On soft errors in the Conjugate Gradient method: sensitivity and robust numerical detection

Emmanuel Agullo, Siegfried Cools, Emrullah Fatih-Yetkin, Luc Giraud, Wim Vanroose

On soft errors in the Conjugate Gradient method: sensitivity and robust numerical detection

Emmanuel Agullo, Siegfried Cools, Emrullah Fatih-Yetkin, Luc Giraud, Wim Vanroose

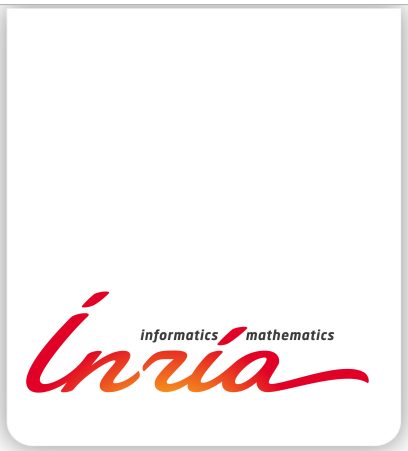
RESEARCH REPORT

N° 9226

November 2018

Project-Team HiePACS

ISSN 0249-6399 ISRN INRIA/RR--9226--FR+ENG



On soft errors in the Conjugate Gradient method: sensitivity and robust numerical detection

Emmanuel Agullo*, Siegfried Cools†, Emrullah Fatih-Yetkin‡,
Luc Giraud*, Wim Vanroose†

Project-Team HiePACS

Research Report n° 9226 — November 2018 — 35 pages

Abstract: The conjugate gradient (CG) method is the most widely used iterative scheme for the solution of large sparse systems of linear equations when the matrix is symmetric positive definite. Although more than sixty year old, it is still a serious candidate for extreme-scale computation on large computing platforms. On the technological side, the continuous shrinking of transistor geometry and the increasing complexity of these devices affect dramatically their sensitivity to natural radiation, and thus diminish their reliability. One of the most common effects produced by natural radiation is the single event upset which consists in a bit-flip in a memory cell producing unexpected results at application level. Consequently, the future computing facilities at extreme scale might be more prone to errors of any kind including bit-flip during calculation. These numerical and technological observations are the main motivations for this work, where we first investigate through extensive numerical experiments the sensitivity of CG to bit-flips in its main computationally intensive kernels, namely the matrix-vector product and the preconditioner application. We further propose numerical criteria to detect the occurrence of such faults; we assess their robustness through extensive numerical experiments.

Key-words: Soft-error, bit-flip, Conjugate Gradient method, numerical detection, sensitivity, robustness, exascale

* Inria, France
† Applied Mathematics Group, University of Antwerpen, Belgium
‡ Kadir Has University, Turkey

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vielle Tour
33405 Talence Cedex

Sur les soft-erreurs dans la méthode du Gradient Conjugué: sensibilité et détection numérique robuste

Résumé : La méthode du gradient conjugue (CG) est la méthode itérative la plus utilisées pour résoudre des ssytèmes linéaires creux de grande taille lorsque la matrice est symétrique définie positive. Bien que vieille de de soixante ans, cette méthode reste une candidate sérieuse pour être mise en œuvre pour la résolution de très grands systèmes linéaires sur des plateformes de calcul de très grande taille. Sur le plan technologique, la réduction permanente de la taille et la complexité croissante des composantes électroniques de ces calculateurs affecte dramatiquement leur sensibilité aux radiations cosmiques ce qui réduit leur fiabilité. L'un des effets les plus courants des rayonnements naturels est la perturbation due à un événement unique qui consiste en un retournement de bit dans une cellule mémoire produisant des résultats inattendus au niveau de l'application. Par conséquent, les futures installations informatiques à très grande échelle pourraient être plus sujettes à des erreurs de toute sorte. y compris le basculement de bit pendant le calcul. Ces observations numériques et technologiques sont les suivantes les principales motivations de ce travail, pour lequel nous étudions d'abord par le biais d'études approfondies et approfondies la sensibilité de la CG aux sauts de bits dans ses principaux domaines d'application. à forte intensité de calcul, à savoir le produit matrice-vecteur et le produit application du préconditionneur. Nous proposons en outre des critères numériques pour détecter l'apparition de tels défauts ; nous évaluons leur robustesse à travers des expériences numériques approfondies.

Mots-clés : Soft-erreur, bit-flip, Gradient Conjugué, détection numérique, sensibilité, robustesse, exascale

Contents

1 Introduction

The flexibility offered by Von Neumann machines for programming complex algorithms motivated the quest for the design of many innovative numerical algorithms in the late 40's. Computers being considered as unreliable machines, it was not rare that the presentation of those algorithms was accompanied with advanced numerical considerations for distinguishing "normal rounding-off errors" due to digital computation from "errors of the computers", using the expressions from [16], which introduced the Conjugate Gradient (CG) algorithm, a seminal paper of that unprecedented fertile period. This numerical technique is still a method of choice for solving large sparse linear systems of equations involving a symmetric positive definite (SPD) matrix. While the study of numerical correctness since led to a continuous, productive research field [17], the tremendous progress in hardware design progressively vanished the interest of detecting *soft* errors (using the modern expression for characterizing faults that do not lead to an immediate failure of a program). At some extra energy cost, hardware mechanisms such as Error Correcting Codes (ECC) were indeed able to correct most soft errors so that, from a numerical design point of view, they could almost be considered as – and indeed often were – a non-existent problem.

The advent of distributed-memory platforms and network of heterogeneous workstations in the 90's once again drew the attention of the computational science community to faults, as long running parallel executions were regularly aborted because of the crash or the non response of only a single processing unit. For this reason, early high-level parallel libraries for programming these parallel platforms such as the Parallel Virtual Machine (PVM) [10] proposed primitives (`hostfailentry()` in the revision 3 of the PVM standard for instance) allowing an application to design tailored schemes for recovering from such *hard* faults (using modern terminology). From the numerical perspective, one major breakthrough was the introduction of algorithm-based fault tolerance (ABFT) schemes [19] to detect and correct errors when matrix operations such as addition, multiplication, scalar product, LU-decomposition, and transposition are performed using multiple processor systems. The proposed method could "detect and correct any failure within a single processor in a multiple processor system", using the words of the authors. On the other hand, the operating system community developed efficient checkpoint-restart (CPR) mechanisms. Together with the progress of interconnect network technologies, their ability to handle those faults in a transparent way for the application once again allowed programmers to view (parallel) computers as reliable computing platforms. As a consequence, in spite of solid proposals (such as [2]), the most widely employed interface for programming distributed-memory machines today, the Message Passing Interface (MPI) standard, does not provide support allowing applications to design customized resilient schemes, even in its revision 3.1 [8], the most recent standard at the time of writing the present article.

As the size of transistors continues to reduce and the number of components continues to increase, soft errors in supercomputers become more and more common. In the fault tolerance literature, many techniques have been proposed to detect and/or correct soft errors. The best-known general technique to detect soft errors is the double modular redundancy (DMR) approach. This approach either uses two different pieces of hardware to perform the same computation at the same time or performs the same computation on the same hardware twice, then compares the two results to detect whether errors occur or not. The most well-known general technique to correct single soft errors is the triple modular redundancy (TMR) approach. TMR either performs the same computation on three different pieces of hardware or uses the same hardware to perform the same computation three times, then compares and selects the results obtained consistently at least twice as the correct result. While DMR and TMR are very general, their overhead is high - up to 100% overhead to detect errors and 200% overhead to correct errors. To protect memory corruption, ECC memory has been widely used by many computer vendors. Although

today's ECC memory can detect and correct bit flips in memory, it brings significant overhead in space, time, and energy. Furthermore, ECC memory is not able to handle computational (i.e., arithmetic) errors that are caused by faults in logic units. New types of unreliable hardware, such as *in-memory* computers [9] (which are no longer Von Neumann machines), are emerging as serious competitors (or, more likely, accelerators) to traditional processors, as they are expected to achieve a much higher energy efficiency.

As the preconditioned version of CG, PCG, remains the subspace Krylov method of choice for solving large sparse SPD linear systems, the goal of this paper is to study its sensitivity to soft errors and propose numerical remedies to detect them. Although much truncated, the above very brief journey through half a dozen decades of the modern computing era shows that our concern for the reliability of machines has been slowly but constantly evolving. This motivates us to characterize these errors from a high-level point of view, disregarding low-level hardware details, may the underneath silicon be an exascale machine, the initial motivation for this work, or, potentially, an embedded computer in high altitude [30] or any type of unreliable digital computer (such as [9]). We therefore only assume that soft errors may occur, corrupting data or computation, without the hardware or system detecting them and notifying the application that a fault occurred. We call this type of soft error, "silent data corruption" (SDC) (after [6]), which may cause a simulation to silently return an incorrect answer that does not reflect any sensibility of the solution to the input data and consequently could lead to a misleading analysis of the computation outcome. We focus on such faults in the present study and we refer the reader to [15] and the references therein for a recent overview on the resilience and fault tolerance issues in HPC.

The first contribution of this paper is to study the sensitivity of PCG to such soft errors. We mainly focus on its main computational kernels, *i.e.*, the matrix-vector product and the preconditioner application. In particular, we investigate the sensitivity to soft errors for various space and time locations.

The second contribution is the proposition of two numerical criteria to detect those soft errors. The PCG method is a very sophisticated and elegant numerical scheme that has many properties induced by the symmetric positive definiteness of the matrix A. Unfortunately, most characteristic properties are no longer valid in finite precision calculation. On the other hand, a lot of work has been devoted to study PCG in finite precision [21, 20]. We therefore consider some of the finite precision results to define a first possible numerical soft error detection mechanism based on the residual gap. We consider an additional criterion, which was already proposed in the original paper on CG [16, Thm 5.5], based on a bound (as opposed to a strict equality), hence expected to be less prone to defection due to finite precision calculation. We study the respective quality of both these criteria and show that, combined together, they are extremely efficient to detect faults, occurring not only in the matrix-vector product and preconditioner application, but also in all the operations involved in PCG.

The rest of the paper is organized as follows. In Section 2, we briefly present the PCG algorithm, we review the main classical error detection techniques and we detail round-off and soft error modeling. In Section 3, we then study the sensitivity of PCG to soft errors occurring in its main computational kernels that are the matrix-vector product and the preconditioner application. In particular, we investigate the dependency on the bit and temporal location of the error. In Section 4, we propose numerical criteria to detect soft errors in those two kernels and assess their robustness through extensive numerical experiments. Finally, we summarize the contributions of this paper in Section 5 and draw up some perspectives for future works.

2 Background

2.1 Preconditioned Conjugate Gradient algorithm

PCG is a Krylov subspace method for solving a large linear system

$$Ax = b$$

where $A \in \mathbb{R}^{n\times n}$ is symmetric positive definite (SPD), the right-hand side $b \in \mathbb{R}^n$ and the solution $x \in \mathbb{R}^n$. In exact arithmetic, the method converges within at most n iterations. In practice with finite precision calculation, this property does not hold and preconditioning [23, 28] is often needed to accelerate the convergence. While in the context of Krylov subspace methods, PCG is a sophisticated, elegant and powerful numerical algorithm [21, 23, 28], its algorithm looks fairly simple and can be written as depicted in Algorithm 1, where M defines the preconditioner.

Algorithm 1 Preconditioned Conjugate Gradient

1: $r_0 := b - Ax_0$; $u_0 = M^{-1}r_0$; $p_0 := r_0$
2: **for** $i = 0, \ldots$ **do**
3: $\quad s_i := Ap_i$
4: $\quad \alpha_i := r_i^T u_i / s_i^T p_i$
5: $\quad x_{i+1} := x_i + \alpha_i p_i$
6: $\quad r_{i+1} := r_i - \alpha_i s_i$
7: $\quad u_{i+1} = M^{-1} r_{i+1}$
8: $\quad \beta_{i+1} := r_{i+1}^T u_{i+1} / r_i^T u_i$
9: $\quad p_{i+1} := u_{i+1} + \beta_{i+1} p_i$
10: **end for**

2.2 Double Modular Redundancy

Although potentially costly, DMR is certainly the best-known general agnostic technique for detecting soft errors. It consists in duplicating both the operations and data, and checking the equality of the output of the redundant calculations. For classical PCG, soft error detection with such a full duplication technique can be implemented as described in Algorithm 2.

Algorithm 2 Preconditioned Conjugate Gradient with DMR detection

1: $r_0^1 := b - Ax_0$; $p_0^1 := r_0^1$; $r_0^2 := b - Ax_0$; $p_0^2 := r_0^2$
2: **for** $i = 0, \ldots$ **do**
3: $\quad s^1 := Ap_i^1$; $s^2 := Ap_i^2$; check$(s^1 == s^2)$
4: $\quad \alpha^1 := (r_i^1, r_i^1)/(s^1, p_i^1)$; $\alpha^2 := (r_i^2, r_i^2)/(s^2, p_i^2)$; check$(\alpha^1 == \alpha^2)$
5: $\quad x_{i+1}^1 := x_i^1 + \alpha^1 p_i^1$; $x_{i+1}^2 := x_i^2 + \alpha^2 p_i^2$; check$(x_{i+1}^1 == x_{i+1}^2)$
6: $\quad r_{i+1}^1 := r_i^1 - \alpha^1 s^1$; $r_{i+1}^2 := r_i^2 - \alpha^2 s^2$; check$(r_{i+1}^1 == r_{i+1}^2)$
7: $\quad u_{i+1}^1 := {M^{-1}}^1 r_{i+1}^1$; $u_{i+1}^2 := {M^{-1}}^2 r_{i+1}^2$; check$(u_{i+1}^1 == u_{i+1}^2)$
8: $\quad \beta^1 := (r_{i+1}^1, u_{i+1}^1)/(r_i^1, u_i^1)$; $\beta^2 := (r_{i+1}^2, u_{i+1}^2)/(r_i^2, u_i^2)$; check$(\beta^1 == \beta^2)$
9: $\quad p_{i+1}^1 := u_{i+1}^1 + \beta^1 p_i^1$; $p_{i+1}^2 := u_{i+1}^2 + \beta^2 p_i^2$; check $(p_{i+1}^1 == p_{i+1}^2)$
10: **end for**

This duplication process is simple, generic and effective. However, the price for this detection technique may not be affordable with respect to computational and storage costs. Level-1 BLAS

operations have a moderate computational cost so that their duplication may be considered acceptable under certain conditions. On the other hand, duplication of the two most computationally intensive kernels that are the matrix-vector product and preconditioning may not be affordable. Alternatively, checksum techniques can be employed to protect and check the correctness of these last two kernels.

2.3 Checksum techniques

Detecting soft errors in matrix calculation with checksum techniques was first proposed in [19] and further investigated for applications with iterative methods in [25]. To protect a matrix-vector product the main idea relies on the following equality that holds in exact arithmetic for any vector $v \in \mathbb{R}^n$:

$$\mathbb{I}^T(Av) = (\mathbb{I}^T A)v$$

where $\mathbb{I}$ is the vector of $\mathbb{R}^n$ with all entries equal to one. Each entry ℓ of the row vector $c^T = (\mathbb{I}^T A)$ is the checksum of the ℓ^{th} column of A that can be securely computed before starting PCG. At each PCG iteration, checking the correctness of the matrix-vector product reduces to test the equality:

$$c^T p_{i-1} = \mathbb{I}^T s_{i-1}.$$

The left-hand side requires an extra dot-product calculation and the right-hand side is simply the sum of all the entries of the output vector s_{i-1}. Because of the symmetry of A the checksum vector can also be computed by $c = A\mathbb{I}$ so that the idea can be applied in a matrix-free context as well as for the preconditioner application calculation.

In finite precision arithmetic, there may not be a bitwise equality between $c^T p_{i-1}$ and $\mathbb{I}^T s_{i-1}$ because of the non-associativity of the sum in presence of round-off errors [17]. To assess the correctness of the matrix-vector product, we cannot simply check the bitwise equality and instead need to consider that the calculation is correct if

$$\frac{|c^T p_{i-1} - \mathbb{I}^T s_{i-1}|}{|c^T p_{i-1}|} \leq \tau, \tag{1}$$

where the threshold τ has to be safely defined. To prevent the occurrence of too many false detection instances (referred to as false-positives), the threshold τ needs to be chosen large enough. However, τ cannot be chosen arbitrarily large; otherwise actual faults would be missed. Consequently, the checksum-based detection methods need some tuning of the threshold τ with respect to round-off effects to achieve effectiveness, as further discussed in Section 4.2.1.

2.4 Round-off and soft error models

2.4.1 Round-off errors

Round-off errors are conservatively modeled with the IEEE 754 specification. In particular, we rely on the IEEE 754 double-precision binary floating-point format, referred to as binary64 and illustrated in Figure 1. Let us denote $(b_i)_{i=0}^{63}$ the sequence of bits defining a double-precision number d where $(b_i)_{i=0}^{51}$ defines the mantissa/fraction, $(b_i)_{i=52}^{62}$ the exponent and b_{63} the sign so that

$$d = (-1)^{b_{63}} 2^{e-1023} (1+m), \text{with } e = \sum_{i=52}^{62} b_i 2^{i-52} \text{ and } m = \sum_{i=0}^{51} b_i 2^{i-52}. \tag{2}$$

The accuracy and stability of numerical algorithms in the presence of round-off errors is still an intense field of research [17] and we will rely on such finite precision results to define a first possible soft error detection mechanism in Section 4.1.1.

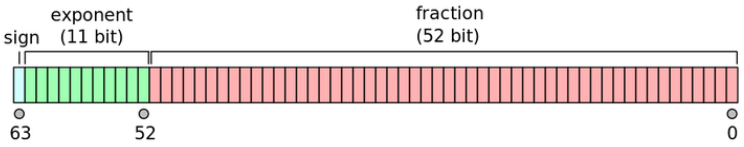


Figure 1: IEEE 754 double-precision binary floating-point format.

2.4.2 Soft errors

In addition to round-off errors (the "normal rounding-off errors" in [16]), soft errors (the "errors of the computers" in [16]) may occur. As discussed in the introduction of the present paper, a brief journey through the modern computing era motivates us to characterize soft errors from a high-level point of view, disregarding low-level hardware details. While there is a substantial literature for qualifying errors based on their hardware characteristics, much less effort has been devoted to qualify them from a high-level point of view. However, quoted in [22], Rees argued back in 1997 that "failure is a matter of function only [and is thus] related to purpose, not to whether an item is physically intact or not". M. Hoemmen and M. Heroux proposed a fault characterization in that perspective [18]. In [6], J. Elliot, M. Hoemmen and F. Mueller also consider a type of fault they do characterize from a high-level point of view, i.e., in their case, "*a fault that silently introduces bad data, while not persistently tainting the data that was used in the calculation. For example, let* $a = 2$ *and* $b = 2$*, then* $c = a + b = 10$." They acknowledge that "*while simplistic, this model presumes no knowledge of the nature of the fault, only that* c *is incorrect. This model assumes that the machine is unreliable in an unpredictable way, and therefore [they] are skeptical of the output it presents.*"

We consider that model referred to as silent data corruption (SDC) in [6] and refine it, depending on (1) whether the input data also gets tainted once the operation has completed, (2) at which stage the perturbation occurs and (3) how the perturbation is incurred. First, we distinguish whether the input data (a and b in the above example) is tainted or not once the operation has completed. Assume, for instance, that a gets tainted (say, $a = 8$ instead of 2) during the execution of the operation, eventually leading to the perturbation of c ($c = 10$). If the perturbation on a occurred while a was on a *persistent* memory (such as the main memory), a remains tainted once the operation has completed. We refer to this case as a *persistent* soft error. In the above example, the final state would be: $c = 10$, $a = 8$, $b = 2$. This type of fault can be detected if the input data have been stored redundantly without the need of performing the computation redundantly. On the contrary, if the perturbation on a occurred while a was on *transient* memory (such as a cache), a does not remain tainted once the operation has completed. Following [18, 5] terminology, we refer to this case as a *transient* soft error. In the example, the final state would be: $c = 10$, $a = 2$, $b = 2$. This second type of fault cannot be detected relying only on input data redundancy and is thus more critical. We therefore focus only on such transient soft errors in the remainder of the paper (and we present results for an analogous study on persistent soft errors in Appendix C).

The second refinement we make explicit with respect to the model proposed in [6] is the stage at which the perturbation occurs. As we do not aim at considering low-level details, we consider the logical operation at which a fault occurs as opposed to an hardware instruction or a floating point operation. In our case, we thus refer to an instruction in Algorithm 1 such as the matrix-vector product (step 3) or the application of the preconditioner (step 7). We will focus mainly on both these operations as they are the most time consuming and DMR may thus be expensive to apply. Nonetheless, we will eventually cover all CG steps in the concluding remarks

of our study on detection mechanisms (see Section 4.2.4 and Figure 14 in particular).

Third, we consider soft errors occurring as bit-flips on the operands of the considered step. Note that this choice introduces a bias as the considered steps may be composed of multiple atomic hardware operations. A bit-flip occurring on the input data (a and b in the above example) may thus be viewed as an early fault while a bit-flip occurring on the output data (c) may be viewed as a late fault. In our model, we thus do not consider intermediate between those. Note once again that in the case of transient faults (focus of this paper), an early bit-flip on a (for instance changing $a = 2$ into $a = 8$) will affect the output data ($c = 10$) and the input data a is set back to its original value. We have studied both such early and late faults and obtained similar results. We therefore focus on early faults in the core of the paper and report the study on late faults in Appendix B. Note that a single transient bit-flip injected on the input data is likely to induce multiple bit-flips on the output data.

We now provide a brief overview on bit-flip arithmetic. If a bit-flip occurs on an operand of value d, its value becomes $d + \delta d$ with a relative perturbation that depends both on the index ℓ of the affected bit and on the original value b_ℓ of this ℓ^{th} bit in the definition of d in (2). The possible relative perturbation $|\delta d|/|d|$ and associated bounds are summarized in Table 1 (see Appendix A for details). Note that the largest relative perturbations are obtained when the

	$\lvert\delta d\rvert/\lvert d\rvert$		
original value	$\ell = 63$	$52 \leq \ell \leq 62$	$0 \leq \ell \leq 51$
$b_\ell = 0$	2	$2^{2^{\ell-52}} - 1$	$\dfrac{2^{\ell-52}}{1+m} \leq \dfrac{1}{4}$
$b_\ell = 1$	2	$\dfrac{1}{2} \leq 1 - 2^{-2^{\ell-52}} \leq 1$	$\dfrac{-2^{\ell-52}}{1+m} \leq \dfrac{1}{2}$

Table 1: Relative perturbation and associated bounds with a bit-flip on the ℓ^{th} bit depending on its original value b_ℓ.

bit-flip affects a bit in the exponent that was originally equal to zero ($b_\ell = 0$).

3 Study of the sensitivity of PCG to soft errors

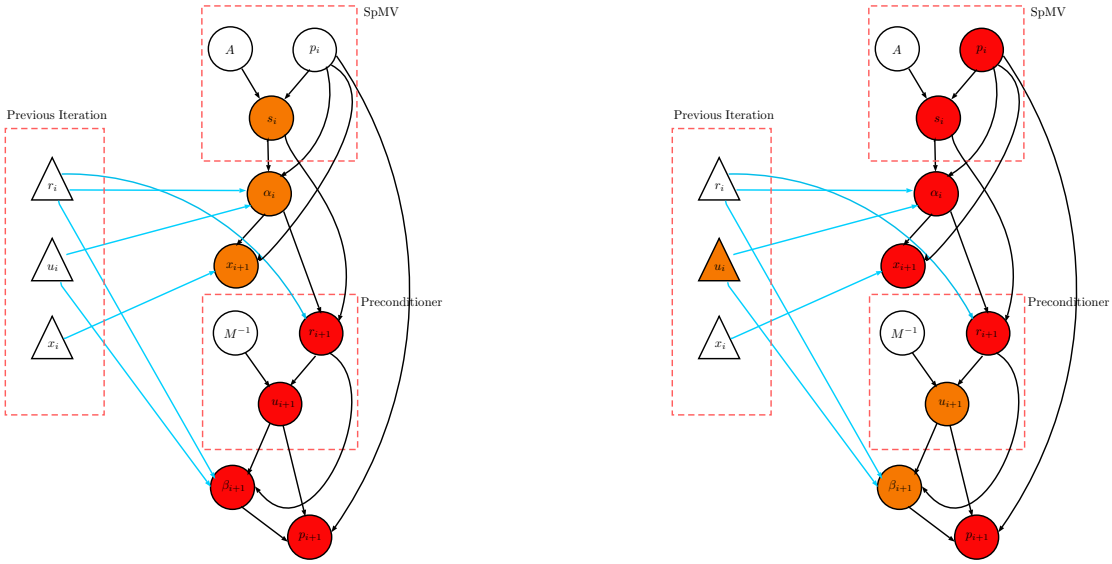
3.1 Propagation of bit-flips in PCG

As discussed above, we are interested in the possible impact of a transient bit-flip that might occur on data computed by the algorithm. Because the most computationally intensive numerical kernels are the matrix-vector product and the preconditioner application we only consider bit-flip in these two key steps of the algorithm. An additional motivation to focus on those two kernels is that the other calculations are mainly cheaper BLAS-1 operations that could be protected by DMR at an affordable extra computational cost. The propagation of the bit-flip in Algorithm 1 is as follows:

1. **Transient error in the matrix-vector calculation.** Assuming a transient error occurs at step 3 of Algorithm 1 during iteration i, after that step, the quantity s_i gets altered, while A and p_i are not. It implies that the current iterate (x_{i+1}) is computed using a corrupted scalar (α_i) and a non-corrupted vector (p_i) whereas the computed residual (r_{i+1}) is computed using both a corrupted scalar (α_i) and vector (s_i).

2. **Transient error in the preconditioner application.** Assuming a transient error at step 7 of Algorithm 1 during iteration i, the next updated iterate (x_{i+2}) in iteration $i+1$ is computed using a corrupted scalar (α_{i+1}) and a consistently corrupted vector (p_{i+1}). Similarly, the iteratively computed residual (r_{i+2}) is updated using the corrupted scalar (α_{i+1}) and corrupted vector (s_{i+1}).

The propagation of transient errors located in the matrix-vector product and in the preconditioner application have therefore a different impact on the quantities computed by CG, possibly introducing different effects on its numerical behavior. Figure 2 shows the data flow in the main PCG loop as well as associated corrupted quantities when a transient soft error appears in the matrix-vector product (left) or preconditioning (right). In these graphs a vertex corresponds to a computing task and an edge to data dependencies between those. The orange color indicates that the task is performed with one corrupted input variable and red is used when more than one input variable is affected by a previous error.



(a) Transient error in matrix-vector product.

(b) Transient error in preconditioner application.

Figure 2: Propagation of transient faults in the PCG algorithm.

3.2 Fault injection protocol

We consider the following parameters to vary the bit-flip locations in time and space. For each matrix and preconditioner application we vary:

1. the time at which the fault occurs: we use nine sample time locations ranging from 10 % up to 90 % of the time interval required for CG to converge (without fault) using the stopping criterion defined by (3) with a prescribed threshold;

2. the index of the corrupted entry in the input vector p_i (r_{i+1}) for the matrix-vector product (resp. the preconditioning application); for that, we randomly select 50 different indices in $[1, n]$;

3. the index of the bit in the 64-bit sequence of the IEEE-754 double precision representation; any of the 64 bits can be flipped.

Note that once the input vector p_i (or r_{i+1}) has been altered, we compute the altered output vector s_i (or u_{i+1}) with step 3 (or 7) and we set back the original value for the input vector p_i (or r_{i+1}) to model a transient fault. We also assessed two alternative modelings of transient faults following the methodology exposed in Section 2.4.2. For instance, in the case of the matrix-vector product, instead of altering p_i, (i) we considered altering an entry of the matrix A itself (and set it back once the updated output s_i vector gets altered) or (ii) we directly altered the output vector s_i. As the obtained results did not significantly change the observed general behavior, we do not report results with those alternative fault injection models for a matter of conciseness. We refer the reader to the Appendix B where the results associated with errors directly injected on the output are reported.

We furthermore consider two threshold values, referred to as low accuracy with $\varepsilon_1 = 10^{-5}$ and high accuracy with $\varepsilon_2 = 10^{-10}$, for the stopping criterion

$$\frac{\|b - Ax_{i+1}\|_1}{\|b\|_1} \leq \varepsilon_{\ell=1,2} \tag{3}$$

for all the test matrices used for the study, listed in Table 2. We consider two classical basic

ID	**Name**	**Size**	**Norm**	**Cond**
1	bcsstk09	1083	6.7e7	9.5e3
2	mesh2e1	306	3.8e2	2.9e2
3	nos5	468	5.8e5	1.1e4
4	lund_b	147	7.4e3	3.0e3
5	mesh1e1	48	9.1	5.2
6	bodyy4	17546	8.3e2	8.0e2
7	Muu	7102	8.3e-4	7.6e1
8	crystm01	4875	5.3e-12	2.3e2
9	fv2	9801	4.5	8.8
10	fv3	9801	4	2.0e2
11	fv1	9801	4.5	8.8

Table 2: Some numerical properties of the test matrices.

preconditioners: Incomplete Cholesky factorization (ICC) and Jacobi diagonal preconditioning (JAC) [23]. Finally, to possibly see the effect of the norm of A, that amplifies the soft-error in p_i, we also consider the case where the matrix is scaled by its 1-norm so that the scaled matrix is of norm one. The fault injection protocol is sketched in Figure 3 for a soft error in the matrix-vector product. In all experiments reported in this manuscript, we systematically consider that a single transient fault is injected per run. With this sampling of the parameter space on the selected data set, several millions of experiments have been run to produce the results reported in the sections dedicated to numerical experiments.

3.3 Numerical experiments

We now study the sensitivity of PCG algorithm with respect to soft error locations in time and space. As bit-flip injections impact the convergence behavior of PCG, the stopping criterion may be reached in a different number of iterations or even be prevented. We thus consider that a

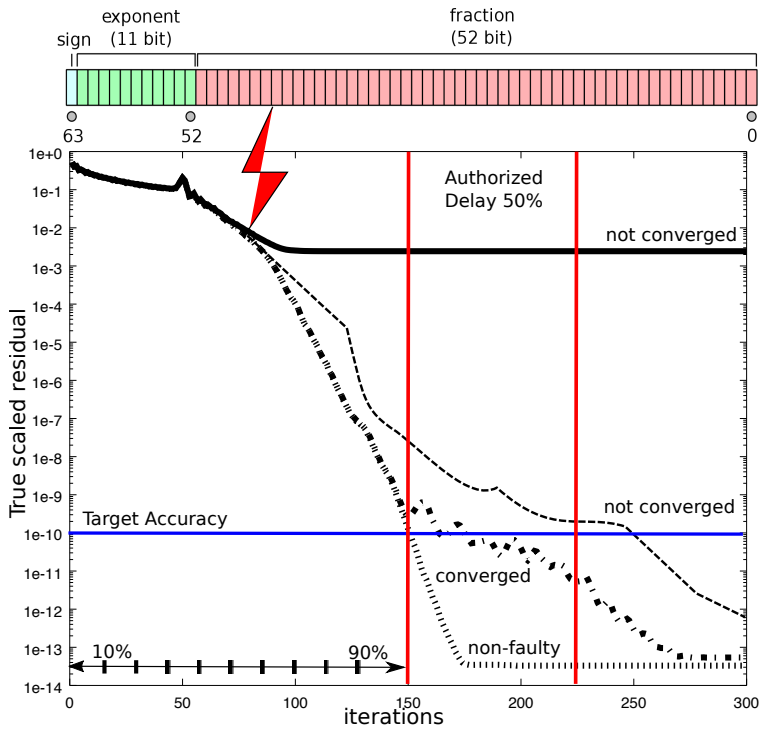


Figure 3: The error is injected in the input vector at any entry, any bit and at any time in the iterative process for either the matrix-vector product or the preconditioner calculation.

faulty execution converges if it achieves the same stopping criterion as the non-faulty execution with some authorized delay in term of number of iterations. In this work, we allow for 50% extra iterations (with respect to the non-faulty execution) to decide whether or not the soft error has prevented PCG to converge. We mention that we varied this extra iteration bound and it did not significantly change the statistical results as the number of non converging cases would remain roughly the same. Some of the bit-flips translate into NaN (Not a Number). Such a situation is accounted for as a non-converged run in the proposed sensitivity analysis (Section 3) and will be considered as a detected fault in the study of detection mechanisms (Section 4).

3.3.1 Effect of the bit-flip value

Figure 4 shows the percentage of successfully converged executions of PCG when a soft error is injected in the matrix-vector product as a function of the index of the flipped bit in the 64-bit sequence of a double precision floating point number for ICC (left), JAC (center) or non preconditioned (right) cases. We display in different rows the experiments associated with bit-flips where the value of the bit was originally zero (one), referred to as "$0 \rightarrow 1$" (resp. "$1 \rightarrow 0$") in this figure. As it could have been expected from the bound on the relative perturbation size reported in Table 1, one can notice that flipping a bit in the exponent from zero to one damages convergence more than a bit moving from one to zero. This trend is particularly visible in the first two rows of the graphs, for the less stringent stopping criterion.

3.3.2 Soft errors in the matrix-vector product

Figure 5 shows the percentage of successfully converged executions of PCG when a soft error is injected in the matrix-vector product as a function of the index of the flipped bit. We report both results on low ($\varepsilon = 10^{-5}$ in (3), top row of the figure) and high ($\varepsilon = 10^{-10}$, bottom row) target accuracy. The bits flipped in the exponent or in high order bits of the mantissa very often prevent CG to converge, especially when a high accuracy is targeted. We also observe that bit-flips in the low-order bits in the mantissa do not prevent CG to converge for any of the two threshold values ε_ℓ; of course the index of the bit from which no effect is observed is lower for large value of ε_ℓ. We also depict the influence of the fault injection time in Figure 6. Those results show that early faults have a larger impact than late ones; this behavior is somehow coherent with the results presented in [3, 4, 26] in the context of inexact Krylov solvers where the inexactness in the matrix-vector calculation can grow as the inverse of the residual norm.

3.3.3 Soft errors in the preconditioner application

A similar experimental study was conducted for assessing the impact of transient faults in the preconditioner application. The impact of the index of the flipped bit is reported in Figure 7, while the influence of the fault injection time is reported in Figure 8. One interesting observation is the lower negative impact of soft errors in the preconditioner application compared to errors in the matrix-vector product. For most of the experiments, a flipped bit in the mantissa does not prevent PCG to converge. To confirm this observation, we performed experiments using the identity as a preconditioner, which reduces to unpreconditioned CG when no fault is injected. Comparing the subsequent results, reported in the last column of Figure 7 with those displayed in the last column of Figure 5, it can be seen that the two graphs exhibit very different behavior. The reason for this significant difference is that the propagation flows of a transient fault occurring in the matrix-vector product and in the preconditioner application are very much different (see Figure 2 and related discussion, above). A deeper theoretical analysis would certainly deserve to be undertaken to better understand this behavior; however, this analysis is out of the scope

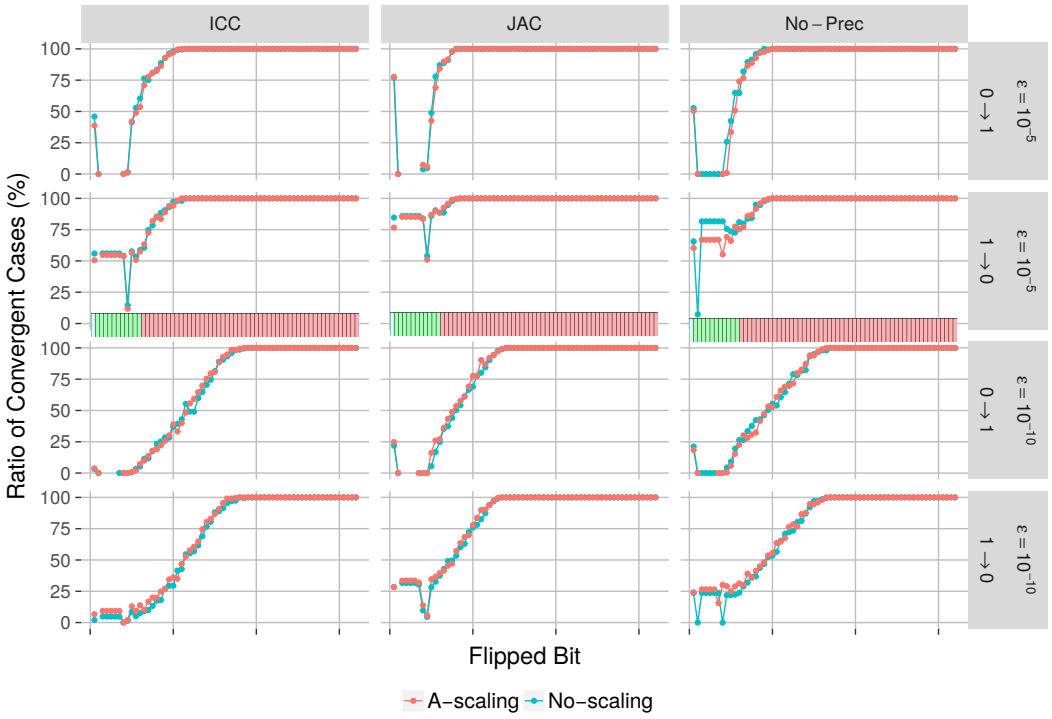


Figure 4: Comparison of the impact on convergence of the bit-flips at originally zero or one bits. The 64-bit indices of the IEEE 754 floating point numbers are displayed between each graph; from left to right, the sign (blue), exponent (green) and mantissa (red) bits are represented. Note that the missing data points at exponents in some figures are mainly due to the numerical range of the polluted vector entries.

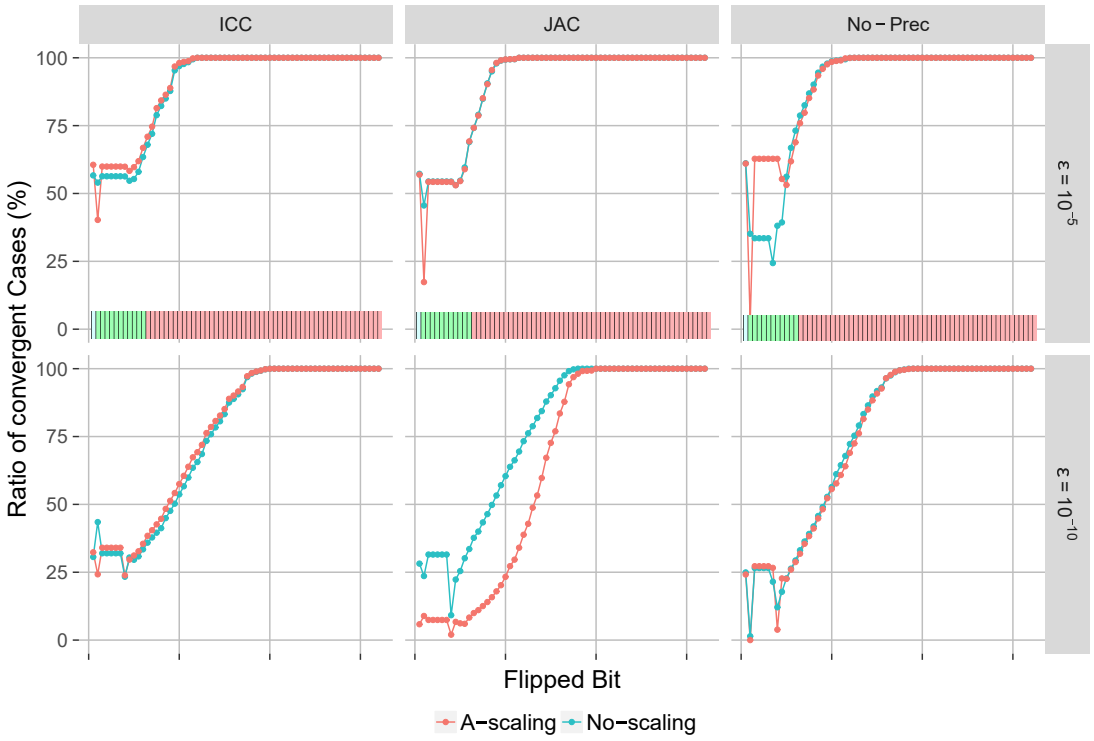


Figure 5: Impact of the index of the flipped bit in the matrix-vector product on PCG convergence success. The 64-bit indices of the IEEE 754 floating point numbers are displayed between each graph; from left to right, the sign (blue), exponent (green) and mantissa (red) bits are represented.

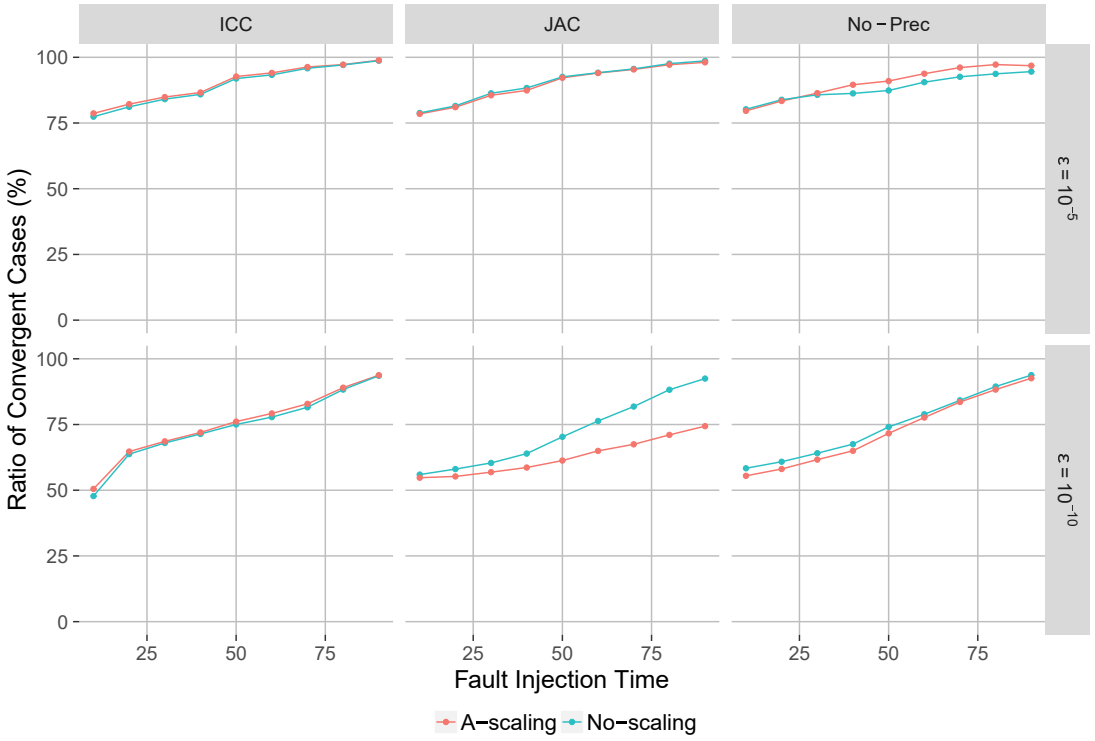


Figure 6: Impact of the fault injection time (as a proportion of the number of iterations with respect to the non-faulty execution) in the matrix-vector product on PCG convergence success.

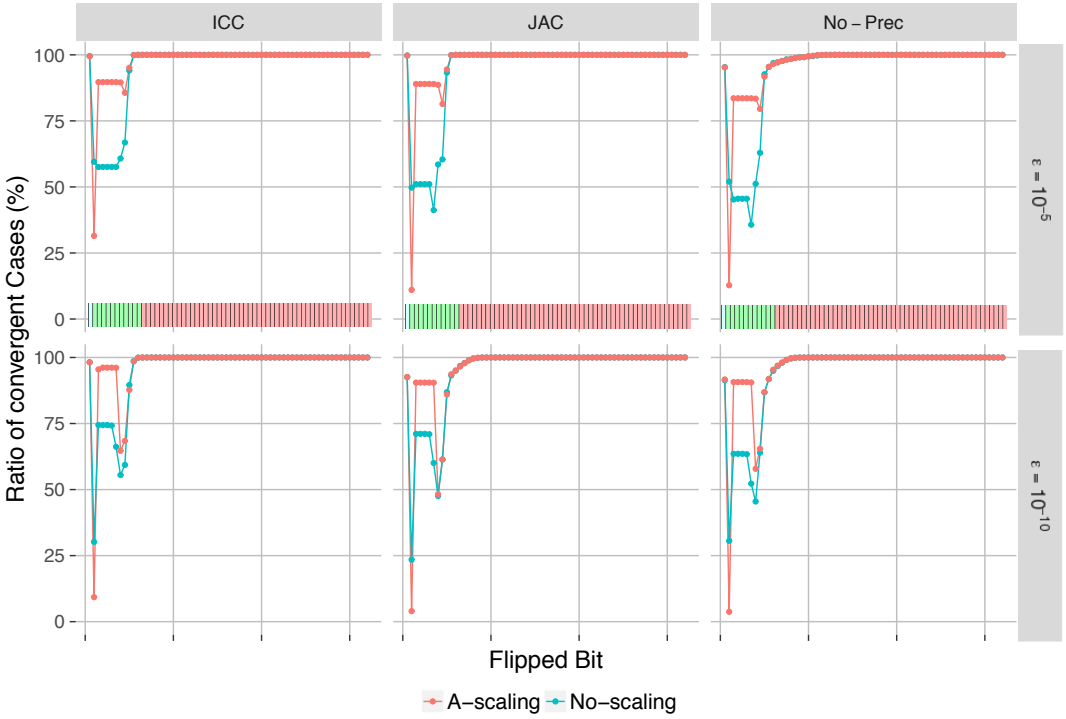


Figure 7: Impact of the index of the flipped bit in the preconditioner application on PCG convergence success.

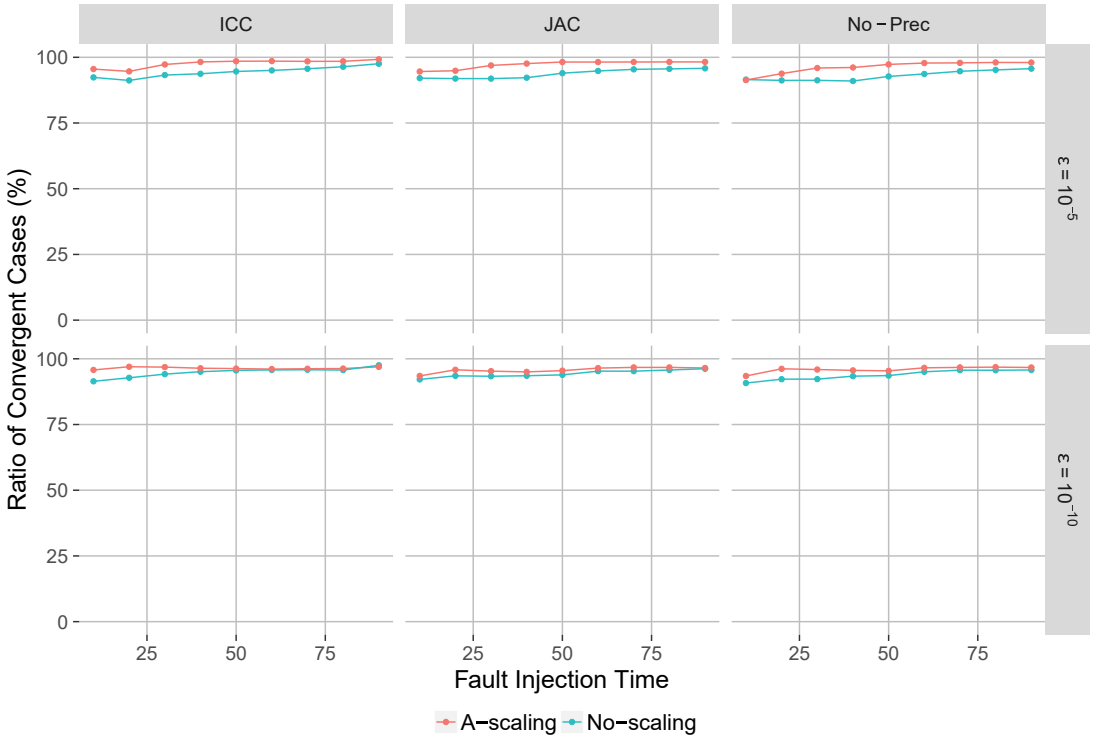
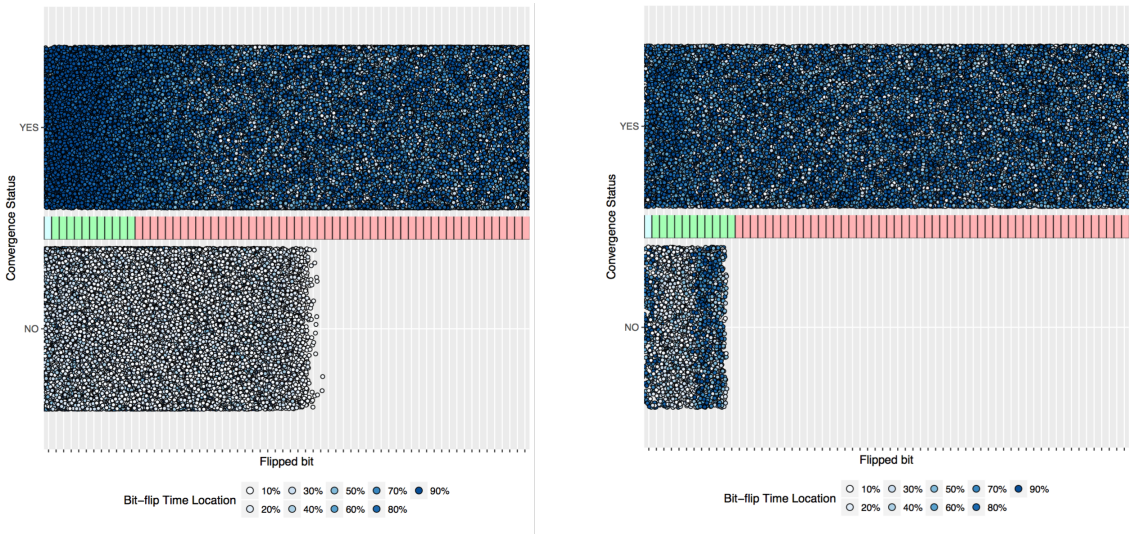


Figure 8: Impact of the fault injection time in the preconditioner application on PCG convergence success.

of the present study. Finally, applying PCG on matrices having a norm equal to one makes the numerical method generally slightly more robust to bit-flip in the exponent. A possible explanation is that in such cases, the values of the entries of the vectors are mostly lower than one, which corresponds to a large number of bits equal to one in their exponents [7]. This trend is more visible for bit-flips in the preconditioner application (see Figure 7) than in the matrix-vector product (see Figure 5).

Another representation of the same results is displayed in Figure 9 where each dot corresponds to a run. The dot location on the x-axis indicates the index of the flipped bit and its color indicates the fault injection time (the darker, the later); dots in the top (bottom) graph correspond to runs where PCG converged (respectively did not converge). This presentation highlights that small perturbations induced by early bit-flips and late large perturbations can damage PCG convergence, while large perturbations in the preconditioner generally have the same effect.



(a) matrix-vector.

(b) preconditioner application.

Figure 9: Distribution of the experimental data for non-scaled/diagonal preconditioner w.r.t. effect on convergence. The blue color scale represents the fault injection time.

3.3.4 Concluding remarks

A few supplementary comments on the experiments presented so far can be made. The first one is that PCG is rather robust and still converges in many cases even when bit-flips occur on exponent or large digits of the mantissa; obviously the less stringent the convergence threshold the more robust PCG is. Much more surprising is that PCG is significantly more sensitive to transient soft errors occurring in the matrix-vector product than to those appearing in the preconditioner application. Finally, in accordance with existing results on inexact Krylov, the earlier the fault appears in the convergence process the larger the impact on the numerical behavior is and large errors close to the convergence might not prevent it to eventually converge.

4 Numerical criteria for detecting soft errors in PCG

4.1 Numerical criteria

The PCG method is a very sophisticated and elegant numerical scheme that has many properties induced by the symmetric positive definiteness of the matrix A, which, in particular, can be used to define a vectorial norm. Among those properties, we can recall the orthogonality between the residual r_i at each iteration or the A-orthogonality of the descent directions p_i. Unfortunately, those properties, as associated characteristic equalities, are no longer valid in finite precision calculation. On the other hand a lot of work has been devoted to study PCG in finite precision, see [21, 20] and references therein. We consider some of the finite precision results to define a first possible soft error detection mechanism based on the residual gap.

4.1.1 Residual gap-based detection

In exact arithmetic, the iteratively computed residual r_i is equal to the true residual defined by $b - Ax_i$ associated with the current iterate x_i; that is

$$r_i - (b - Ax_i) = 0. \tag{4}$$

In finite precision calculation, the computed quantities (denoted with a bar) differ from their exact mathematical values. A first consequence is that $f_i \equiv \bar{r}_i - (b - A\bar{x}_i)$ is not longer zero and defines the gap between the true and the iteratively computed residual referred to as the residual gap that determines the maximal attainable accuracy. Using the rounding error analysis performed in [12, 13], given an initial guess $\bar{x}_0$, the computed vectors satisfy

$$\bar{p}_0 = \bar{r}_0 = b - A\bar{x}_0 + f_0$$

and

$$\begin{aligned}
\bar{x}_i &= \bar{x}_{i-1} + \bar{\alpha}_{i-1}\bar{p}_{i-1} + \delta x_i, && (5)\\
\bar{r}_i &= \bar{r}_{i-1} - \bar{\alpha}_{i-1}A\bar{p}_{i-1} + \delta r_i, && (6)\\
\bar{p}_i &= \bar{r}_i - \bar{\beta}_i\bar{p}_{i-1} + \delta p_i, && (7)
\end{aligned}$$

where the individual δ-terms account for the local round-off errors associated with the different iterative updates.

Using (5) and (6), a recurrence on the residual gap can be derived that accounts for the accumulation of the local round-off errors

$$\begin{aligned}
f_i &= \bar{r}_i - (b - A\bar{x}_i)\\
&= \bar{r}_{i-1} - \bar{\alpha}_{i-1}A\bar{p}_{i-1} + \delta r_i - (b - A(\bar{x}_{i-1} + \bar{\alpha}_{i-1}\bar{p}_{i-1} + \delta x_i))\\
&= f_{i-1} + \delta r_i + A\delta x_i
\end{aligned}$$

so that we have

$$f_i = f_0 + A\sum_{\ell=1}^{i}\delta r_\ell + A\sum_{\ell=1}^{i}\delta x_\ell.$$

The norm of the residual gap between the true and the computed residuals has been intensively studied [12, 29, 27]. Assuming the standard model of floating point arithmetic with machine

precision ϵ, see, e.g. [12, 29, 27], it is shown in [29] that the following upper bound holds for the norm of the residual gap

$$\|f_i\| \leq \epsilon \left(m||A|| \sum_{\ell=0}^{i} ||\bar{x}_\ell|| + \sum_{\ell=0}^{i} ||r_\ell|| \right), \tag{8}$$

where m corresponds to the maximal number of non-zero entries in the rows of the matrix A. This bound on the residual gap can be used to detect soft errors that have larger effects than the one predicted by the worse case scenario of the rounding error analysis. It can be assessed on a periodic basis, for instance every other *CheckPeriod* iterations, as illustrated at lines 11 to 16 in Algorithm 3.

4.1.2 α-based detection

Among the numerous relationships that exist between the quantities computed by PCG, there is one that is possibly less prone to defection due to finite precision calculation because it is a bound and not a strict equality as for the orthogonality or A-orthogonality properties. It was already presented in the original paper on CG [16, Thm 5.5],

$$\forall i \quad \frac{1}{\lambda_{\max}} < \alpha_i < \frac{1}{\lambda_{\min}} \tag{9}$$

where $\lambda_{\max}$ ($\lambda_{\min}$) denotes the largest (respectively the smallest) eigenvalue of A, or the preconditioned matrix. From a practical view point, the calculation or the tight approximation of $\lambda_{\min}$ is generally expensive while $\lambda_{\max}$ can often be cheaply approximated using for instance randomized techniques [14]. Consequently, we only consider the lower bound to define our α-based detection mechanism, which is cheap to check at each iteration to possibly detect that an error occurred, as illustrated at lines 6 to 8 in Algorithm 3. The overall PCG algorithm equipped with both residual gap and α-based detection is given in Algorithm 3.

4.2 Numerical experiments

In this section, we aim at evaluating the robustness and genericity of the numerical detection mechanisms described in the previous sections (we do not consider here the full DMR technique). We consider the same experimental framework as the one used in Section 3.3 except that we discard the experiments where the bit-flips translate into a NaN (such errors are generally captured by the operating system and cannot be considered as silent errors).

4.2.1 Checksum-based detection

Although the well-known checksum technique can be applied for protecting both the matrix-vector product and the preconditioner application, we first focus on its capabilities in the case of bit-flip in the matrix-vector calculation. As indicated in Section 2.3, a threshold τ has to be defined that should comply with two conflicting constraints that are: be large enough to reduce as much as possible the false-positives and be small enough to detect all the errors. To illustrate the lack of robustness of the checksum in finite precision in the context of an iterative solver, we slightly broaden the classical terminology to characterize fault detection mechanisms. Consequently we also consider experiments without fault to evaluate the rate of false-positives detected by this criterion. Table 3 discusses the color codes used in this section; the first four rows correspond to the classical taxonomy of the outcomes of any kind of decision methodology that we extend in the last two rows to allow for a finer analysis; they correspond to the situations where the

Algorithm 3 PCG enhanced with both residual gap-based and α-based detection

Require: **A**, b, x_0,**M**, λ_{max}, *CheckPeriod*.
1: $r_0 := b - Ax_0$; $u_0 = M^{-1}r_0$; $p_0 := r_0$
2: $f_0 = \epsilon(||r_0|| + m||A||||x_0||)$
3: **for** $i = 0, \ldots$ **do**
4: $s_i := Ap_i$
5: $\alpha_i := r_i^T u_i / s_i^T p_i$
6: **if** $\alpha_i < \frac{1}{\lambda_{max}}$ **then**
7: CreateDetectionAlert()
8: **end if**
9: $x_{i+1} := x_i + \alpha_i p_i$
10: $r_{i+1} := r_i - \alpha_i s_i$
11: $f_{i+1} = f_i + \epsilon(||r_{i+1}|| + m||A||||x_{i+1}||)$
12: **if** $mod(i, CheckPeriod) == 0$ **then**
13: **if** $||r_{i+1} - (b - Ax_{i+1})|| > f_{i+1}$ **then**
14: CreateDetectionAlert()
15: **end if**
16: **end if**
17: $u_{i+1} = M^{-1}r_{i+1}$
18: $\beta_{i+1} := r_{i+1}^T u_{i+1} / r_i^T u_i$
19: $p_{i+1} := u_{i+1} + \beta_{i+1} p_i$
20: **end for**

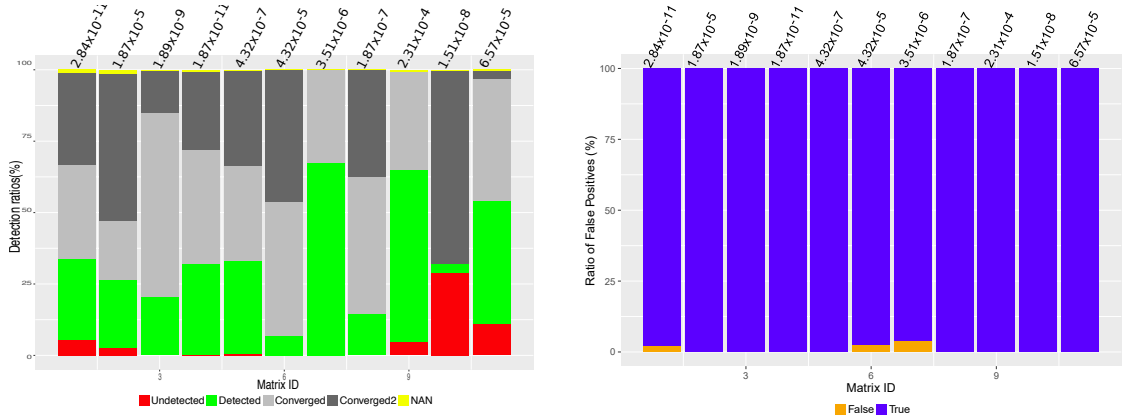
Color	Term	Explanation
	true-positive	A fault that prevented convergence has occurred and the detector did raise an alert
	false-negative	A fault that prevented convergence has occurred convergence and the detector did not raise an alert
	true-negative	No fault has occurred and the detector did not raise an alert
	false-positive	No fault has occurred and the detector did raise an alert
	-	A fault that did not prevent convergence has occurred and the detector did raise an alert
	-	A fault that did not prevent convergence has occurred and the detector did not raise an alert

Table 3: Corresponding terms for color codes.

criterion detects (or not) a soft error that does not prevent PCG to converge. In a more binary setting, the light grey could be interpreted as a false positive (that in our case will be reserved to situations where no error was injected while the detector raises an alert) and the dark grey one could have been seen as a false-negative. More precisely, dark grey corresponds to runs where PCG did converge despite a soft error but the checksum criterion did not detect it, light grey are runs with PCG convergence in which the checksum detects the soft error, green are runs where PCG did not converge and the soft error was detected by the checksum. Red represents runs where PCG did not converge and the checksum did not detect the soft error. This is the worst situation that characterizes the lack of robustness of the checksum criterion. The orange part indicates the ratio of false-positives resulting from the checksum criterion, that detects fault that do not exist and finally the blue color corresponds to the case non-faulty execution without any

alert from detection mechanism.

In Figure 10a, we report on experiments where the checksum threshold parameter has been optimally tuned for each individual matrix to minimize the sum of the false-positives and the missed errors; that is, the two situations where the criterion gives wrong information. For this purpose, in addition to our faulty runs, we also perform additional non faulty experiments varying the right-hand sides. The abscissa corresponds to the matrix index as defined in Table 2 and the values on top of the bars are the individual threshold values τ. One can see that we still miss the detection of some errors that prevent PCG to converge (red part). In Figure 10b, we use the optimal threshold parameters τ to further perform an additional thousand experiments without faults but varying randomly the right-hand sides. Looking at experiments on Matrix 1



(a) Faulty experiments.

(b) Non-faulty experiments.

Figure 10: Ratio of missed errors (a) and false-positives (b) with thresholds τ (top of the bars) optimized to minimize the sum of the missed errors and false-positives for each matrix.

(first bar in the two plots), one can see that, even a posteriori, it it not possible to define a perfect threshold value that discards both the missed errors (in red in Figure 10a) and the false-positives (in orange in Figure 10b). Those two graphs illustrate the lack of reliability and robustness of the checksum-based criterion in finite precision arithmetic. Consequently, we do not study it any further to detect soft-error in the preconditioner application.

4.2.2 Residual gap-based detection

In this section we investigate the robustness of the fault detection mechanism based on the residual gap. As discussed in Section 1 and illustrated in Figure 2, one could expect that soft errors in the matrix-vector product will very likely create a larger residual gap than predicted by the theoretical bound that only accounts for the worse case induced by round-off. The faults in the preconditioner calculation generate corrupted quantities that similarly affect the computed residual and the current iterate, so that the corresponding error mostly vanishes in the residual gap. Consequently, we first consider the experiments where the faults are injected in the matrix-vector product. Furthermore, because we want to design criteria able to detect faults that prevent PCG to converge, we only consider non converging runs in our analysis below.

It would not make sense to check the residual gap at each iteration, as the required matrix-vector product would be as costly as DMR to protect the matrix-vector calculation. Consequently, we investigate the robustness of three policies: either we check this criterion only periodically

(every other *CheckPeriod* = 10 iterations in our experiments, as in Algorithm 3, corresponding to the blue plot in Figure 11) or only when exiting PCG (when it exceeded the maximal number of iterations, green plot in Figure 11), or both periodically and on exit of PCG (red plot in Figure 11). Figure 11 shows the ratio of undetected faults for these three detection policies as a function of the bit-flip index. We consider experiments with both non scaled and scaled A since the norm of the matrix appears in the upper bound of the residual gap (see Equation (8)). It can be observed that the strategy that combines a periodic checking with a final one is the most effective; it successfully detects all the faults in the matrix-vector operation that prevent PCG to converge. The defect of the policy that performs the check only when exiting PCG is that the upper bound in Equation (8) continuously increases along the iteration so that the actual residual gap, which becomes larger than the bound after a fault, ultimately ends up below the upper bound. Symmetrically, the policy that only checks periodically the bound misses the faults that occurs between the last check and the exit of PCG when it exceeds the maximum number of iterations. Note in particular that, in the case of ICC at low accuracy (top left quadrant), the number of iterations until convergence is often smaller than the period, making the final check even more important (as it is the only one performed in those cases).

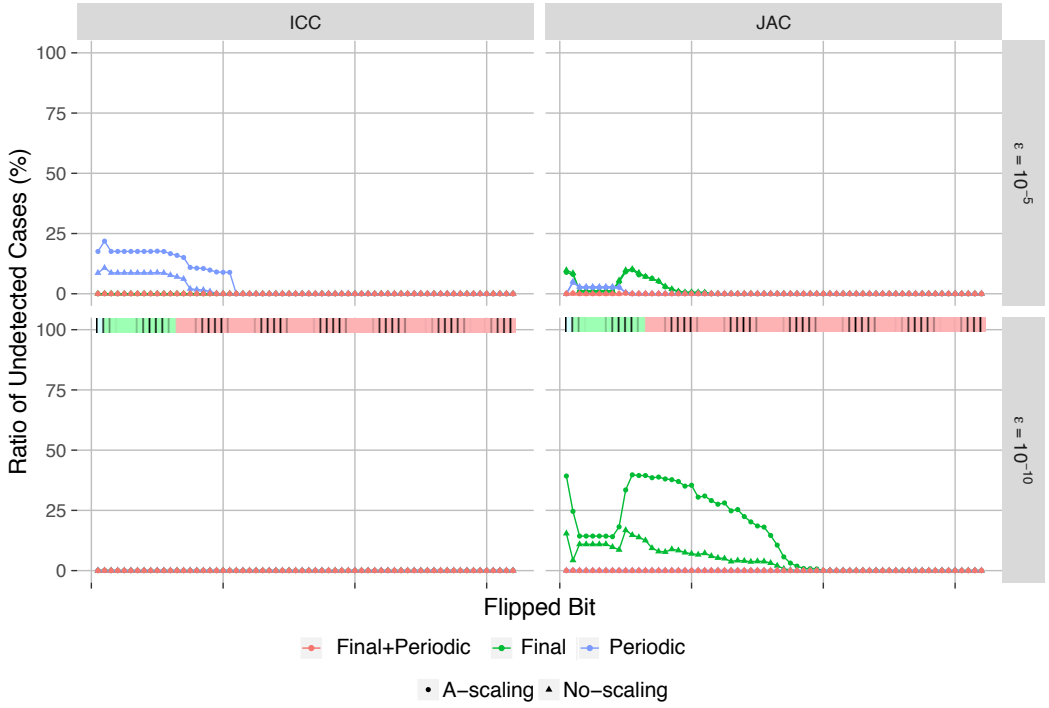


Figure 11: Detection performance of the three policies based on the residual gap criterion.

To fully assess the superiority of the residual gap detection over the checksum criterion to detect fault in the matrix-vector product, we the ratio of undetected faults for the non-convergent PCG display in Figure 12. The checksum criterion misses many bit-flips especially when high accuracy is targeted. The number of misses is higher than the ones observed in Figure 10a because we use threshold τ optimized over the set of matrices and not individually as for the experiments

considered in Figure 10a. Those experiments confirm the weaknesses of the checksum criterion and confirm the robustness of the residual gap criterion combined with periodic and final checks.

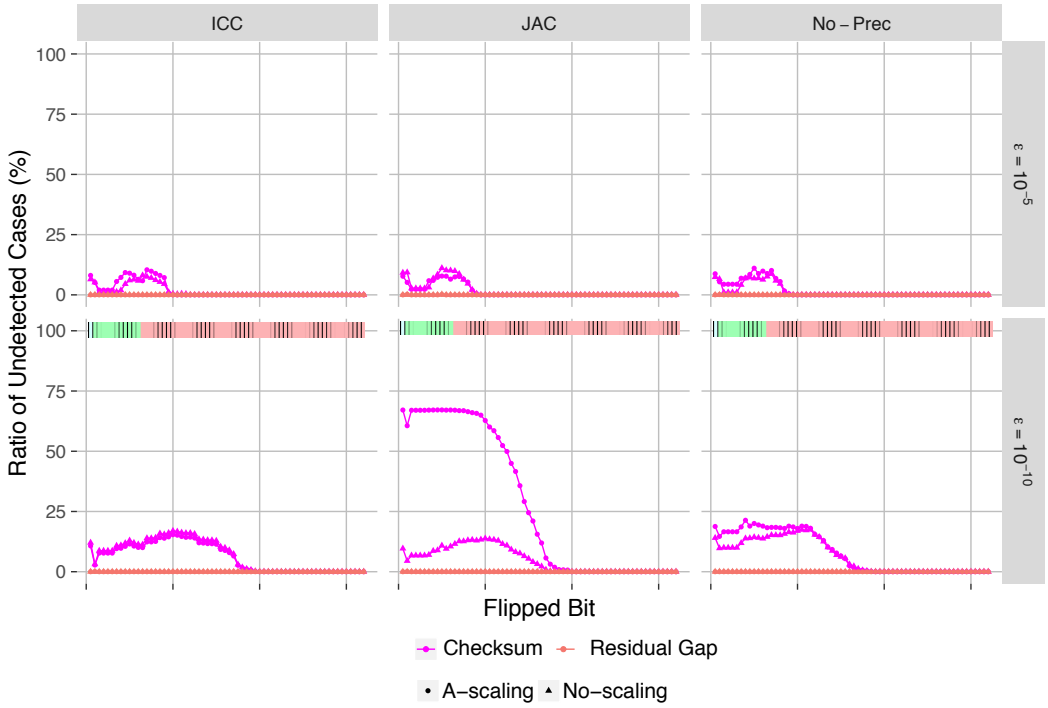


Figure 12: Detection performance of gap deviation and checksum-based methodologies for faults in the matrix-vector calculation.

4.2.3 α-based detection

In this section we study the robustness and reliability of the α-based criterion (given by Equation (9)) to detect possible soft errors in the preconditioner application as it proved to be not effective to detect errors in the matrix-vector calculation. We display the ratio of undetected soft errors as a function of the bit-flip index as well as the ratio for the residual gap criterion in Figure 13. As it could have been expected due to the consistent propagation of the error in the iterate and residual updates, the residual gap criterion is very often ineffective. On the contrary, the α-based criterion appears to be very robust and only missed a very few errors for bit-flips occurring on the low order bit of the exponent.

4.2.4 Concluding remarks

In this section, we have assessed three numerical criteria to detect soft errors: the classical checksum mechanism, a bound on the norm of the residual gap and a bound on the α value. The numerical experiments have revealed the difficulty to define the threshold associated with the checksum in finite precision calculation. In that context, the rounding error analysis of the

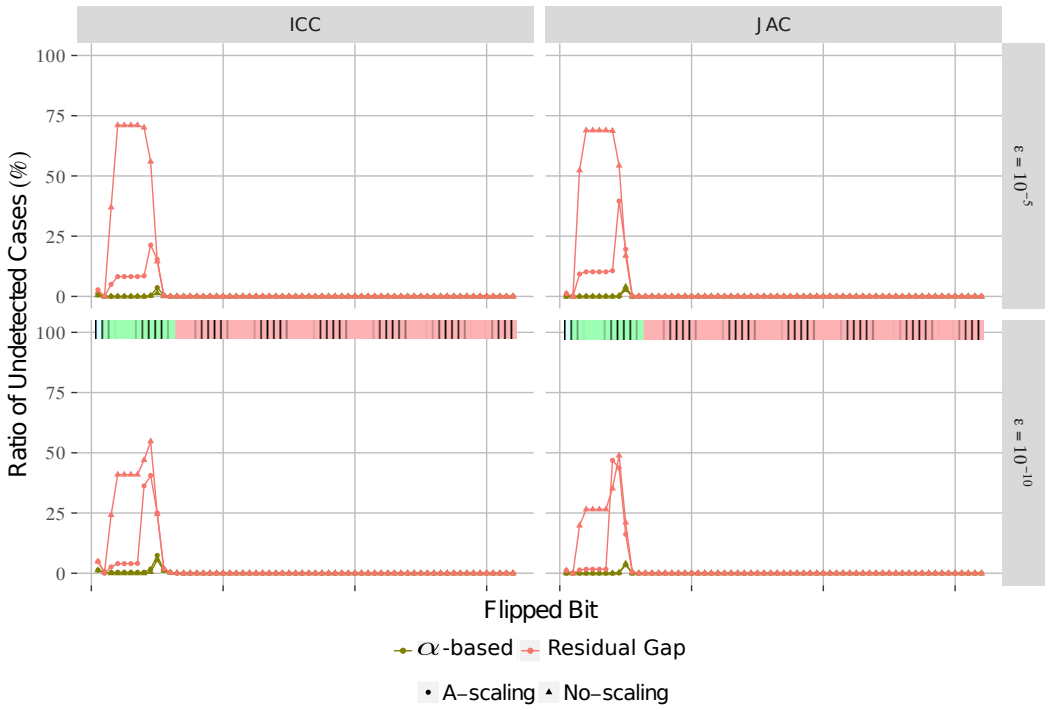


Figure 13: Detection performance of the residual gap and α-based methodologies for faults in the preconditioner calculation.

residual gap measurement provides a robust criterion. This criterion is particularly effective to detect soft errors in the matrix-vector product. Because the residual gap deviation bound is based on solid theoretical results, no false-positives can exist in non-faulty executions. Finally, the criterion based on α allows the detection of most of the errors in the preconditioner application. The combination of these last two criteria enables us to equip the PCG algorithm with numerical techniques to detect errors that do not suffer from false-positives; consequently they are robust and reliable. A possible drawback of the α criterion is that it requires knowledge of the largest eigenvalue; we note that some numerically scalable multi-level preconditioning techniques provide this information (see [1] and reference therein).

One could naturally wonder whether these two detection mechanisms, which are robust criteria for detecting soft errors in the matrix-vector product and preconditioner application, would also be a robust criterion to detect errors occurring in the other steps of the PCG algorithm. In that respect, we performed an additional extensive set of experiments by injecting faults in all steps of Algorithm 1. We report in Figure 14 the outcome of these experiments with the detection success ratio for detecting faults injected in each of these individual steps: for the residual gap criterion, the α-based criterion or the combination of both criteria as expressed in Algorithm 3. Several comments can be made. First, regarding the sensitivity of faults occurring in step 9; altering an entry of the descent direction does not prevent PCG from converging. Although possibly surprising, this is consistent with the observation made in Section 3.3.3, that is, the weak sensitivity of PCG to soft-errors in the calculation of the preconditioned residual. Second, the ability of the residual gap detection to catch a soft error occurring in the computation of α_i (step 4) or the iterate update (step 5) is remarkable. Finally, we point out the relatively large impact on the robustness of PCG to recover from an error in the computation of β_i (step 8) that subsequently affects the entire descent direction. A more detailed sensitivity analysis would certainly deserve to be carried out to better study soft errors and possible detection mechanisms for this step, which is the main source of undetected faults.

5 Conclusion and perspectives

In this paper we have experimentally investigated the robustness of PCG to transient soft-errors in its (usually) most time consuming kernels: the preconditioner and matrix-vector product. As it could have been expected we observed that soft errors affecting the exponent have a more detrimental impact on the convergence of PCG than low order bits in the mantissa. Surprisingly, we noticed that PCG was more robust to soft errors in the preconditioner that in the matrix-vector calculation. Based on these observations we proposed numerical criteria that aim at detecting soft errors that prevent PCG to converge. In particular, we illustrated that the classical checksum approach, based on equalities in exact arithmetic, lacks robustness even with tuned thresholds. Alternatively, criteria based on the residual gap and on the range of validity of the values of α, allow for the definition of robust and safe numerical mechanisms. Future works will consist in using those criteria to design a self-correcting (or *self-stabilizing* using the terminology of [24]) PCG algorithm and in investigating similar approaches for modern variant of PCG such as pipelined PCG [11] as well as extending our work to non-symmetric Krylov subspace solvers.

Acknowledgments

This work has been funded by the EXA2CT European Project on Exascale Algorithms and Advanced Computational Techniques, which receives funding from the EU's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 610741. Experiments presented in this

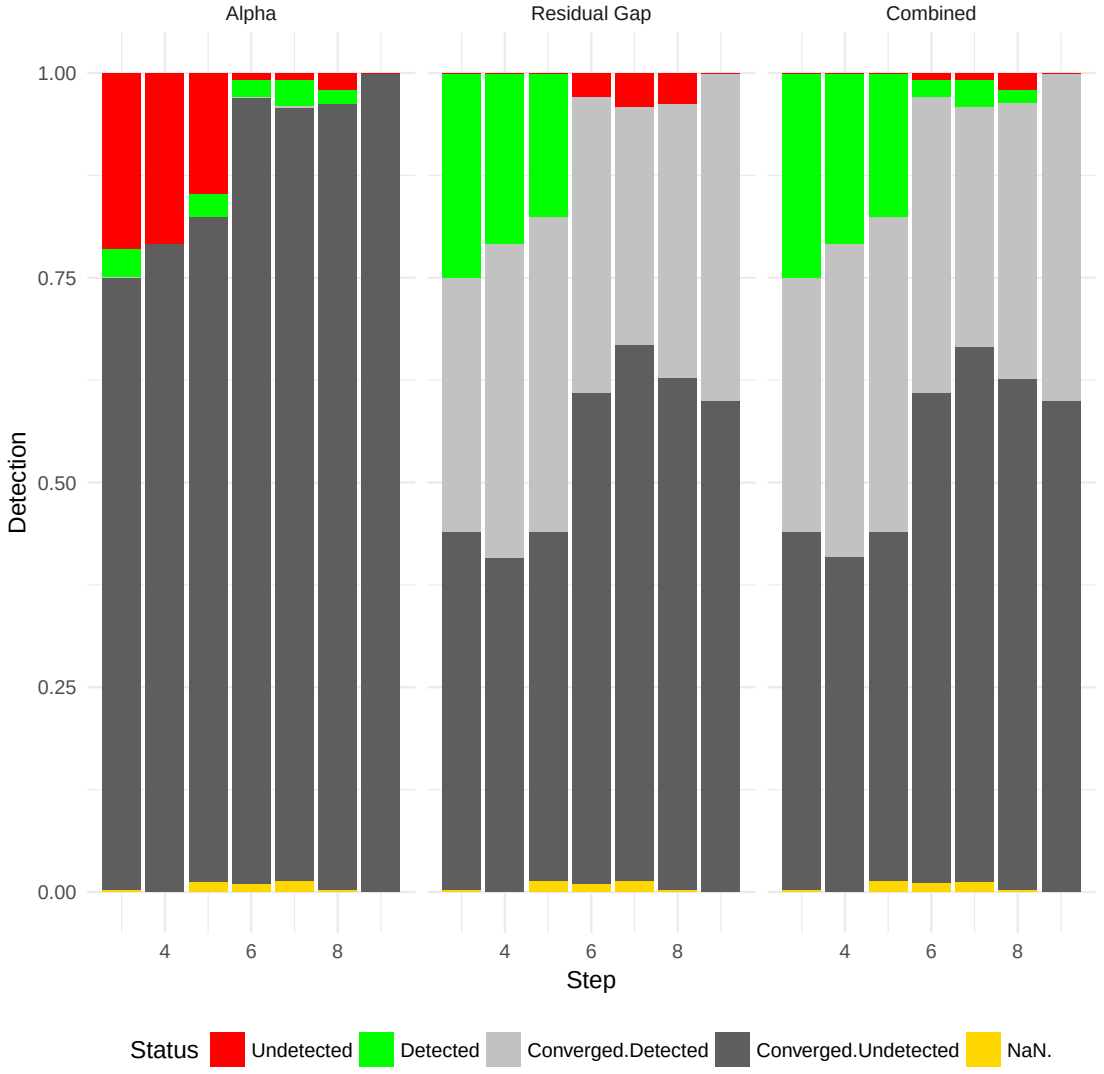


Figure 14: Comparison of the detection success of gap-based, alpha-based and combined methodologies for the other steps of PCG. The experimental set is different from the other experiment due to the random selection of the index of the corrupted entry. Because of that, the sensitivity to the soft-error at step-7 is slightly different from the results depicted in figures 7 and 8.

paper were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see https://www.plafrim.fr/). Siegfried Cools acknowledges funding by the Research Foundation Flanders (FWO) under grand number 12H4617N.

References

[1] Emmanuel Agullo, Luc Giraud, and L Poirel. Robust coarse spaces for abstract Schwarz preconditioners via generalized eigenproblems. Research Report RR-8978, INRIA Bordeaux, November 2016.

[2] Wesley Bland, Aurelien Bouteiller, Thomas Hérault, George Bosilca, and Jack J. Dongarra. Post-failure recovery of MPI communication capability: Design and rationale. *IJHPCA*, 27(3):244–254, 2013.

[3] A. Bouras and V. Frayssé. Inexact matrix-vector products in Krylov methods for solving linear systems: a relaxation strategy. *SIAM Journal on Matrix Analysis and Applications*, 26(23):660–678, 2005.

[4] A. Bouras, V. Frayssé, and L. Giraud. A relaxation strategy for inner-outer linear solvers in domain decomposition methods. Technical Report TR/PA/00/17, CERFACS, Toulouse, France, 2000.

[5] J. Elliott, F. Mueller, M. Stoyanov, and C. Webster. Quantifying the Impact of Single Bit Flips on Floating Point Arithmetic. *Technical Report 2013-2*, pages 1–13, 2013.

[6] James Elliott, Mark Hoemmen, and Frank Mueller. Evaluating the impact of sdc on the gmres iterative solver. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 1193–1202. IEEE, 2014.

[7] James Elliott, Mark Hoemmen, and Frank Mueller. Resilience in Numerical Methods: A Position on Fault Models and Methodologies. 2014.

[8] The MPI Forum. Mpi: A message passing interface standard version 3.1, June, 2015.

[9] Manuel Le Gallo, Abu Sebastian, Roland Mathis, Matteo Manica, Heiner Giefers, Tomas Tuma, Costas Bekas, Alessandro Curioni, and Evangelos Eleftheriou. Mixed-precision in-memory computing. *Nature Electronics*, 1:246–253, 2018.

[10] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidyalingam S Sunderam. *PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing*. MIT press, 1994.

[11] P. Ghysels and W. Vanroose. Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm. *Parallel Computing*, 40(7):224–238, 2014.

[12] Anne Greenbaum. Estimating the attainable accuracy of recursively computed residual methods. *SIAM Journal on Matrix Analysis and Applications*, 18(3):535–551, 1997.

[13] Martin H. Gutknecht and Zdenek Strakoš. Accuracy of two three-term and three two-term recurrences for Krylov space solvers. *SIAM Journal on Matrix Analysis and Applications*, 22(1):213–229, 2000.

[14] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[15] Thomas Herault and Yves Robert, editors. *Fault-Tolerance Techniques for High-Performance Computing*. Springer International Publishing, 2015.

[16] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 46(6):409–436, December 1952.

[17] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.

[18] Mark Hoemmen and Ma Heroux. Fault-tolerant iterative methods via selective reliability. *Proceedings of the 2011 International ...*, 2011.

[19] Kuang-hua Huang and Jacob a Abraham. Algorithm-Based Fault Tolerance for Matnx Operations. *IEEE Transactions on Computers*, c(6):518–528, 1984.

[20] J. Liesen and Z. Strakoš. *Krylov Subspace Methods*. Numerical Mathematics and Scientific Computation. Oxford University Press, 2013.

[21] Gérard Meurant and Zdenek Strakoš. *The Lanczos and conjugate gradient algorithms in finite precision arithmetic*, volume 15. Cambridge University Press, 2006.

[22] Behrooz Parhami. Defect, fault, error,..., or failure? *IEEE Transactions on Reliability*, 46(4):450–451, 1997.

[23] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[24] Piyush Sao and Richard Vuduc. Self-stabilizing iterative solvers. *Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems - ScalA '13*, pages 1–8, 2013.

[25] Manu Shantharam, Sowmyalatha Srinivasmurthy, and Padma Raghavan. Characterizing the impact of soft errors on iterative methods in scientific computing. *Proceedings of the international conference on Supercomputing - ICS '11*, page 152, 2011.

[26] V. Simoncini and D. B. Szyld. Theory of inexact Krylov subspace methods and applications to scientific computing. *SIAM Journal Scientific Computing*, 25:454–477, 2003.

[27] Z. Strakoš and P. Tichy. Error estimation in preconditioned conjugate gradients. *BIT Numerical Mathematics*, pages 789–817, 2005.

[28] Henk A. Van der Vorst. *Iterative Krylov methods for large linear systems*. Cambridge monographs on applied and computational mathematics. Cambridge University Press, Cambridge, UK, New York, 2003.

[29] Henk A. van der Vorst and Qiang Ye. Residual replacement strategies for Krylov subspace iterative methods for the convergence of true residuals. *SIAM Journal Scientific Computing*, 22(3):835–852, 2000.

[30] Raoul Velazco, Pascal Fouillat, and Ricardo Reis. *Radiation effects on embedded systems*. Springer Science & Business Media, 2007.

A Details of the computation conducted for the values presented in Table 1

Here are the details of the computation conducted to obtain the values in Table 1, with $\tilde{d} = d + \delta d$:

1. Sign bit-flip $\ell = 63$
   (a) init value $b_\ell = 0, 1$ $\tilde{d} = -d \Rightarrow |\delta d| = |2d|$
2. Exponent bit-flip $52 \leq \ell \leq 62$
   (a) init value $b_\ell = 0$ $\tilde{d} = 2^{2^{\ell-52}} d \Rightarrow \frac{|\delta d|}{|d|} = 2^{2^{\ell-52}} - 1$
   (b) init value $b_\ell = 1$ $\tilde{d} = 2^{-2^{\ell-52}} d \Rightarrow \frac{|\delta d|}{|d|} = 1 - 2^{-2^{\ell-52}} \leq 1$
3. Mantissa bit-flip $0 \leq \ell \leq 51$
   (a) init value $b_\ell = 0$ $\tilde{d} = i(-1)^{b_{63}} 2^{e-1023} \left(1 + m + 2^{\ell-52}\right) \Rightarrow \frac{|\delta d|}{|d|} = \frac{2^{\ell-52}}{1+m}$ with worse case $\ell = 50 \Rightarrow \frac{|\delta d|}{|d|} \leq 1/4$.
   (b) init value $b_\ell = 1$ $|\delta d| \Rightarrow \frac{|\delta d|}{|d|} = \frac{|-2^{\ell-52}|}{1+m}$ with worse case $\ell = 51 \Rightarrow \frac{|\delta d}{|d|} \leq 1/2$.

B Effect of bit-flip at output vector of SpMV

A possible model for simulating the bitflip can be altering the output vector of the matrix-vector operation. We have not explored exhaustively this option that can be viewed as a localized transient error. Nevertheless some experiments have been conducted on the SPMV only by altering an entry of the output vector s_i in Alg.1, step 3.

We display in Figure 15 the sensitivity of PCG to such faults, that can be compared to the graphs depicted in Figure 5 for transient fault in the input data. The observed trends are similar in both cases.

We also investigated the robustness of the two numerical criteria, namely the residual gap and the alpha-based criterion for soft error injected in the output vector. The results, to be compared with those given in Figure 12, are depicted in Figure 16. The same robustness of the residual gap detection mechanism can be observed in both cases.

Although not assessed by numerical experiments, we can expect that simular behaviour would be observed if permanent errors were injected by altering the output of the preconditioner application. Then sensitivity of PCG to such a fault should be simular the ones depicted in Figure 7 and the robustness of the proposed detection mechanisme comparable to what is displayed in Figure 13.

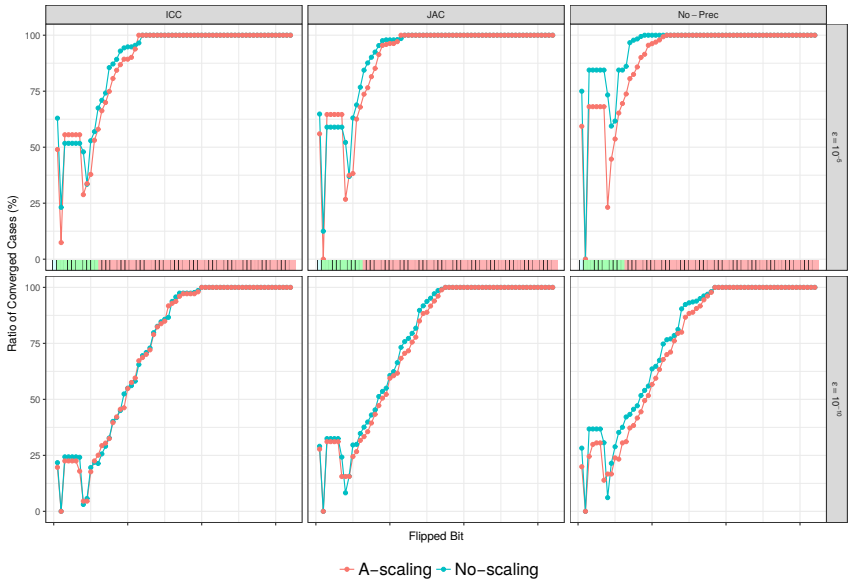


Figure 15: Impact of the index of the flipped bit in the SpMV output vector on PCG convergence success

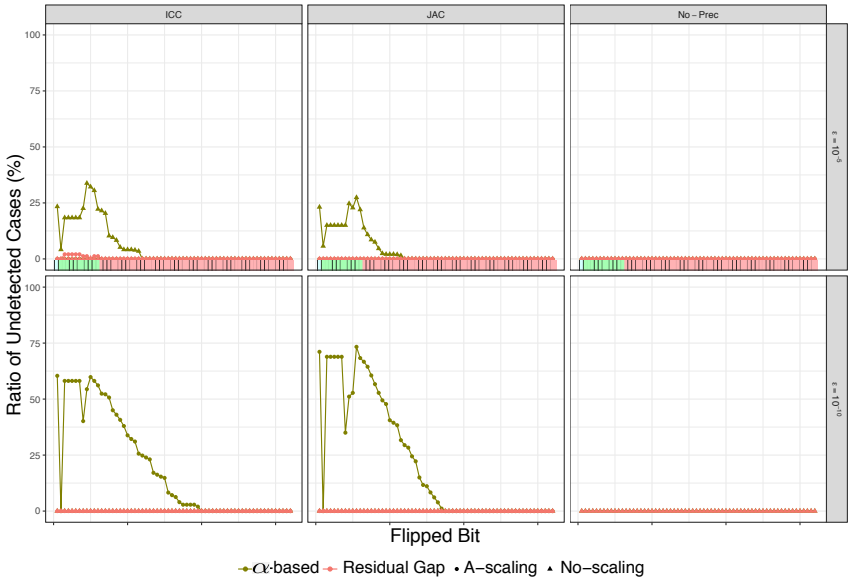


Figure 16: Detection success comparison of deviation and alpha-based methodologies for faults in the output of the matrix-vector calculation.

C Sensitivity of persistent bit-flips on both SpMV and Preconditioner

In that section, we considered persistently altering an entry of the input vector p_i(or r_{i+1}) of SpMV operation (or preconditioner application) given with Alg.1, step 3. Unlike the transient faults, the polluted entry did not set back to its original numerical value. The results are depicted in Figure 17 and Figure 18 for SpMV operation and preconditioner application respectively. Robustness of the detection mechanism is also given in figures 19 and 20. Those results show that, preconditioner application is more sensitive to persistent soft errors than SpMV operation. That seems an exact opposite behaviour to transient case. This situation can be explained by focusing on step 4 (computation of α) of Alg.1 which is given as follows;

$$\alpha_i = \frac{r_i^T u_i}{p_i^T s_i} \tag{10}$$

Let us consider 4 different soft error scenario:

1. **Transient soft error on p_i:** Transient soft-error will be corrected after step 3. Hence, only the output vector s_i will be polluted. The impact on α calculation will not be coherent as given below:
$$\alpha_i = \frac{r_i^T u_i}{p_i^T \textcolor{red}{s_i}} \tag{11}$$
Note that, red color corresponds to polluted vector.

2. **Persistent soft error on p_i:** If polluted entry of p_i kept after step 3, unlike the previous case both vector in the denominator will be polluted. Because of the coherency of the pollution, PCG masked more faulty cases.
$$\alpha_i = \frac{r_i^T u_i}{\textcolor{red}{p_i}^T \textcolor{red}{s_i}} \tag{12}$$

3. **Transient soft error on r_{i+1}:** If we have a transient fault on r_{i+1}, it will be turned into a persistent fault on p_{i+1} at step 9, and consequently the behaviour will be similar to the persistent soft error on p_{i+1} at step 3.

4. **Persistent soft error on r_{i+1}:** The main difference of persistent soft error on r_{i+1} from the transient one is the polluted r_{i+1} vector. Even though we have a more coherent α calculation for that case (all vectors are polluted at one iteration), the next residue (r_{i+2}) will be changed dramatically due to persistently polluted r_{i+1} at step 6.

The impact of permanent bit flip in the matrix-vector product on the robustness of PCG is displayed in Figure 17 (to be compared with results in Figure 5 for the transient faults). It can be seen that the behaviour is very different and actually ressembles very much the effect of transient aoft error in the preconditioner given in Figure 7. This can be explained by the fact that a transient bit flip in the preconditioner application will persistently corrupt p_{p+1} that will be involved in the matrix-vector at the next iteration.

Similarly in Figure 18, we display the robustness of PCG to persistent error in the preconditioner that exhibites very similar behaviour as transient soft error in the matrice-vector product as shown in Figure 5; as these transient soft-errors in the matrix-vector (step 3 in Algorithm1) will translate in a permanent error in the input vector of the preconditioner application (step 7 in Algorithm1).

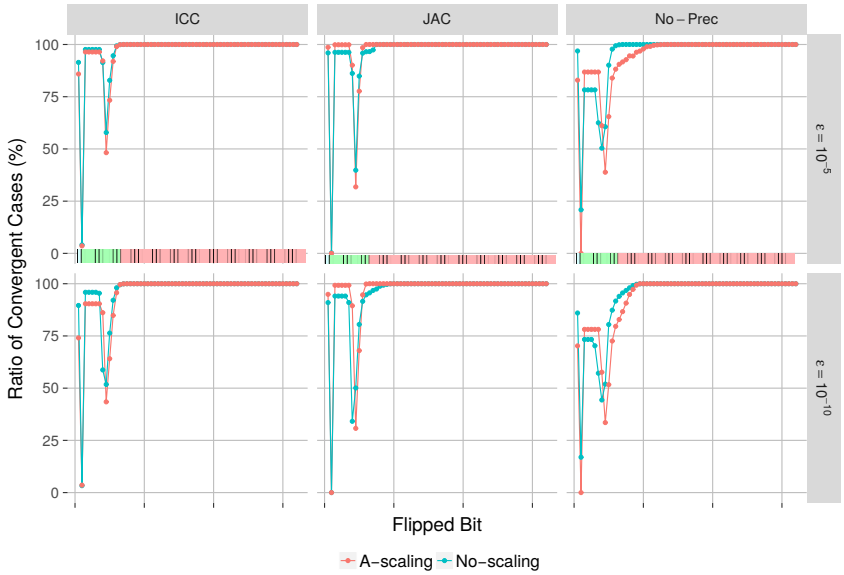


Figure 17: Impact of the index of the persistently flipped bit in the SpMV on PCG convergence success

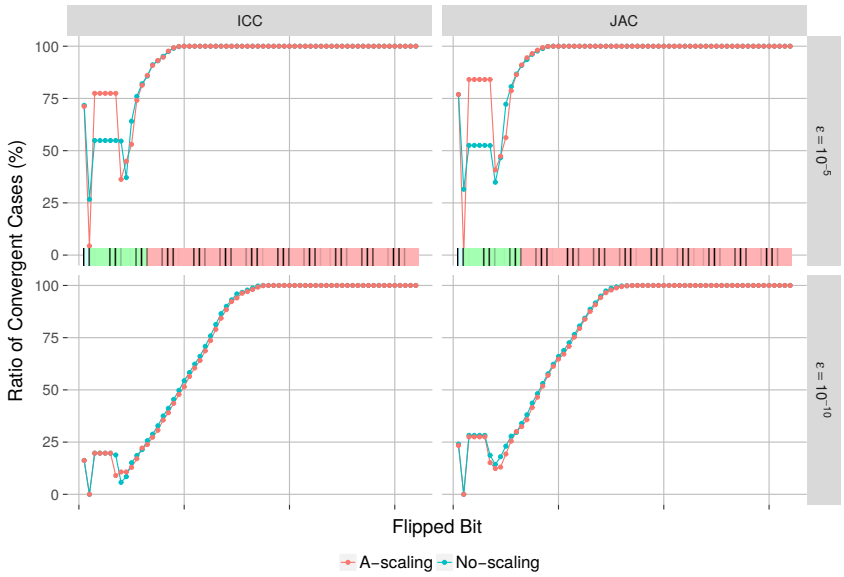


Figure 18: Impact of the index of the persistently flipped bit in the preconditioner application on PCG convergence success.

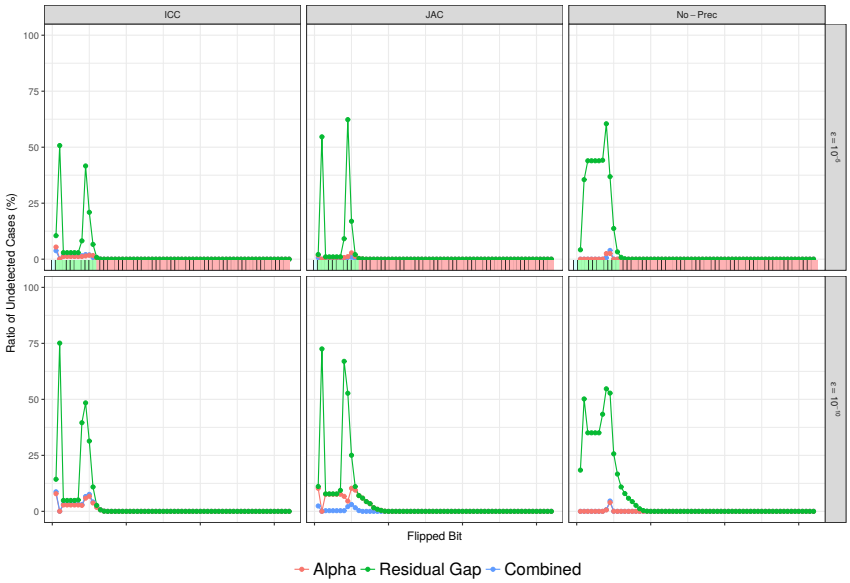


Figure 19: Detection success comparison of deviation and alpha-based methodologies for persistent faults in the matrix-vector calculation.

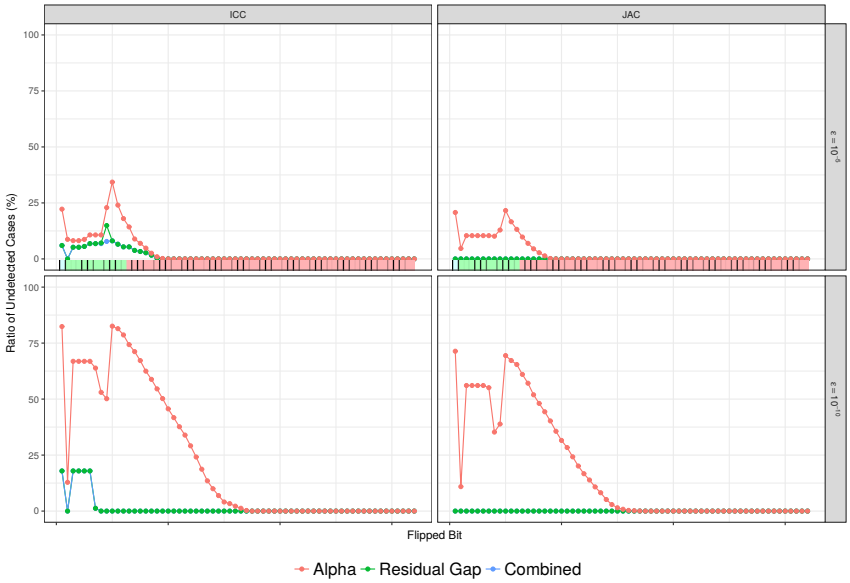
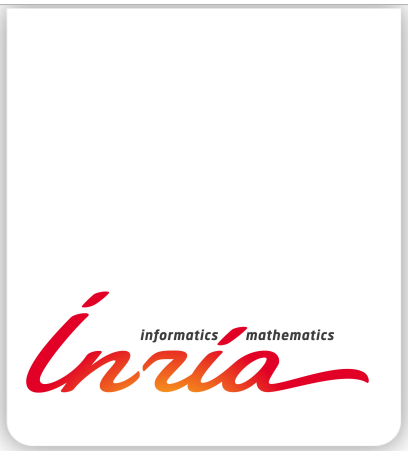


Figure 20: Detection success comparison of deviation and alpha-based methodologies for persistent faults in the preconditioner application.



RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vielle Tour
33405 Talence Cedex

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

ISSN 0249-6399