

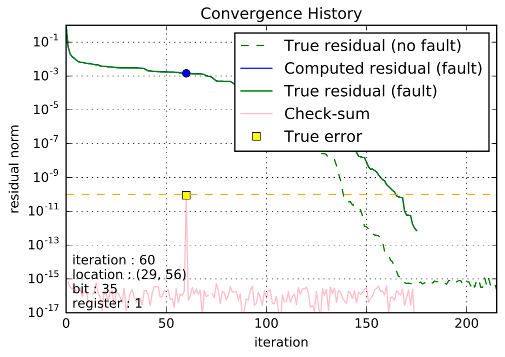
Because all calculations are performed using floating point arithmetic, the previous equations do not hold anymore due to rounding errors (and the loss of associativity). Only an approximation of $\|w - \widetilde{w}\|$ can be obtained from Equation (**??**). Figures **??** and **??** plot the check-sum computed at each iteration i for different injected bit-flips and compares it to the true error $\|w_i - \widetilde{w}_i\|$. In figures **??** and **??**, the error is important and well approximated by the check-sum as the magnitude of the peak in the check-sum (pink line) is roughly the same as the error (yellow square). In **??**, the peak is barely noticeable because of rounding errors in the check-sum computation, but for those extremely small errors, accuracy on the error estimation might be less critical for the fault detection. The same behaviour can be observed in left and right-preconditioned-GMRES, with the additional remark that since the matrice is larger, rounding errors are even more important and smaller fault are harder to approximate and distinguish from rounding errors. Moreover, as illustrated in Equation (**??**) for left-preconditioned-GMRES, the error is slightly overestimated by the check-sum as the peak is slightly taller than the error. To conclude, the check-sum described in Equation (**??**) seems to be a good approximation of the error, but may loose in accuracy as the matrix size increases, making small faults harder to detect.

GMRES executions on **gre_216a**



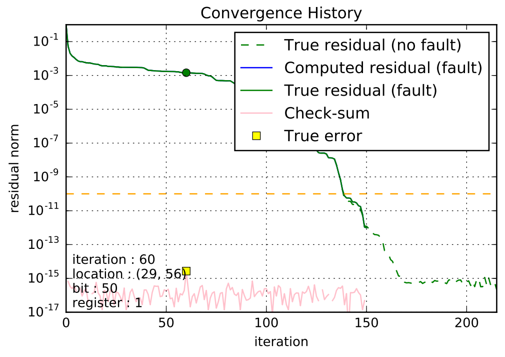
)

(a)



)

(b)

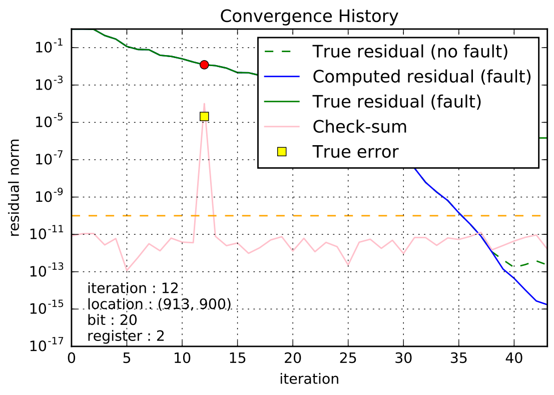


)

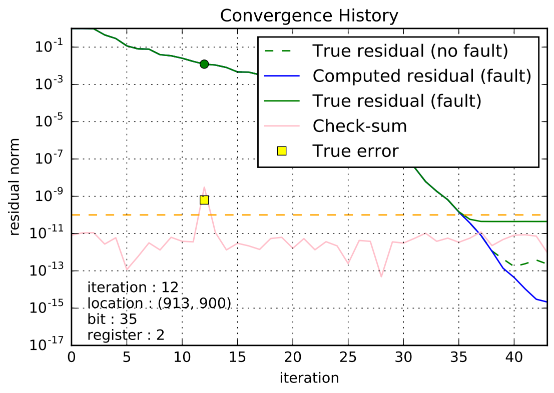
(c)

Figure 16. Convergence histories of 3 GMRES executions on HB/gre_216a, disrupted by a transient fault. The check-sum at each iteration is plotted, as well as the true error denoted by a yellow square. The check-sum seems to be a good approximation of the error as the peak in the check-sum at the fault iteration has roughly the same magnitude as the error.

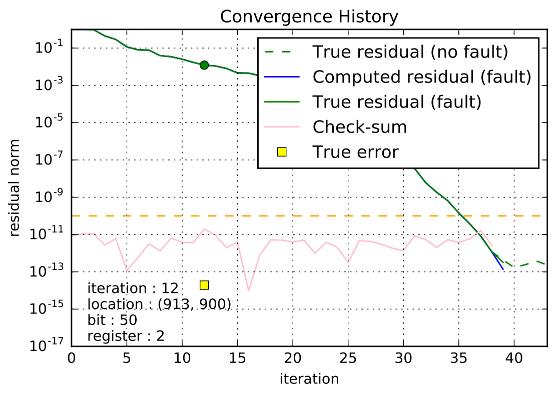
left-precond-GMRES on pores_2



(a)

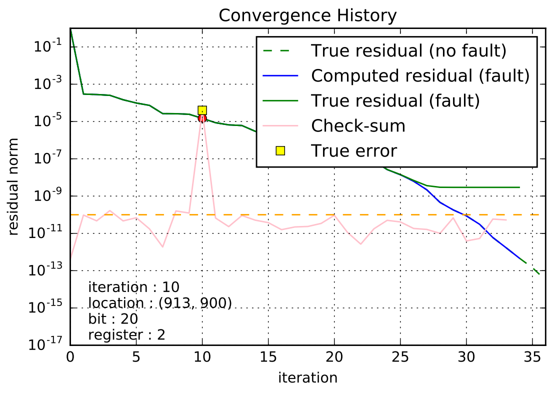


(b)

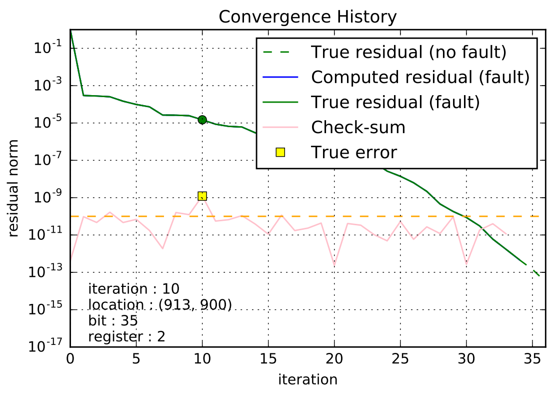


(c)

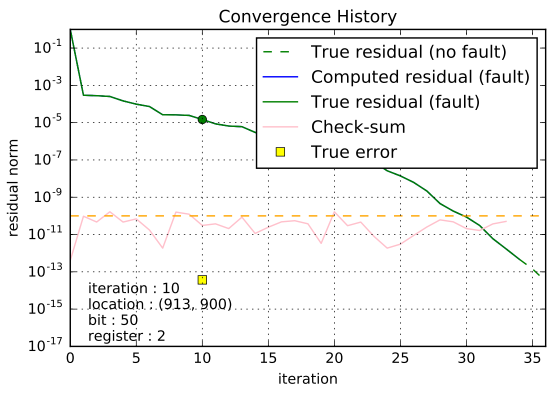
right-precond-GMRES on pores_2



(d)



(e)



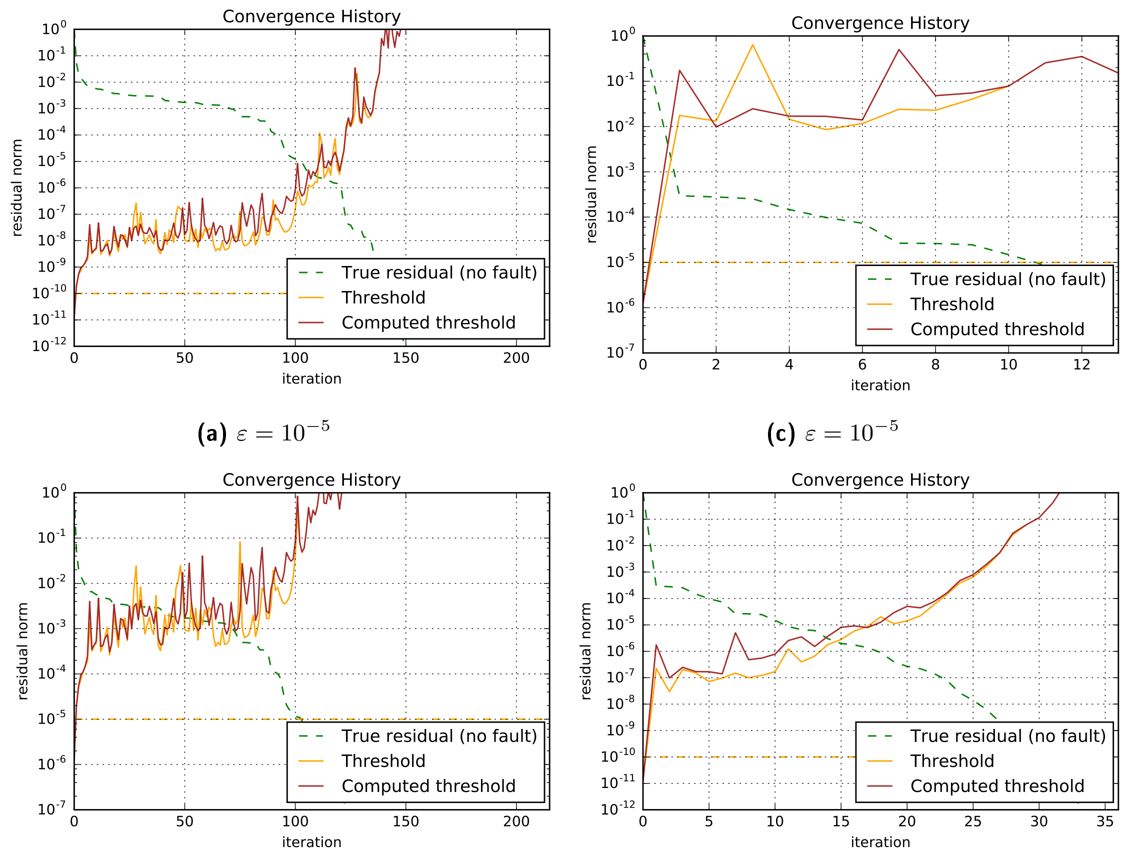
(f)

Figure 17. Convergence histories of 3 left-preconditioned-GMRES executions and 3 right-preconditioned-GMRES executions on HB/pored_2, disrupted by a transient fault. The check-sum at each iteration is plotted, as well as the true error denoted by a yellow square. In left preconditioned-GMRES, the error is slightly overestimated by the check-sum. In both cases, low errors are not accurately estimated because of significant rounding errors in the check-sum computation caused by the large matrix size. Larger errors, however, are quite well approximated.

5.2.2 Approximation of the threshold and evaluation

Based on the observation that $\widetilde{y}_{j,i}$ usually remains nearly constant as j increases, we propose to approximate $\widetilde{y}_{l,i}$ with $\widetilde{y}_{i,i}$ at each iteration i to compute an approximated threshold. This observation does not seem to benefit from much theoretical attention but is supported by the intuition that the solution's directions are mainly determined by the first iteration, whereas the latter iterations are responsible for their refinement. Figure **??** compares the true threshold (orange line) to the approximated one (red line). In each cases, the two curves do not overlap perfectly, nevertheless the general trend of the true threshold seems to be followed by its implementable approximation.

GMRES executions on **gre_216a** **preconditioned-GMRES** on **pores_2**



(a) $\varepsilon = 10^{-5}$ **(c)** $\varepsilon = 10^{-5}$

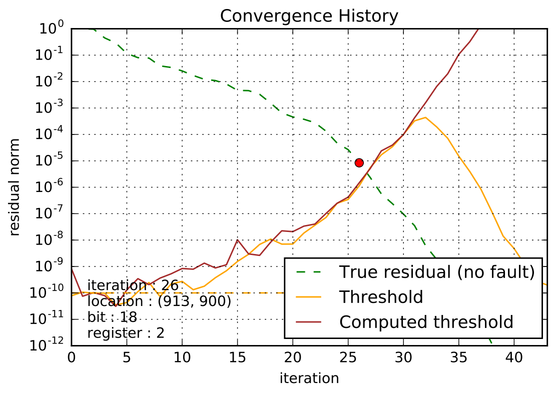
(b) $\varepsilon = 10^{-10}$ **(d)** $\varepsilon = 10^{-10}$

Figure 18. Convergence histories of a GMRES execution on HB/gre_216a and a right-preconditioned-GMRES executions on HB/pored_2 for two target accuracies ($\varepsilon = 10^{-5}$ and $\varepsilon = 10^{-10}$). No fault are injected in the executions. The true threshold ($c \cdot \varepsilon / |\widetilde{y}_{\ell,i}|$) and the approximation ($c \cdot \varepsilon / |\widetilde{y}_{i,i}|$) are compared. Even though the curves do not perfectly overlap, the general evolution of the true threshold seems to be preserved in the approximated one.

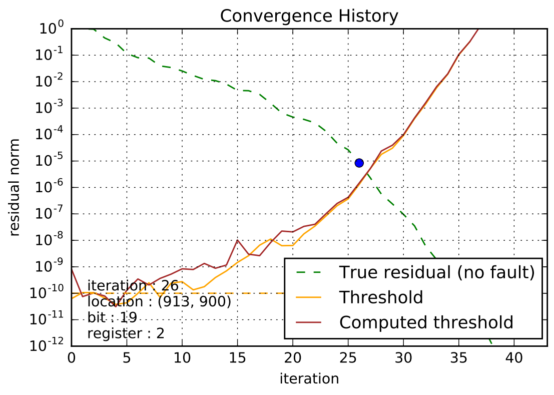
However when faults are injected, the estimation quality may be altered by large faults (bit-flips on the most significant bits). Figure **??** shows two left-preconditioned-gmres (resp.

right-preconditioned-gmres) executions, disrupted by a large fault on bit 18 (resp bit 7), and a slightly smaller fault on bit 19 (resp bit 8). The first observation is that the threshold is altered differently depending on the algorithm. On one hand for left-preconditioned-GMRES, the perturbation appears only at the end of the execution, and modifies significantly the true threshold value (orange line) and not its estimation (red line), when compared to the non-faulty execution (Figure **??**). On the other hand for right-preconditioned-GMRES, the impact on the estimation appears immediately and exclusively at the fault iteration. Moreover in both cases, a slightly smaller fault is enough to completely efface any of the previously described behaviour.

left-precond-GMRES on **pores_2**

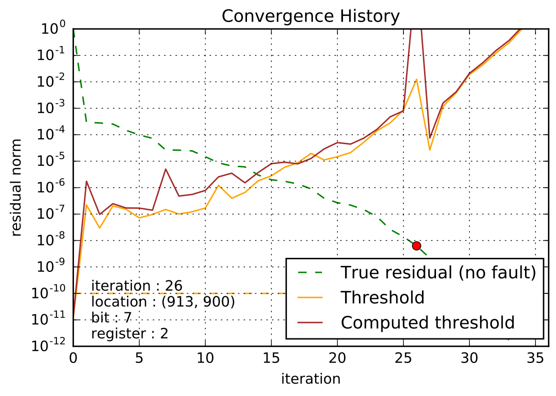


(a) bit 18

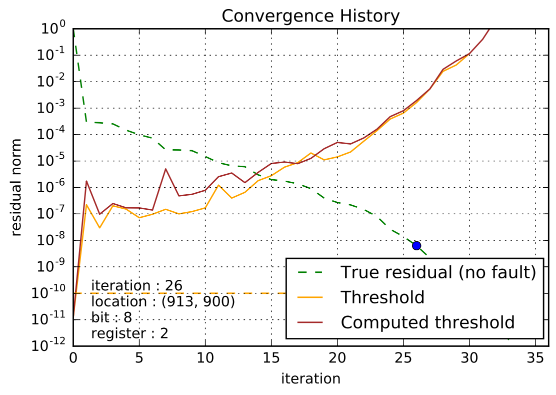


(b) bit 19

right-precond-GMRES on **pores_2**



(c) bit 7



(d) bit 8

Figure 19. Convergence histories of left and right-preconditioned-GMRES executions on HB/pored_2 disrupted by faults at the same iteration. The largest faults have a significant impact on the quality of the threshold approximation, as illustrated in **a)** and **c)**. The faults impact differently each variant of preconditioned-GMRES, as in left-preconditioned-GMRES, the symptoms appears later on, while in right-preconditioned-GMRES, the approximation quality is immediately altered. In **b)** and **d)**, the fault is slightly smaller, but enough to make the perturbation on the threshold approximation completely disappear. For the largest fault (most significant bits and last iterations), it seems that the computed threshold may not be a good approximation of the true threshold, in particular in right-preconditioned-GMRES.

5.2.3 Implementation of the practical scheme

Let $0 < c < 1$. To sum up, here is the practical implementable scheme.

1. The target accuracy is set to $(1 - c) \cdot \varepsilon$.

2. At the beginning of the execution, the matrix A data is encoded with $\mathbf{1^T}$

3. At each iteration i, check-sum(i) and $\tau_c(i)' = c \cdot \varepsilon / \widetilde{y}_{i,i}$ are computed

4. The following test is performed

$$\text{check-sum(i)} < \tau_c(i)' = c \cdot \varepsilon / \widetilde{y}_{i,i}$$

5. If the test is true, then no fault is detected, otherwise the detection is triggered.

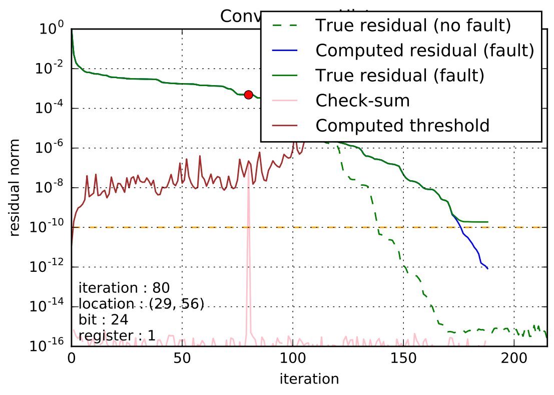
Since the detection is performed at each iteration, the outcome naming needs to be updated (Table **??**). a detection is said to be correct when it happened at the exact iteration when the fault occurred, whereas a detection triggered at any iteration different from the faulty one is said to be incorrect. In Figure **??**, the approximated threshold $\tau_{0.5}$ and the check-sum are

Table 4. New set of outcomes and their colors.

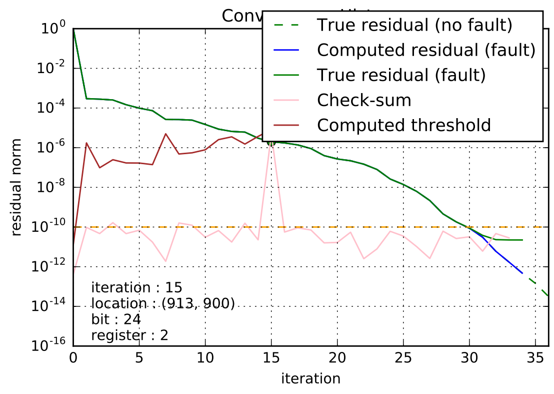
	Convergence	No convergence
Correct detection	**No impact fault detected**	**Critical fault detected**
Incorrect detection	**Incorrect detection**	**Incorrect detection**
No detection	**No impact fault ignored**	**Critical fault ignored**

plotted at each iteration for 2 GMRES executions disrupted by a fault. In Figure **??**, the check-sum at the fault iteration is smaller than the threshold, so the fault is not detected, and the execution does converge, which produce a *No impact fault ignored* from Table **??**. In Figure **??**, the check-sum is greater than the threshold at the fault iteration and the execution does not converge, which produce a *Critical fault detected* from Table **??**. However in Figure **??**, two convergence histories of executions disrupted by an incorrectly detected fault are plotted for GMRES and preconditioned-GMRES. In Figure **??**, check-sum(i) remains below $\tau_{0.5}(i)$ so the fault is undetected, even though the execution does not converge, which correspond to a *Critical fault not detected* from Table **??**. In Figure **??**, check-sum(i) exceeds $\tau_{0.5}(i)$ at the fault iteration so a fault is detected, but the execution still converges which correspond to a *No impact fault detected* from Table **??**.

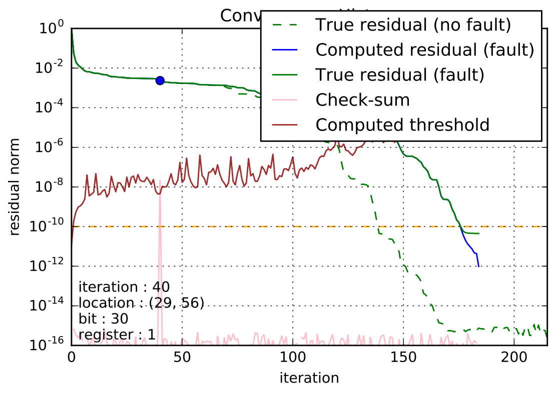
-GMRES executions on **gre_216a** **preconditioned-GMRES** on **pores_2**



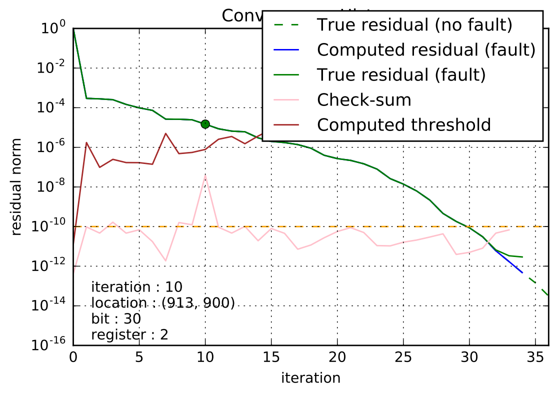
(a) No impact fault ignored



(c) No impact fault ignored



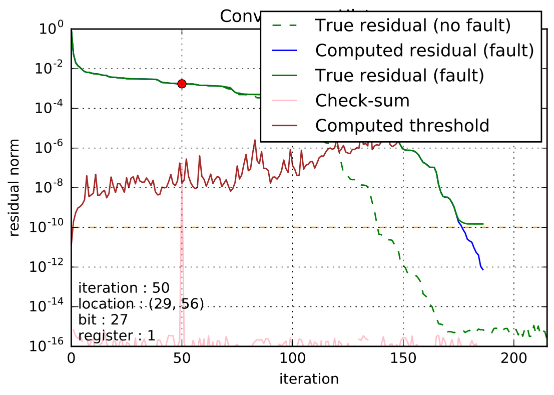
(b) Critical fault detected



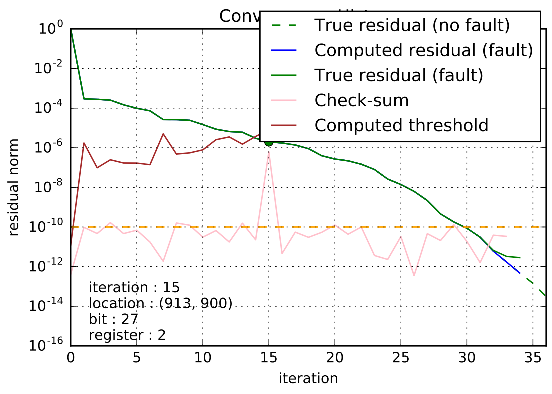
(d) Critical fault detected

Figure 20. Convergence histories of 2 GMRES executions on HB/gre_216a and 2 preconditioned-GMRES executions on HB/pored_2, disrupted by transient faults that are properly detected. The check-sum and the threshold at each iteration are plotted. In figures **??** and **??**, the check-sum at each iteration remains below the threshold, so no fault is detected and the executions converge, producing a *No impact fault ignored* from Table **??** in both cases. In figures **??** and **??**, the check-sum becomes higher than the threshold at the fault iteration, so the faults are detected and the executions do not converge, producing a *Critical fault detected* from Table **??** in both cases.

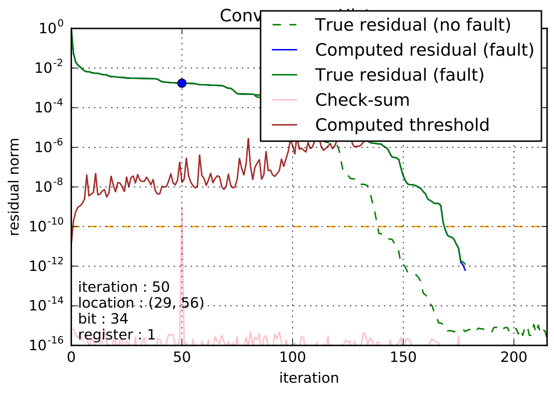
GMRES executions on **gre_216a** **preconditioned-GMRES** on **pores_2**



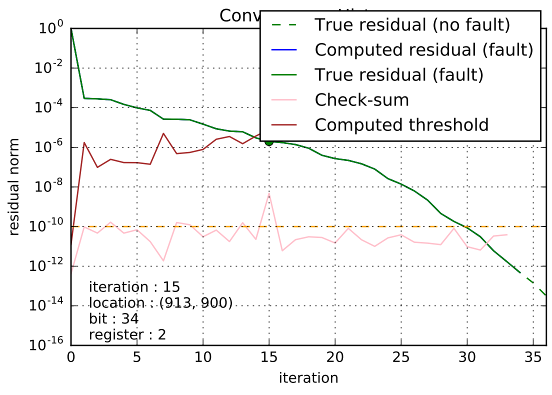
(a) Critical fault ignored



(c) Critical fault ignored



(b) No impact fault detected



(d) No impact fault detected

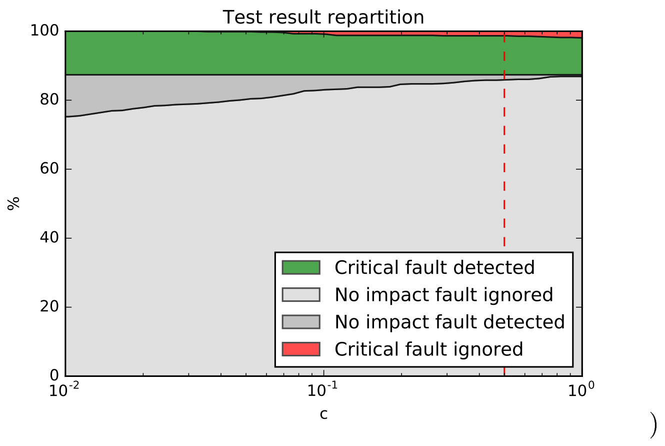
Figure 21. Convergence histories of 2 GMRES executions on HB/gre_216a and 2 preconditioned-GMRES executions on HB/pored_2, disrupted by transient faults that are not properly detected. In figures **??** and **??**, the check-sum at each iteration remains below the threshold, so no fault is detected and the executions do not converge, producing a *Critical fault ignored* from Table **??** in both cases. In figures **??** and **??**, the check-sum becomes slightly higher than the threshold at the fault iteration, so the faults are detected but the executions converge anyway, producing a *No impact fault detected* from Table **??** in both cases.

This scheme requires some additional operations to be performed, mainly caused by the check-sum computation. First, since the target accuracy is set to $(1-c)\cdot\varepsilon < \varepsilon$, some additional iterations $\widetilde{\ell} - \ell$ may be required for the execution to converge. Second, the quantity $\mathbf{1^T}\cdot\mathbf{A}$ is computed once at the beginning, and requires $\Theta(nnz)$ (number of nonzero elements in A) operations. Then, at each iteration, the vectors $(\mathbf{1^T}\cdot\mathbf{A})\cdot\mathbf{v}$ and $\mathbf{1^T}\cdot(\mathbf{A}\cdot\mathbf{v})$ as well as their Euclidean distance are computed, necessitating at each iteration $\Theta(n)$ operations. Finally, $\Theta(1)$ operations are needed for the check-sum $<$ threshold comparison. Therefore, the test requires $\Theta(nnz) + \Theta(n\cdot\widetilde{\ell})$ additional operations, negligible when compared to the $\Theta(\ell\cdot nnz)$ operations of the classical algorithm, as long as $n \ll nnz$ and $\widetilde{\ell} \approx \ell$. Moreover, those computations can be performed in parallel in a separate computation unit since their outputs are not used in the classical execution.

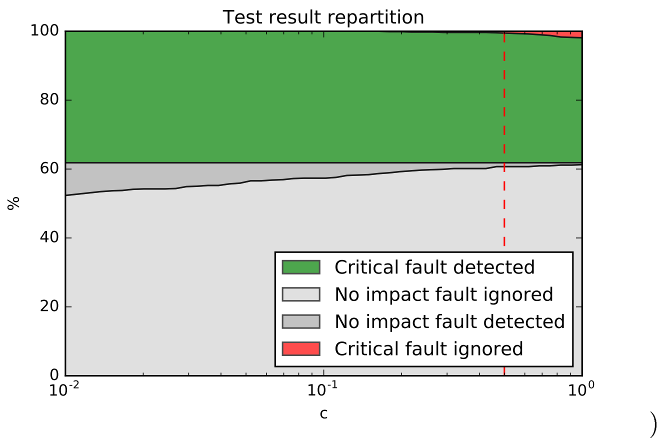
5.2.4 Evaluation of the practical detection scheme

Figure **??** plots the proportion (% of executions) of detection scheme outcomes for several values of c. Because of the previously described approximations, Theorem **??** does not hold anymore and some *Critical fault ignored* are produced. Except for the preconditioned-GMRES case in low accuracy ($\varepsilon = 10^{-5}$), the detection scheme does perform well as the proportion of critical fault missed is relatively low. For more details, the $c = 0.5$ cases are plotted in Figure **??** and in Figure **??** for the results on the matrix set. However, the case preconditioned-GMRES in low accuracy is worrying. To understand why the test performed poorly in this situation, Figure **??** plots the approximated threshold and check-sum in the case of an important fault inducing a large error. We can see that in both plots, the fault increases significantly the threshold, disrupting the convergence (which never seems to happen in GMRES executions without preconditioner). This problem comes from the approximation on $\widetilde{y}_{l,f} \to \widetilde{y}_{f,f}$, so another approximation should probably be established to ensure better results in various inputs.

GMRES executions on **gre_216a**



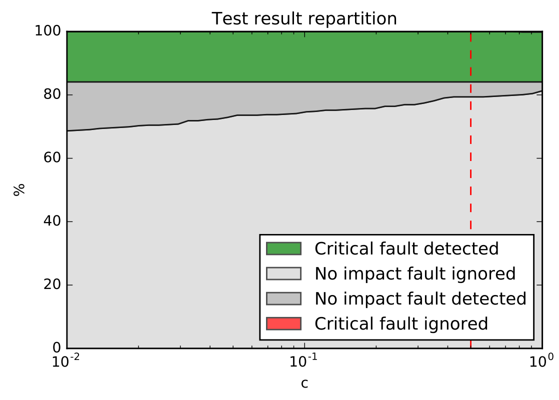
(a) $\varepsilon = 10^{-5}$



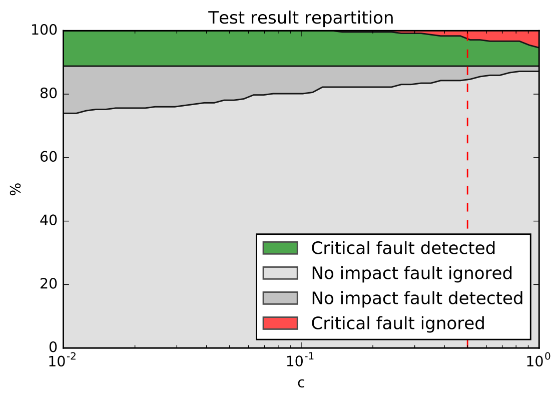
(b) $\varepsilon = 10^{-10}$

Figure 22. Diagrams representing the test outcome proportion for several values of c. To compute them, faulty executions covering a large part of the fault parameter space were performed. The case $c = 0.5$ represented by the dashed red line is detailed in Figure **??**. The dashed orange line corresponds to the proportion of execution where an incorrect detection happens.

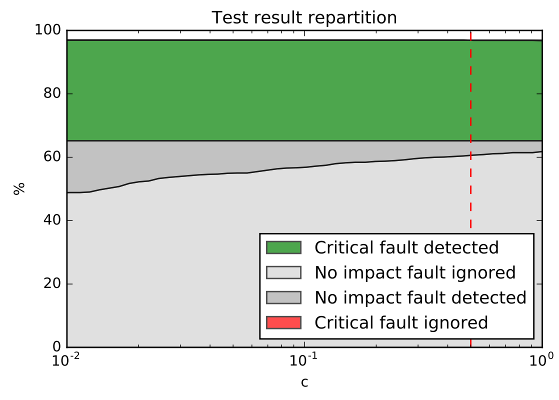
left-precond-GMRES on **pores_2** **right-precond-GMRES** on **pores_2**



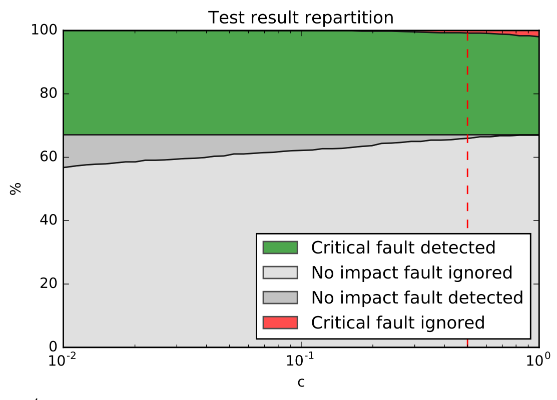
(a) $\varepsilon = 10^{-5}$



(c) $\varepsilon = 10^{-5}$



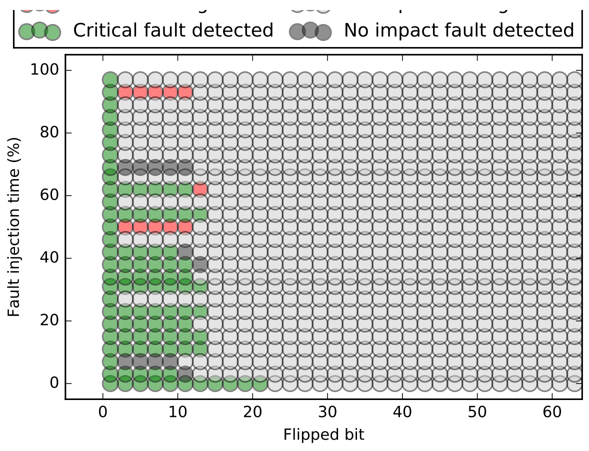
(b) $\varepsilon = 10^{-10}$



(d) $\varepsilon = 10^{-10}$

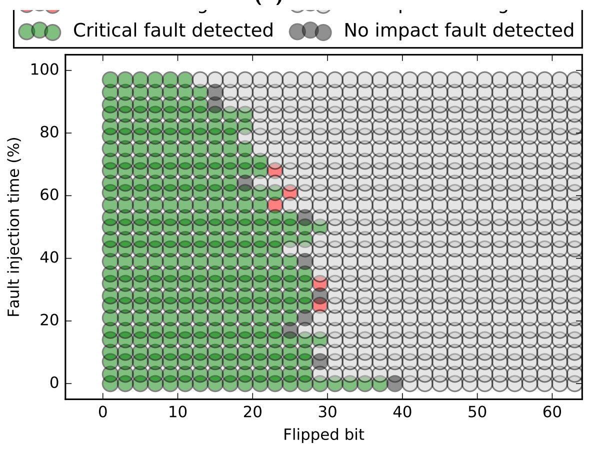
Figure 23. Diagrams representing the test outcome proportion for several values of c. To compute them, faulty executions covering a large part of the fault parameter space were performed. The case $c = 0.5$ represented by the dashed red line is detailed in Figure **??**. The dashed orange line corresponds to the proportion of execution where an incorrect detection happens.

GMRES executions on **gre_216a**



)

(a) $\varepsilon = 10^{-5}$

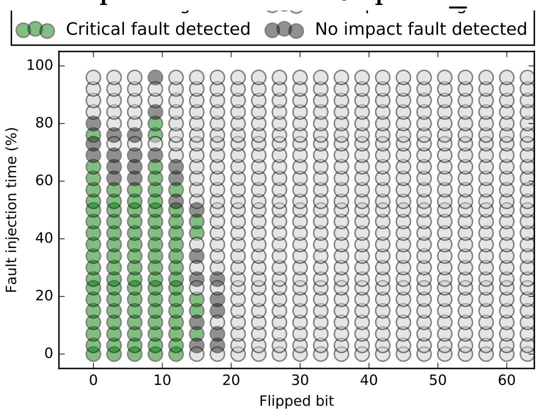


)

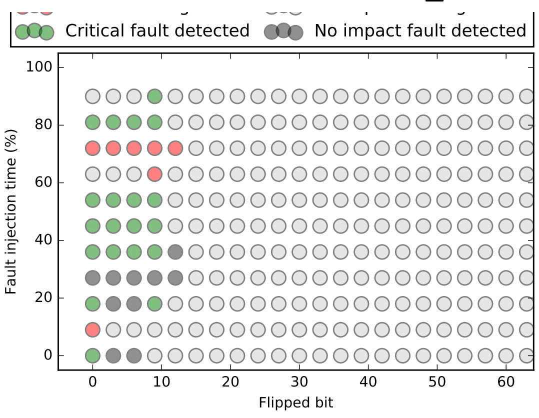
(b) $\varepsilon = 10^{-10}$

Figure 24. Diagram detailing the test results for $c = 0.5$. Each dot corresponds to a faulty execution, and its color represents the test outcome.

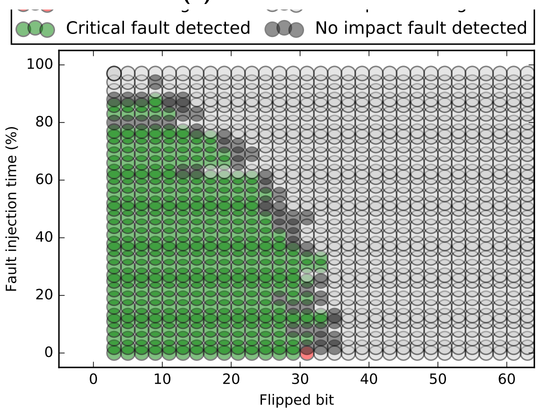
left-precond-GMRES on **pores_2** **right-precond-GMRES** on **pores_2**



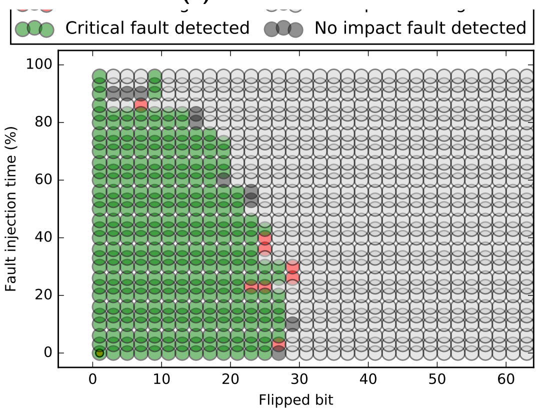
(a) $\varepsilon = 10^{-5}$



(c) $\varepsilon = 10^{-5}$

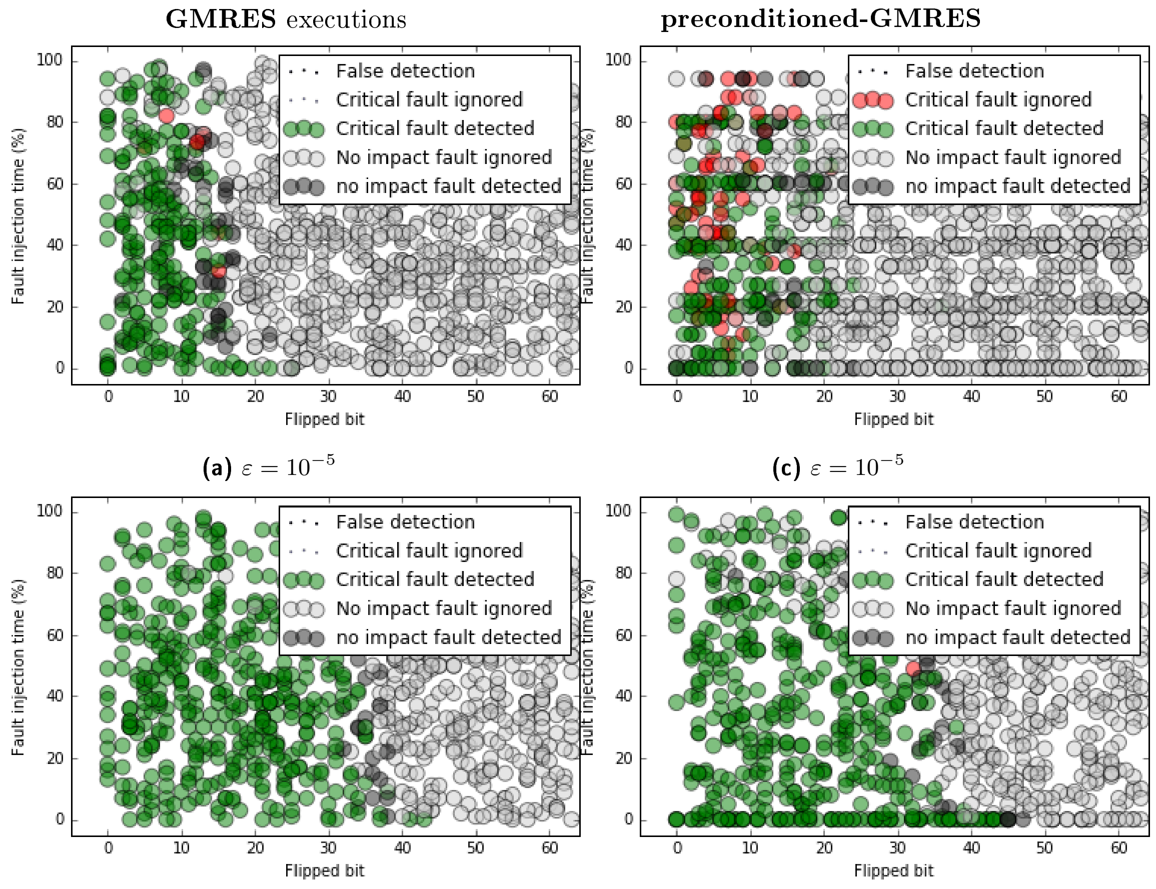


(b) $\varepsilon = 10^{-10}$



(d) $\varepsilon = 10^{-10}$

Figure 25. Diagram detailing the test results for $c = 0.5$. Each dot corresponds to a faulty execution, and its color represents the test outcome.



(a) $\varepsilon = 10^{-5}$ **(c)** $\varepsilon = 10^{-5}$

(b) $\varepsilon = 10^{-10}$ **(d)** $\varepsilon = 10^{-10}$

Figure 26. Diagram detailing the test results for $c = 0.5$ and several matrices. Each dot corresponds to a faulty execution, and its color represents the test outcome.

6 Conclusion

In this study, we observed how transient faults can impact GMRES executions. Since some faults may induce errors large enough to disrupt the convergence, we proposed an experimental and an analytical study to get a better understanding of this phenomenon, by adapting the theory of inexact and relaxed GMRES. With this knowledge, a theoretical criterion able to accurately detect such faults was derived. However, to be of practical use, the detection scheme had to be adapted, using some approximation that enabled its implementation. It was then evaluated and performed well for the inputs used. However it may present some limitations. First, the quality of the error approximation by check-sum might deteriorate as the system size increases, so further experiments involving larger matrices should be performed. Then, the threshold approximation depends on a empirical statement that do not benefit from much theoretical attention, hence it might not be accurate for all inputs. Also, its implementation depends on some assumptions, such as the necessity for the matrix A to be able to be encoded while it often comes only in the form of an operator. Finally, future work could include the detection in both the SpMV operation and the preconditioner application for the preconditioned-GMRES case, an adaptation for variants of the GMRES algorithm such as pipelined-GMREs [**?**], and a parallel implementation.

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399