A complementary note on soft errors in the Conjugate Gradient method: the persistent error case

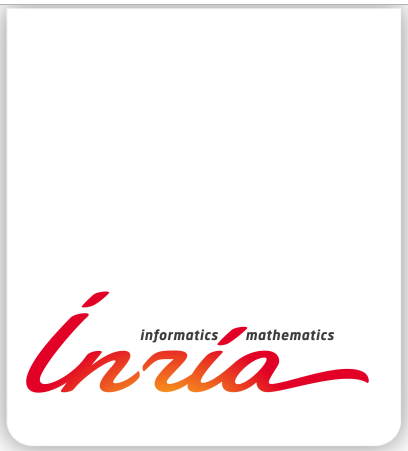
Emmanuel Agullo, Siegfried Cools, Emrullah Fatih-Yetkin, Luc Giraud, Nick Schenkels, Wim Vanroose

A complementary note on soft errors in the Conjugate Gradient method: the persistent error case

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

RESEARCH REPORT
N° 9360
September 2020
Project-Team HiePACS

ISSN 0249-6399 ISRN INRIA/RR--9360--FR+ENG



A complementary note on soft errors in the Conjugate Gradient method: the persistent error case

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

Project-Team HiePACS

Research Report n° 9360 — September 2020 — 17 pages

Abstract: This note is a follow up study to [1], where we studied the resilience of the preconditioned conjugate gradient method (`PCG`). We complement the original work by performing a similar series of numerical experiments, but using what we called persistent instead of transient bit-flips.

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

Une note complémentaire sur les soft errors dans la méthode du Gradient Conjugué: le cas des fautes persistantes

Résumé : Cette note est une étude qui fait suite à [1], où nous avons étudié la résilience de la méthode du gradient conjugué préconditionné (PCG). Nous complétons le travail initial en effectuant une série similaire d'expériences numériques, mais en utilisant ce que nous avons appelé des bit-flips persistants au lieu de transitoires.

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

Contents

1 Introduction

This note is a complementary study to [1], where we investigated the resilience of the preconditioned conjugate gradient method (PCG). In the original work we only considered what we referred to as transient bit-flips, which only affect the output of the calculations but do not definitively corrupt the input data. We looked at the sensitivity of PCG when it is affected by this type of soft error and derived 2 criteria that can be used to detect the occurrence of these bit-flips. In this new study we perform a similar set of experiments, but using what we called persistent bit-flips, i.e., a bit-flip that permanently corrupts the input data. This will allow us to verify whether or not this type of soft error affects the PCG algorithm in a different way, and further test the robustness of the 2 criteria we derived.

In order to keep this note short but self-contained we give a short summary of the original study in Section 2, but refer to [1] for the motivation of this work, most of the details, and further references. In Section 3 we perform a new sensitivity study and in Section 4 we look at the robustness and performance of the 2 detection criteria we proposed in [1].

2 Short summary of the original study

The PCG algorithm [2] shown in Algorithm 1 is still widely used and a prime candidate for extreme-scale computations on large computing platforms. Since these platforms will be more and more prone to errors of different kinds during their calculations, it is of great interest to know the behavior of PCG under various errors and to see whether or not we can detect them. We first limited ourselves to soft errors occurring as bit-flips during the calculations and to the two main computational kernels of the algorithm: the matrix-vector product (step 3) and the preconditioning (step 7).

Algorithm 1 Preconditioned conjugate gradient (PCG)

1: $r_0 := b - Ax_0$; $u_0 = M^{-1}r_0$; $p_0 := u_0$; $\gamma_0 := r_0^T u_0$
2: **for** $i = 0, \ldots$ **do**
3: $\quad s_i := Ap_i$
4: $\quad \alpha_i := \gamma_i / s_i^T p_i$
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: $\quad \gamma_{i+1} := r_{i+1}^T u_{i+1}$
9: $\quad \beta_{i+1} := \gamma_{i+1} / \gamma_i$
10: $\quad p_{i+1} := u_{i+1} + \beta_{i+1} p_i$
11: **end for**

We make the distinction between two types of bit-flips: transient and persistent. Transient errors occur inside the computation kernel, but except for the output of the computation, nothing is corrupted. In a persistent error, the input data will be corrupted as well, affecting all further calculations where it is used. Recalling the example from the original study: if we wish calculate $a+b=c$, then $(a,b,c) = (2,2,10)$ would be an example of a transient error and $(a,b,c) = (2,8,10)$ one of a persistent bit-flip in b.

In [1] we studied the sensitivity of PCG to transient bit-flips with respect to the bit in which the flip occurred using the standard IEEE 754 format and notation, see Figure 2.1, as well as the time at which the bit-flips occurred, i.e., in the first few iterations, in the middle or close to the

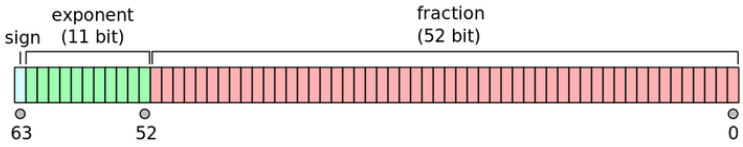


Figure 2.1: IEEE 754 double-precision binary floating-point format.

convergence. As could have been expected, we observed that `PCG` was most sensitive to bit-flips in the higher bits corresponding to the exponent and the sign. Bit-flips in the lower bits often had no effect, with a transition area depending on the preconditioner and the targeted accuracy. The algorithm was also slightly more sensitive to bit-flips that occurred in the early iterations.

We also derived two criteria that could be used to detect the occurrence of these bit-flips. The residual gap-based criterium uses the fact that there is a gap between the recursively calculated residue in `PCG` and the true residual. In exact arithmetic

$$f_i = r_i - (b - Ax_i) = 0,$$

but in finite precision calculations, we have the following upper bound:

$$\|f_i\| \leq \epsilon \left(m \|A\| \sum_{\ell=0}^{i} \|x_\ell\| + \sum_{\ell=0}^{i} \|r_\ell\| \right), \tag{1}$$

where ϵ is the machine precision and m the maximum number of non-zero entries in the rows of A. The second criterium is based on the fact that

$$\forall i \quad \frac{1}{\lambda_{max}} < \alpha_i, \tag{2}$$

with λ_{max} the largest eigenvalue of A.

The bounds presented in Equations (1) and (2) are valid in finite precision arithmetic, so by monitoring them during the algorithm we can detect it when a bit-flip results in them being broken. Algrithm 2 shows has this can be done, with the gap-based criterium on lines 11 to 16 and the α-based on lines 6 to 8. Note that we only perform the gap-based check every `CheckPeriod` iterations in order to avoid computing the true residual every iteration.

When we tested the capabilities of these criteria, we observed that the residual gap-based detection was very good at detecting the transient bit-flips occurring in the matrix vector product, but less efficient at detecting those in the preconditioning step. The opposite was, however, true for the α-based detection, meaning that combined they were able to detect almost all critical faults. When we injected transient bit-flips in every step of `PCG`, and not only in the calculation of the matrix-vector product or the preconditioning step, we observed that combined both criteria could again detect almost all critical faults.

Additionally we compared the capacity of these methods to detect a bit-flip in the matrix-vector product with that of the well know checksum method. We saw that while it is possible to get good results applying a checksum to the matrix-vector product, it can be difficult to determine a good threshold parameter τ, and that the residual gap-based criterium performed much better.

Algorithm 2 `PCG` enhanced with both residual gap-based and α-based detection

Require: **A**, b, x_0,**M**, λ_{max}, *CheckPeriod*.
$r_0 := b - Ax_0$; $u_0 = M^{-1}r_0$; $p_0 := r_0$
$f_0 = \epsilon(||r_0|| + m||A||||x_0||)$
for $i = 0, \ldots$ **do**
  $s_i := Ap_i$
  $\alpha_i := r_i^T u_i / s_i^T p_i$
  if $\alpha_i < \frac{1}{\lambda_{max}}$ **then**
    CreateDetectionAlert()
  end if
  $x_{i+1} := x_i + \alpha_i p_i$
  $r_{i+1} := r_i - \alpha_i s_i$
  $f_{i+1} = f_i + \epsilon(||r_{i+1}|| + m||A||||x_{i+1}||)$
  if $mod(i, CheckPeriod) == 0$ **then**
    if $||r_{i+1} - (b - Ax_{i+1})|| > f_{i+1}$ **then**
      CreateDetectionAlert()
    end if
  end if
  $u_{i+1} = M^{-1}r_{i+1}$
  $\beta_{i+1} := r_{i+1}^T u_{i+1} / r_i^T u_i$
  $p_{i+1} := u_{i+1} + \beta_{i+1} p_i$
end for

3 Study of the sensitivity of PCG to soft errors

3.1 Propagation of bit-flips in PCG

While a bit-flip can happen at any point in the computational kernel and affect for example some intermediate result, we considered only the two extreme cases of transient bit-flips in [1] . We considered early transient bit-flips, i.e., in p_i or r_{i+1}, or late transient bit-flips, i.e., in s_i or u_{i+1}.

In our new numerical experiments we will consider persistent bit-flips. This means that we will inject a bit-flip in p_i or r_{i+1}, but do not reset it to its original value after s_i or u_{i+1} are calculated. Other than this, the experimental setup is identical to that in [1]. This can again be seen as the most extreme case of a persistent bit flip as it will also affect the other steps of `PCG` that use p_i or r_{i+1}. An overview of this is given in Figure 3.1.

3.2 Soft errors in the matrix-vector product

In Figure 3.2 (Figure 3.3 in the transient case in [1]) we show the ratio of convergent cases per bit where the error is injected and based on the value of the original bit. We see that bit-flips from 0 to 1 are far more critical than those going from 1 to 0, which is in line with the theoretical analysis we performed in [1]. In Figure 3.3 (for comparison purpose, see Figure 3.4 for the transient counterpart in [1]) we show a more concise presentation of these results by removing the distinction on the value of the original bit. Finally, in Figure 3.4 (Figure 3.5 for transient in [1]) we show the effect of the fault injection time. Overall, however, we observe behavior that is similar to that of transient bit-flips in the preconditioner (see Figure 3.5 and 3.7 for transient errors in [1]).

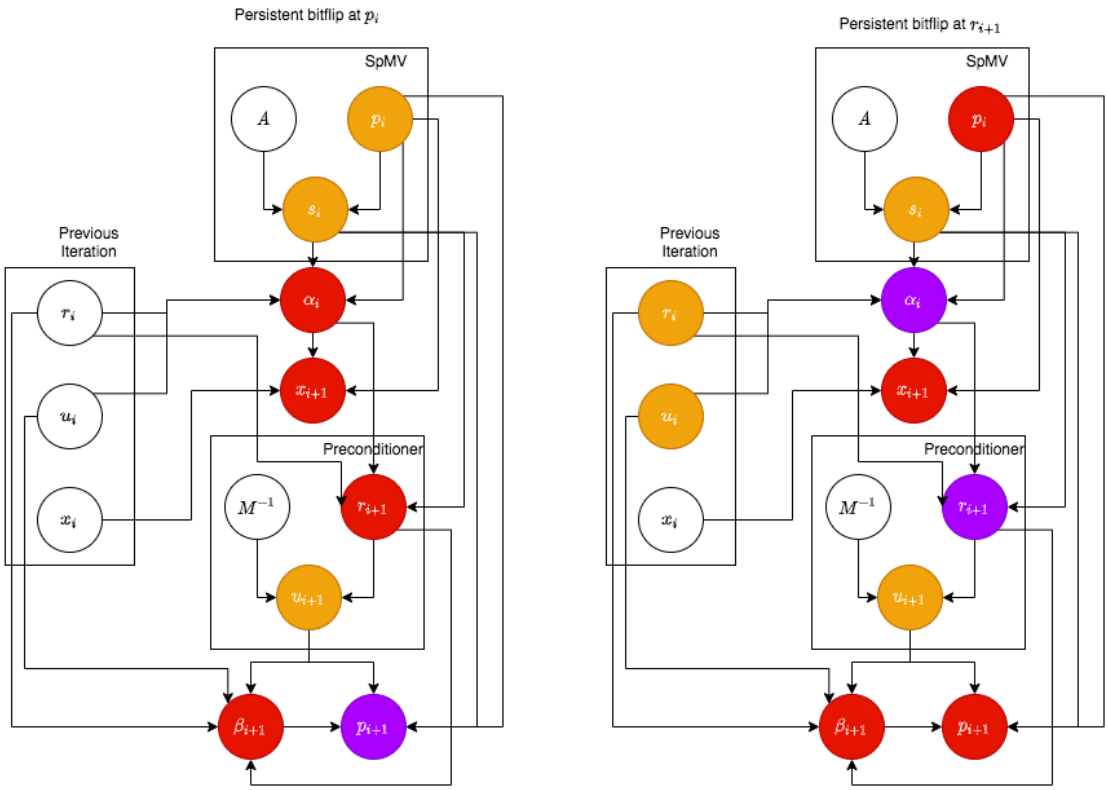


Figure 3.1: Propagation of transient errors in the PCG algorithm. Orange, red, and purple indicate 1, 2, or more than 2 corrupted input variables respectively.

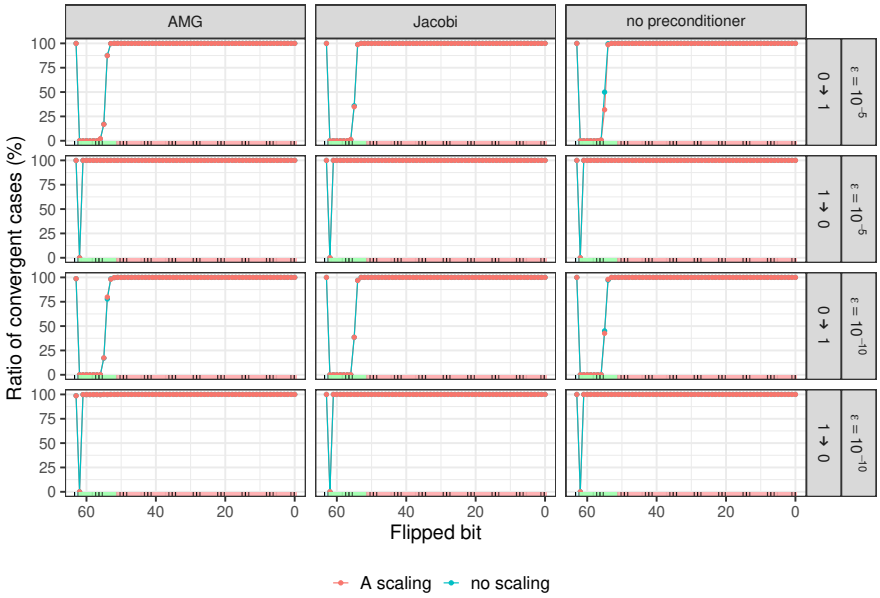


Figure 3.2: Comparison of the impact on convergence of the bit-flips at originally zero or one bits. The 64-bit indices of the IEEE 754 floating point numbers are displayed between each graph; from left to right, the sign (blue), exponent (green) and mantissa (red) bits are represented.

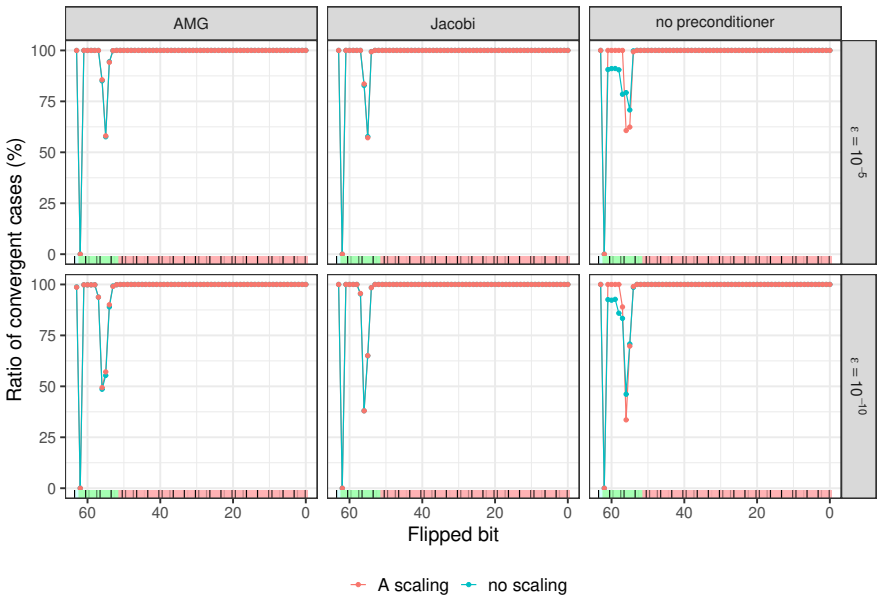


Figure 3.3: Impact of the index of the flipped bit in the matrix-vector product on PCG convergence success. The 64-bit indices of the IEEE 754 floating point numbers are displayed between each graph; from left to right, the sign (blue), exponent (green) and mantissa (red) bits are represented.

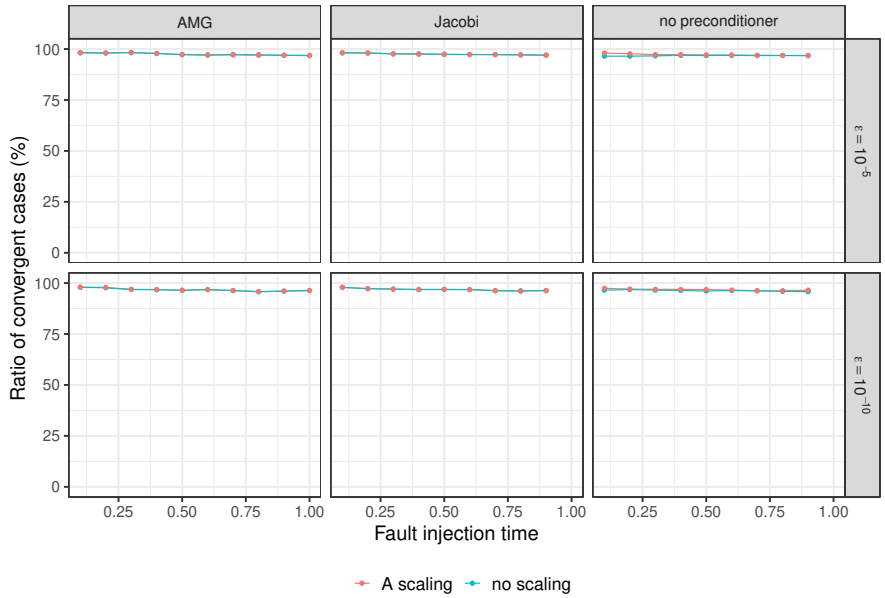


Figure 3.4: Impact of the bit-flip injection time (as a proportion of the number of iterations with respect to the non-faulty execution) in the matrix-vector product on PCG convergence success.

3.3 Soft errors in the preconditioner application

In figures 3.5 (Figure 3.6 for transient errors in [1]) and 3.6 (Figure 3.7 for transient errors in [1]) we show the ratio of convergent cases when we inject a persistent bit-flip in the preconditioner calculation. In this case we see that the convergence of `PCG` closely resembles that of the case of transient bit-flips in the matrix-vector product (see Figure 3.4 and 3.5 in [1]).

3.4 Concluding remarks

When we compare the results for transient and persistent bit-flips, we see that there is a switch in how they effect `PCG`. A transient bit-flip in the matrix-vector product has a very similar effect as a persistent bit-flip in the preconditioner. Similarly, a transient bit-flip in the preconditioner has a very similar effect as a persistent bit-flip in the matrix-vector product.

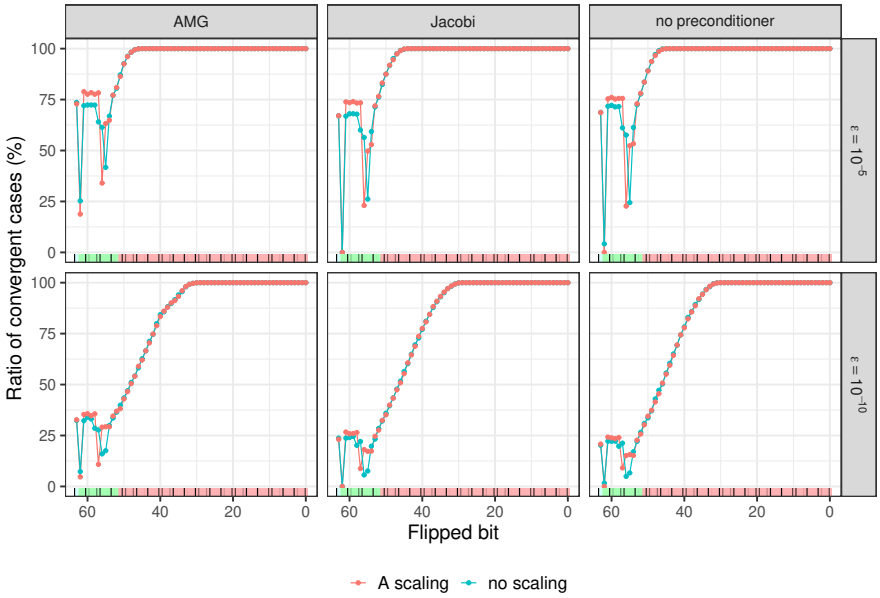


Figure 3.5: Impact of the index of the flipped bit in the preconditioner application on PCG convergence success.

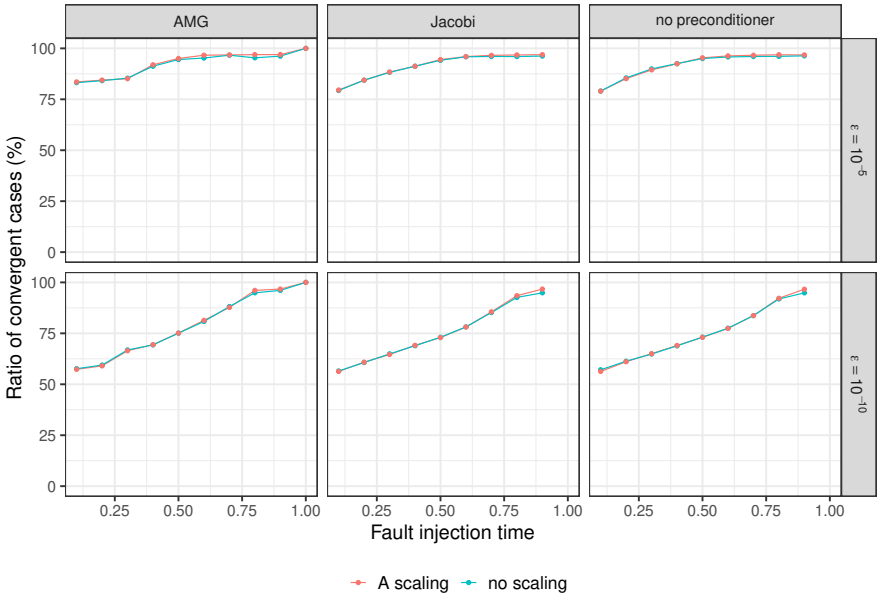
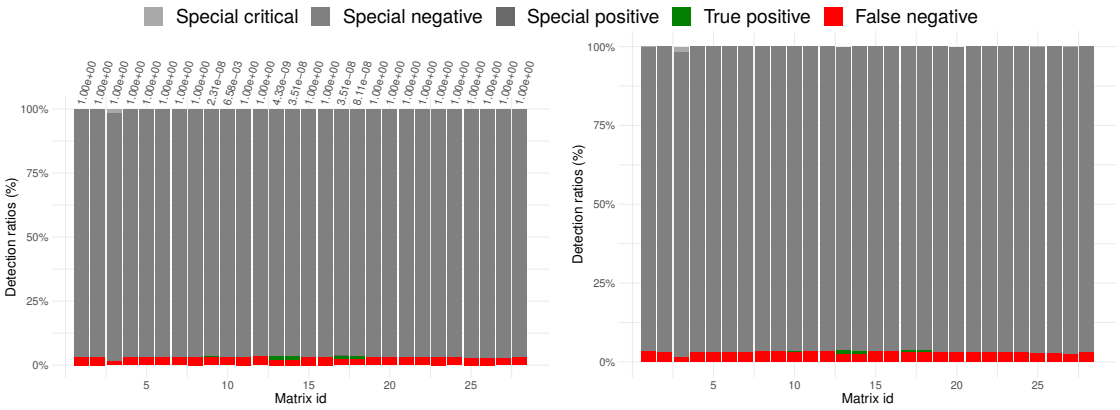


Figure 3.6: Impact of the bit-flip injection time in the preconditioner application on PCG convergence success.

4 Detecting soft errors in PCG

4.1 Checksum-based detection

The checksum detection mechanism relies on an mathematical equality that is not valid in finite precision calculation where a relative threshold τ needs to be introduced to make it practical. The selection of this threshold that comply with two conflicting constraints: be large enough to reduce the false positives and be small enough to limit the number of false negatives. We follow the same optimisation procedure as described in [1, Section 4.2.1] to define the threshold used for the experiments depicted in Figure 4.1a that corresponds to Figure 4.1a in [1] for transient error. It can be observed that while the optimal threshold is not perfect as some false negatives are still triggered and only a few true positive are detected. Figure 4.1b corresponds to Figure 4.2a in [1] for a non-optimal value. Note that due to the different behaviour of the cost function for persistent faults, the optimal value of τ is almost alway 1 in our study, see Appendix A for more details.



(a) Optimal threshold τ.

(b) Non-optimal threshold $\tau = 0.79$.

Figure 4.1: Outcome of the checksum-based detection for the faulty runs for the experiments with a Jacobi preconditioner and $\varepsilon =$ 1e-10 ($\omega = 0.5$). Colour note: the large grey component corresponds to the special negative cases.

4.2 Bit-flips in the matrix-vector product

When it came to transient bit-flips, the residual gap-based criterium performed very good, whereas the α-based criterium did not. In Figures 4.2 (Figure 4.4 for transient errors in [1]) and 4.3 we can see, however, that for persistent bit-flips the opposite is true. Similar as with our results from the sensitivity study we see that the behavior observed in bit-flips in the matrix-vector product and the preconditioner is opposite between transient and persistent bit-flips.

4.3 Bit-flips in the preconditioner

Where the α-based criterium was very efficient in detecting transient bit-flips in the preconditioner, it is less effective at detecting persistent ones as it can be seen in Figure 4.4 (Figure 4.5 for transient errors in [1]). The residual gap-based criterium, however, is now much more effective at detecting those. This follows the same pattern as we observed in the previous experiments,

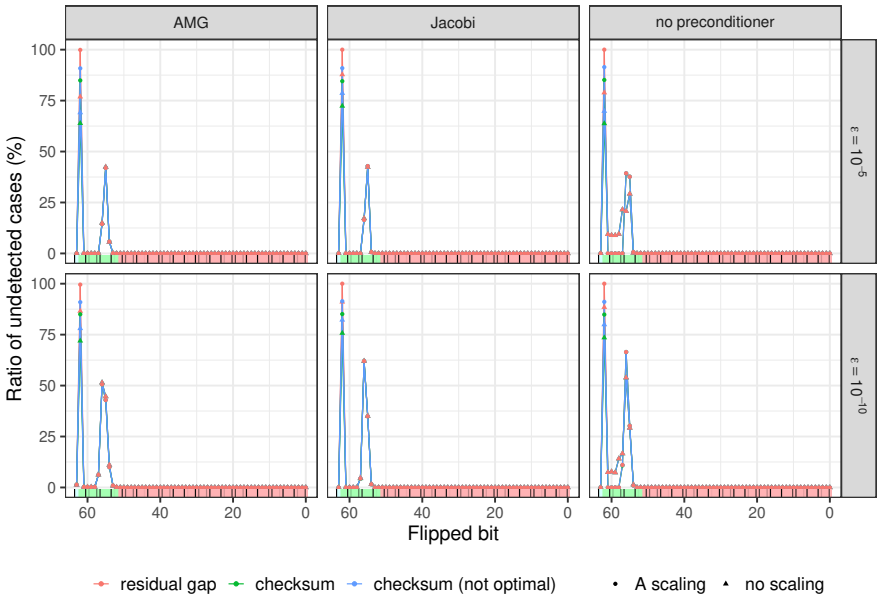


Figure 4.2: Detection performance of residual gap deviation and checksum-based methodologies for soft errors in the matrix-vector calculation with persistent faults in p_i for the matrix-vector product.

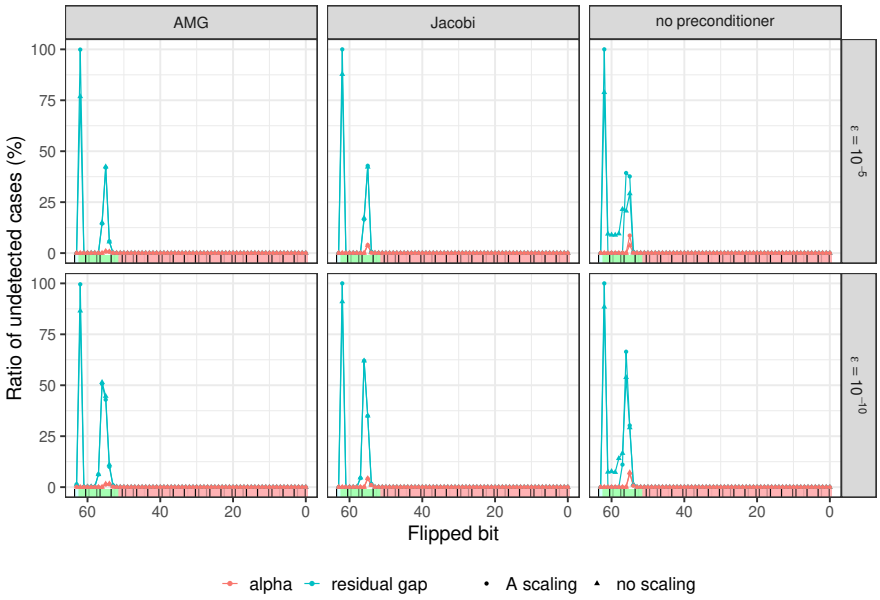


Figure 4.3: Detection performance of gap deviation and α-based methodologies for soft errors in the matrix-vector calculation with persistent faults in p_i for the matrix-vector product.

where we see that there is a switch in the behavior of PCG between transient bit-flips in the matrix-vector product and persistent bit-flips preconditioner, and between transient bit-flips in the preconditioner and persistent bit-flips in the matrix vector-product (see figures 4.4 and 4.5 for transient errors in [1]).

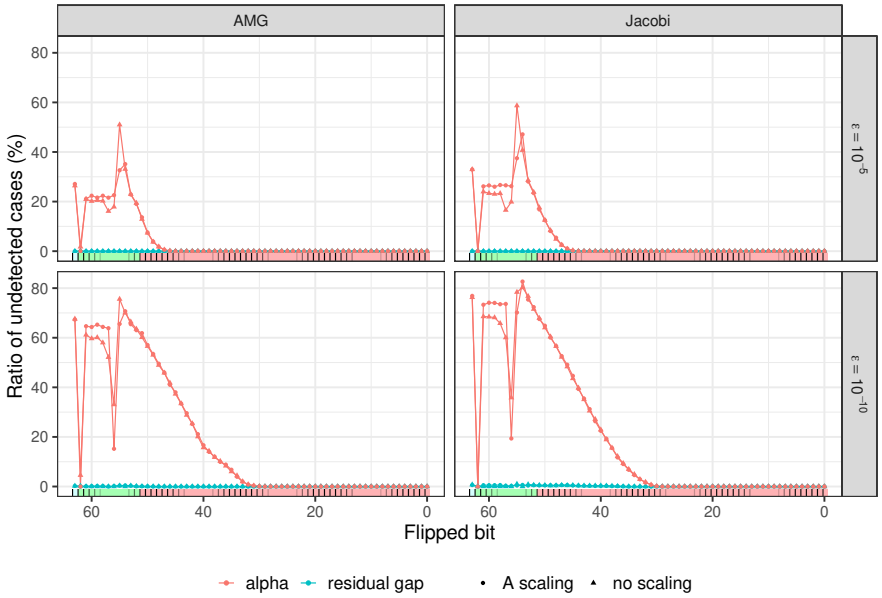


Figure 4.4: Detection performance of the residual gap and α-based methodologies for soft errors in the preconditioner calculation with persistent faults in r_{i+1} in the preconditioner calculation.

4.4 Combined detection

In Figure 4.5 we show what happens when we combine the α and the gap-based detection methods to detect persistent bit-flips in the matrix-vector product and preconditioning steps of PCG. Figure 4.6 expands on these results by looking at what happens when persistent bit-flips can occur in every step of PCG. As was the case for transient bit-flips (see Figure 4.6 in [1]), we see that both criteria are very complementary and can successfully be combined to detect almost all critical faults.

5 Conclusions

In this follow up study to [1] we studied the behavior of PCG when it is affected by persistent bit-flips in the matrix-vector product or preconditioner. We observed that there is a switch in this behavior between the matrix-vector product and the preconditioner when we compare it to transient bit-flips.

However, since the 2 detection criteria, i.e., the residual gap-based and the α-based criteria, where complementary when it came to detecting these transient bit-flips, they still are able to detect almost all critical faults caused by persistent bit-flips.

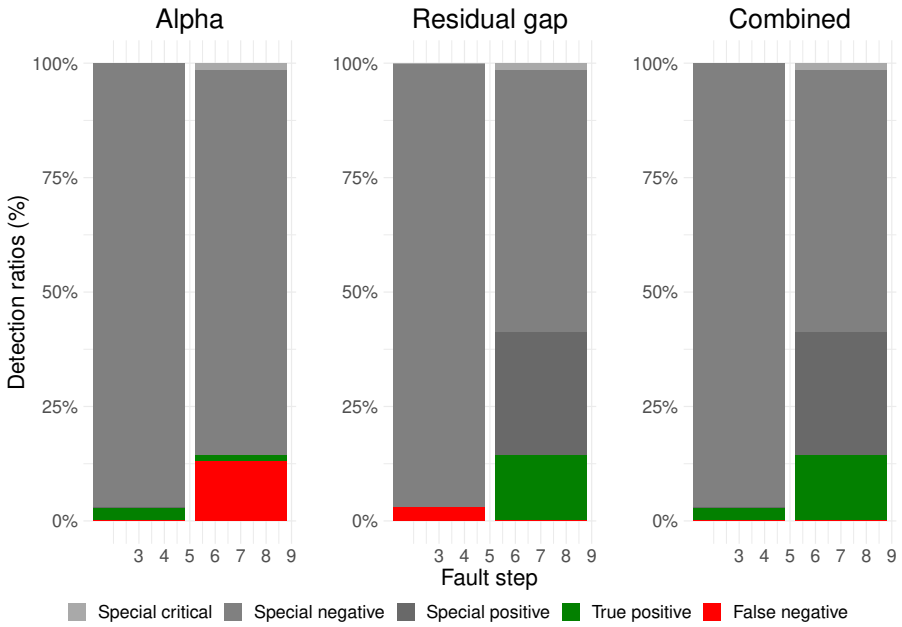


Figure 4.5: Comparison of the detection success of the alpha and residual gap based methodologies, and their combination for persistent bit-flips in the matrix-vector product or preconditioner (p_i or r_{i+1}.) steps of PCG.

References

[1] Emmanuel Agullo, Siegfried Cools, Emrullah Fatih-Yetkin, Luc Giraud, Nick Schenkel, and Wim Vanroose. On soft errors in the Conjugate Gradient method: sensitivity and robust numerical detection - revised. Research Report RR-9330, Inria Bordeaux Sud-Ouest, November 2020.

[2] 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.

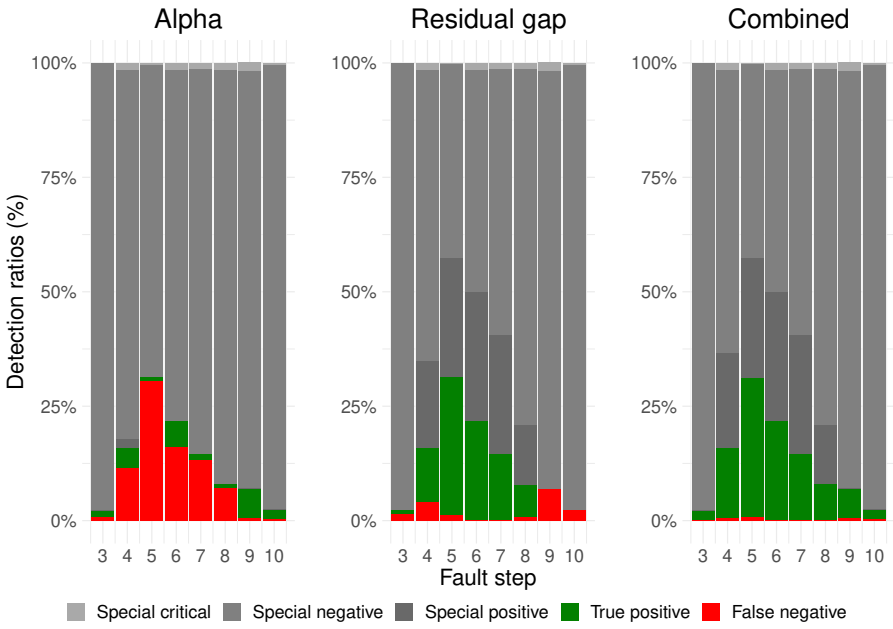


Figure 4.6: Comparison of the detection success of the alpha and residual gap based methodologies, and their combination for persistent bit-flips in every step of PCG.

A Checksum cost function

We choose τ by minimizing the cost function

$$\text{cost}(\tau) = \omega \frac{n_1}{n_1 + n_2} \text{FP}(\tau) + (1 - \omega) \frac{n_2}{n_1 + n_2} \text{FN}(\tau), \tag{3}$$

for $\omega, \tau \in [0, 1]$. If more than one value of τ minimizes this function, we take the largest value in order minimize false positive cases (FP). Note that in our numerical experiments we always used $\omega = 0.5$. In figures 1.1, 1.2, and 1.3 we show the cost function for different combinations of ε, preconditioner, and ω. These correspond to figures B.1, B.2, and B.3 in Appendix B from the original study [1]. We see that the cost function behaves very differently in the case of persistent errors. In the case of transient faults the cost function always had a convex shape, leading to an optimal value of $\tau < 1$. In the case of persistent faults, the cost function decreases, stabilizes, but does not increase again – at least for most of the matrices in our study. This results in an optimal value of $\tau = 1$ in almost all cases.

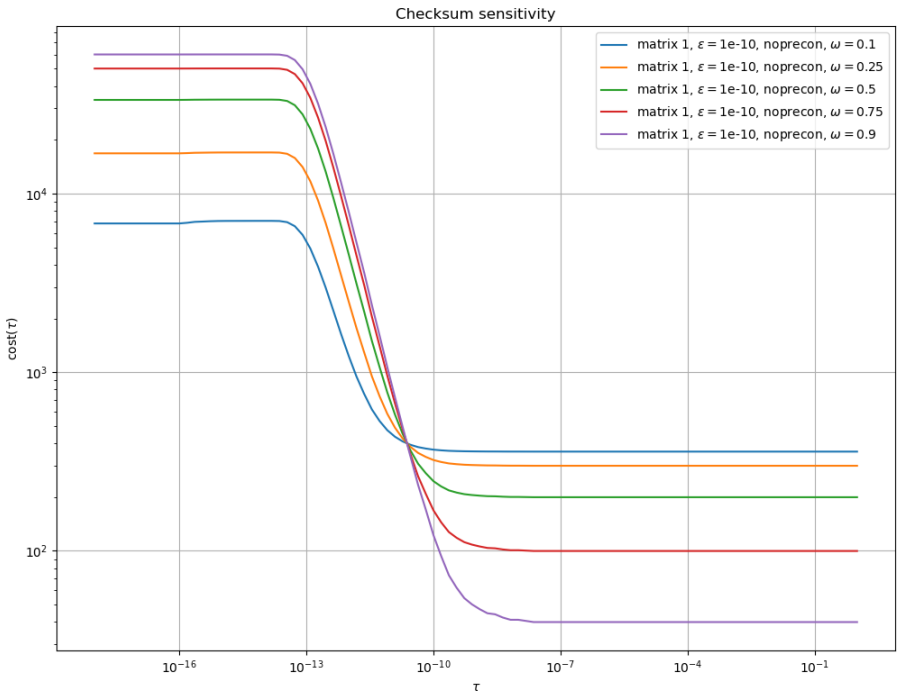


Figure 1.1: Cost function for a given matrix, ε, and preconditioner, but with different values of ω.

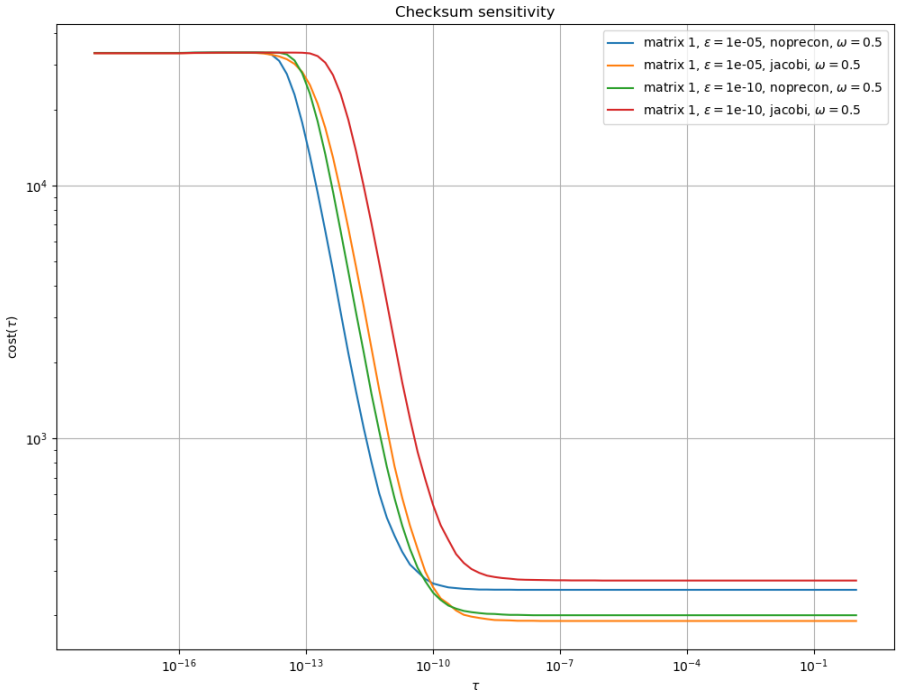


Figure 1.2: Cost function for a given matrix, and ω, but with different combinations of ε and preconditioner.

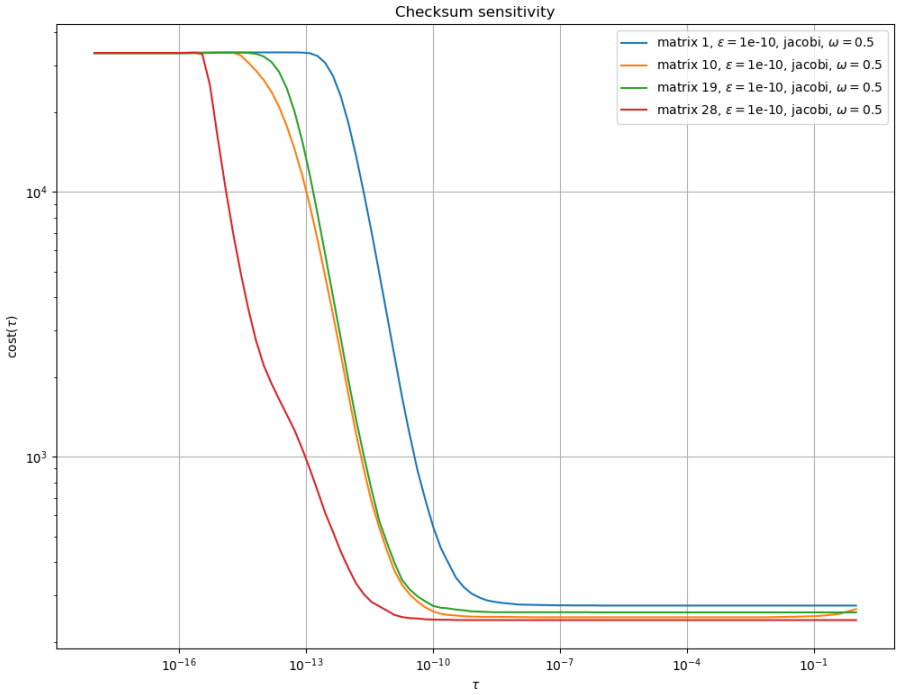
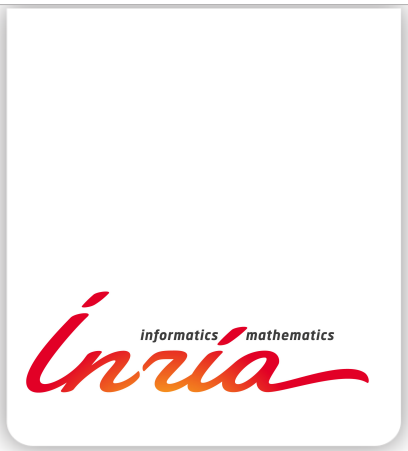


Figure 1.3: Cost function for a given ε, preconditioner, and ω, but for different matrices.



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