Towards resilient parallel linear Krylov solvers: recover-restart strategies

Emmanuel Agullo, Luc Giraud, Abdou Guermouche, Jean Roman, Mawussi Zounon

Towards resilient parallel linear Krylov solvers: recover-restart strategies

Emmanuel Agullo, Luc Giraud, Abdou Guermouche, Jean Roman, Mawussi Zounon

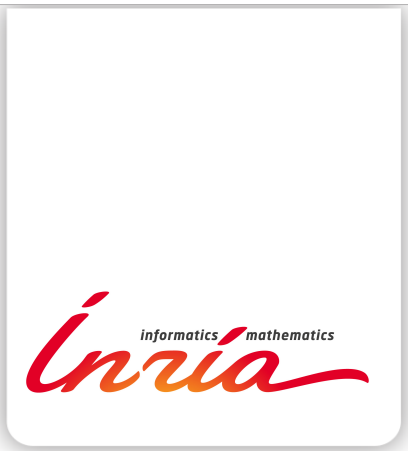
RESEARCH REPORT

N° 8324

June 2013

Project-Teams HiePACS

ISSN 0249-6399 ISRN INRIA/RR--8324--FR+ENG



Towards resilient parallel linear Krylov solvers: recover-restart strategies

Emmanuel Agullo*, Luc Giraud*, Abdou Guermouche†,
Jean Roman*, Mawussi Zounon*

Project-Teams HiePACS

Research Report n° 8324 — June 2013 — 36 pages

Abstract: The advent of extreme scale machines will require the use of parallel resources at an unprecedented scale, probably leading to a high rate of hardware faults. High Performance Computing (HPC) applications that aim at exploiting all these resources will thus need to be resilient, *i.e.*, be able to compute a correct solution in presence of faults. In this work, we investigate possible remedies in the framework of the solution of large sparse linear systems that is the inner most numerical kernel in many scientific and engineering applications and also one of the most time consuming part. More precisely, we present recovery followed by restarting strategies in the framework of Krylov subspace solvers where lost entries of the iterate are interpolated to define a new initial guess before restarting the Krylov method. In particular, we consider two interpolation policies that preserve key numerical properties of well-known solvers, namely the monotony decrease of the A-norm of the error of the conjugate gradient (CG) or the residual norm decrease of GMRES. We assess the impact of the recovery method, the fault rate and the number of processors on the robustness of the resulting linear solvers. We consider experiments with CG, GMRES and Bi-CGStab.

Key-words: Resilience, linear Krylov solvers, linear and least-square interpolation, monotonic convergence.

* Inria Bordeaux-Sud Ouest, France
† Université de Bordeaux 1, France

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vielle Tour
33405 Talence Cedex

Vers des solveurs linéaires de Krylov parallèles résilients

Résumé : Les machines exaflops annoncées pour la fin de la décennie seront très probablement sujettes à des taux de panne très élevés. Dans ce rapport nous présentons des techniques d'interpolation pour recouvrer des erreurs matérielles dans le contexte des solveurs linéaires de type Krylov. Pour chacune des techniques proposées nous démontrons qu'elles permettent de garantir des propriétés de décroissance monotone de la norme des résidus ou de la norme-A de l'erreur pour des méthodes telles que le gradient conjugué ou GMRES. A travers de nombreuses expérimentations numériques nous étudions qualitativement le comportement des différentes variantes lorsque le nombre de cœurs de calcul et le taux de panne varie.

Mots-clés : Résilience, solveurs de Krylov linéaires, interpolation linéaire ou de moindres carrés, convergence monotone.

Contents

1 Introduction

The current challenge in high performance computing (HPC) is to increase the level of computational power, by using the largest number of resources. This use of parallel resources at large scale leads to a significant decrease of the mean time between faults (MTBF) of HPC systems. Faults may be classified in *soft* and *hard* faults, according to their impact on the system. A soft fault is an inconsistency, usually not persistent and that does not lead *directly* to routine interruption. Typical soft faults are: bit flip, data corruption, invalid address values that still point to valid user data space [6]. A hard fault is a fault that causes immediate routine interruption. For example operating system crashes, memory crashes, unexpected processor unplugs are hard faults. In this work, we focus on hard faults. To deal with the permanent decrease of MTBF, HPC applications have to be resilient, *i.e.*, be able to compute a correct output despite the presence of faults.

In many large scale simulations, the most computational intensive kernel is often the iterative solution of very large sparse systems of linear equations. The development of resilient numerical methods and robust algorithms for the solution of large sparse systems of equations that still converge in presence of multiple and frequent faults is thus essential. Many studies focus on soft faults. For example, in [7] it is shown that iterative methods are vulnerable to soft faults, by exhibiting silent data corruptions and the poor ability to detect them. An error correction code based scheme is proposed in [24] to reduce linear solver soft fault vulnerability in the $L1$ and $L2$ cache. Fault detection and correction are efficient, because there is no need to restart the application. However data corruption is often silent and difficult to detect. To address soft faults, [12, 6] have developed fault-tolerant techniques based on the protection of a well chosen subset of data against soft fault. This model of fault tolerance allows programmers to demand reliability as needed for critical data and fault-susceptible programs. The selective reliability scheme aims at proposing specific annotations to declare the reliability of data [12].

To deal with hard faults, the most popular approaches are based on variants of checkpoint and restart techniques [8, 9, 10, 14, 22, 23]. The common checkpoint scheme consists in periodically saving data to a device such as a remote disk. When a fault occurs, all or selected processes are rolled back to the point of the most recent checkpoint, and their data are restored from the data saved. Application-level checkpointing schemes are also provided for the current main two parallel programming tools that are OpenMP [10] and MPI [19]. The checkpoint and restart approach is robust but may not scale well in certain cases [11]. The additional usage of resources (such as memory, disk) that is required by checkpoint and restart schemes may be prohibitive; or, the time to restore data might become larger than the MTBF [11].

Algorithm-Based Fault Tolerance (ABFT) techniques address soft and hard fault tolerance issues at an algorithm level. ABFT schemes have been designed to detect and correct faults in matrix computation [21]. Different ABFT schemes are discussed in [1, 3, 4, 13, 15, 20, 26]. Though ABFT schemes are disk-less, they may induce significant computational overhead. In [13, 18, 22] is proposed an ABFT scheme for iterative methods, named *lossy approach*, which consists of recomputing the entries of the lost data and exploiting all the possible redundancies of a parallel linear solver implementation. With the *lossy approach*, neither checkpoint nor checksum is necessary for the recovery. If no fault occurs during an execution, the fault-tolerance overhead of the *lossy approach* is zero.

In this work, we focus on fault-tolerance schemes that do not induce overhead when no fault occurs and do not assume any structure in the linear system nor data redundancy in the parallel solver implementation. We extend the recover-restart strategy introduced in [22]. In particular, we propose a recovery approach based on linear least squares properties and we generalize the techniques to the situations of multiple concurrent faults. We also show that the

proposed recover-restart schemes preserve key monotony properties of CG and GMRES. Except Equation (2), which comes from [22] and serves as a basis for the present work, all the theoretical results and numerical experiments presented in this manuscript are original to the best of our knowledge.

The paper is organized as follows. In Section 2, we present various recovery techniques and describe different variants to handle multiple faults. We present numerical experiments in Section 4 where the fault rate is varied to study the robustness of the proposed techniques. Some conclusions and perspectives are discussed in Section 5.

2 Strategies for fault recovery

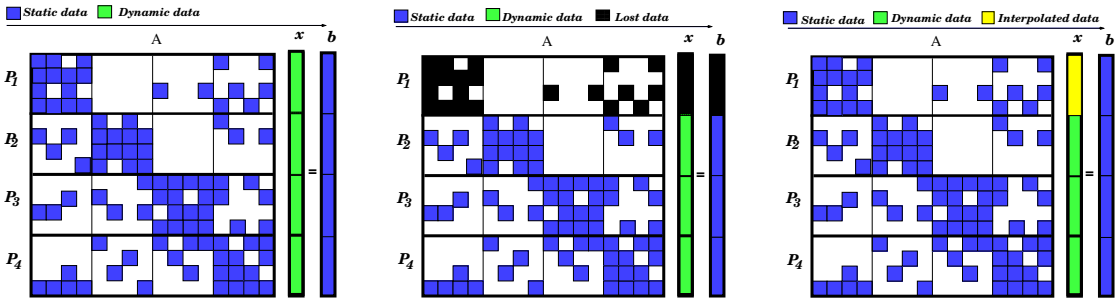
2.1 Context

In this paper, we consider the solution of sparse linear systems of equation of the form:

$$Ax = b \tag{1}$$

where the matrix $A \in \mathbb{R}^{n \times n}$ is non singular, the right-hand side $b \in \mathbb{R}^n$ and the solution $x \in \mathbb{R}^n$. We denote $a_{i,j}$ the entry of A on row i, column j. More precisely, we focus on iterative schemes based on parallel Krylov subspace methods. In a parallel distributed environment, Krylov subspace solvers are commonly parallelized thanks to a block-row partition of the sparse linear system (1). Let p be the number of partitions, such that each block-row is mapped to a processor. For all i, $i \in [1, p]$, I_i denotes the set of rows mapped to processor i. With respect to this notation, processor i stores the block-row $A_{I_i,:}$ and x_{I_i} as well as the entries of all the vectors involved in the Krylov solver associated with the corresponding row index of this block-row. If the block A_{I_i,I_j} contains at least one non zero entry, processor j is referred to as neighbor of processor i as communication will occur between those two processors to perform a parallel matrix-vector product. By $J_i = \{\ell, a_{\ell,I_i} \neq 0\}$, we denote the set of row indices in the block-column $A_{:,I_i}$ that contain non zero entries and $|J_i|$ denotes the cardinality of this set.

When a fault occurs on a processor, all available data in its memory are lost. We consider the formalism proposed in [22] where lost data are classified into three categories: the *computational environment*, the *static* data and the *dynamic* data. The computational environment is all the data needed to perform the computation (code of the program, environment variables, ...). The static data are those that are setup during the initialization phase and that remain unchanged during the computation. The coefficient matrix A, the right-hand side vector b are static data. Dynamic data are all data whose value may change during the computation. The Krylov basis vectors (e.g., Arnoldi basis, descent directions, residual, ...) and the iterate are examples of dynamic data. In Figure 1a, we depict a block row distribution on four processors. The data in blue is the static data associated with the linear system (i.e., matrix and right-hand side) while the data in green is the dynamic data (here only the iterate is shown). If processor P_1 fails, the first block row of A as well as the first entries of x and b are lost (in black in Figure 1b). We assume that when a fault occurs, the failed processor is replaced immediately and the associated computational environment and static data are restored. In Figure 1c for instance, the first matrix block row as well as the corresponding righ-hand side are restored as they are static data. However the iterate, being a dynamic data, is definitely lost and we discuss in the following strategies for recovering it. Indeed, for the sake of genericity among Krylov solvers, our strategies do not attempt to recover all the dynamic data but only the iterate. More precisely we investigate recovery techniques that interpolate the lost entries of the iterates using interpolation strategies that make sense for the linear systems to be solved. The interpolated



(a) Before Fault (b) Faulty iteration (c) Recovery

Figure 1: General recovery scheme. The matrix is initially distributed with a block row partition, here on four processors (a). When a fault occurs on processor P_1, the corresponding data is lost (b). Whereas static data can be immediately restored, dynamic data that has been lost cannot and we investigate numerical strategies for recovering it (c).

entries and the current values available on the other processors are used as a new initial guess to restart the Krylov iterations.

We assume in the rest of Section 2 that a fault occurs during iteration k+1 and the proposed recoveries are thus based on the value of the iterate at iteration k. We furthermore first make the assumption that only one processor can fail at a time in sections 2.2 and 2.3 and relax that assumption in Section 2.4 for studying the multiple fault case.

2.2 Linear interpolation

The linear interpolation, first introduced in [22] and denoted LI in the sequel, consists in interpolating lost data by using data from non-failed processors. Let $x^{(k)}$ be the approximate solution when a fault occurs. After the fault, the entries of $x^{(k)}$ are known on all processors except the failed one. The LI strategy computes a new approximate solution by solving a local linear system associated with the failed processor. If processor i fails, $x^{(LI)}$ is computed via

$$\begin{cases} x_{I_j}^{(LI)} = x_{I_j}^{(k)} & \text{for } j \neq i, \\ x_{I_i}^{(LI)} = A_{I_i,I_i}^{-1}(b_{I_i} - \sum_{j \neq i} A_{I_i,I_j} x_{I_j}^{(k)}). \end{cases} \tag{2}$$

The motivation of for this interpolation strategy is that, at convergence ($x^{(k)} = x$), it reconstitutes the exact same solution ($x^{(LI)} = x$) as long as A_{I_i,I_i} is non singular. We now furthermore show that such an interpolation exhibits a property in term of A-norm of the error for symmetric positive definite matrices as expressed in the proposition below.

Proposition 1 *Let A be symmetric positive definite (SPD). Let $k+1$ be the iteration during which the fault occurs on processor i. The recovered entries $x_{I_i}^{(LI)}$ defined by Equation (2) are always uniquely defined. Furthermore, let $e^{(k)} = x - x^{(k)}$ denote the forward error associated with the iterate before the fault occurs, and $e^{(LI)} = x - x^{(LI)}$ be the forward error associated with the new initial guess recovered using the LI strategy (2), we have:*

$$\|e^{(LI)}\|_A \leq \|e^{(k)}\|_A.$$

Proof 1 *1. Uniquely defined $x_{I_i}^{(LI)}$: because A is SPD so is A_{I_i,I_i} that is consequently non singular.*

2. Monotonic decrease of $\|e^{(LI)}\|_A$: for the sake of simplicity of exposure, but without any loss of generality we consider a two processor case and assume that the first processor fails. Let $A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}$ be a SPD matrix, where $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ denotes the exact solution of the linear solution. The equations associated with the exact solution are:

$$A_{1,1}x_1 + A_{1,2}x_2 = b_1, \tag{3a}$$
$$A_{2,1}x_1 + A_{2,2}x_2 = b_2. \tag{3b}$$

By linear interpolation (Equation (2)), we furthermore have:

$$A_{1,1}x_1^{(LI)} + A_{1,2}x_2^{(k)} = b_1, \tag{4a}$$
$$x_2^{(LI)} = x_2^{(k)}. \tag{4b}$$

Given two vectors, y and z, we recall that:

$$y^T Az = y_1^T A_{1,1} z_1 + y_1^T A_{1,2} z_2 + y_2^T A_{2,1} z_1 + y_2^T A_{2,2} z_2, \tag{5}$$

$$\|y\|_A^2 = y_1^T A_{1,1} y_1 + y_2^T A_{2,2} y_2 + 2y_1^T A_{1,2} y_2, \tag{6}$$
$$\|y - z\|_A^2 = y^T Ay - 2y^T Az + z^T Az, \tag{7}$$
$$(y+z)^T A(y-z) = y^T Ay - z^T Az. \tag{8}$$

The proof consists in showing that $\delta = \|x^{(LI)} - x\|_A^2 - \|x^{(k)} - x\|_A^2$ is non positive. It is easy to see by (4b) and (7) that:

$$\delta = (x_1^{(LI)})^T \left(A_{1,1}x_1^{(LI)} + 2A_{1,2}x_2^{(k)} \right) - (x_1^{(k)})^T \left(A_{1,1}x_1^{(k)} + 2A_{1,2}x_2^{(k)} \right) + 2\left((x_1^{(k)})^T A_{1,1}x_1 + (x_1^{(k)})^T A_{1,2}x_2 - (x_1^{(LI)})^T A_{1,1}x_1 - (x_1^{(LI)})^T A_{1,2}x_2 \right).$$

By (3a) and (8), we have:

$$\delta = \left(x_1^{(LI)} - x_1^{(k)} \right)^T A_{1,1} \left(x_1^{(LI)} + x_1^{(k)} \right) + 2\left(x_1^{(LI)} - x_1^{(k)} \right)^T \left(A_{1,2}x_2^{(k)} - b_1 \right)$$
$$= \left(x_1^{(LI)} - x_1^{(k)} \right)^T \left(A_{1,1}x_1^{(LI)} + A_{1,2}x_2^{(k)} - 2b_1 + A_{1,1}x_1^{(k)} + A_{1,2}x_2^{(k)} \right)$$

Because A is SPD, so is $A_{1,1}$ and $A_{1,1}^T A_{1,1}^{-1} = I$. Then by (4a), we have,

$$\begin{aligned}
\delta &= \left(x_1^{(LI)} - x_1^{(k)} \right)^T A_{1,1}^T A_{1,1}^{-1} \left(-b_1 + A_{1,1}x_1^{(k)} + A_{1,2}x_2^{(k)} \right) \\
&= -\left((A_{1,1}x_1^{(LI)}) - (A_{1,1}x_1^{(k)}) \right)^T A_{1,1}^{-1} \left(b_1 - A_{1,1}x_1^{(k)} - A_{1,2}x_2^{(k)} \right), \\
&= \left(b_1 - A_{1,1}x_1^{(k)} - A_{1,2}x_2^{(k)} \right)^T A_{1,1}^{-1} \left(b_1 - A_{1,1}x_1^{(k)} - A_{1,2}x_2^{(k)} \right), \\
&= -\|b_1 - A_{1,1}x_1^{(k)} - A_{1,2}x_2^{(k)}\|_{A_{1,1}^{-1}}^2 \\
&\leq 0.
\end{aligned}$$

Note that the proof also gives us a quantitative information on the decrease:

$$\delta = \|x^{(LI)} - x\|_A^2 - \|x^{(k)} - x\|_A^2 = -\|b_1 - A_{1,1}x_1^{(k)} - A_{1,2}x_2^{(k)}\|_{A_{1,1}^{-1}}^2.$$

Finally, in the general case, it can be noticed that the LI strategy is only defined if the diagonal block A_{I_i,I_i} has full rank. In the next section, we propose an interpolation variant that will enable more flexibility in the case of multiple faults and does not make any rank assumption.

2.3 Least squares interpolation

The LI strategy is based on the solution of a local linear system. The new variant we propose relies on a least squares solution and is denoted LSI in the sequel. A new variant that relies on a least squares solution can also be defined that is denoted LSI in the sequel. Assuming that processor i has failed, x_{I_i} is interpolated as follows:

$$\begin{cases} x_{I_j}^{(LSI)} = x_{I_j}^{(k)} & \text{for } j \neq i, \\ x_{I_i}^{(LSI)} = \underset{x_{I_i}}{\operatorname{argmin}} \|(b - \sum_{j \neq i} A_{:,I_j} x_j^{(k)}) - A_{:,I_i} x_{I_i}\|. \end{cases} \tag{9}$$

We notice that the matrix involved in the least squares problem, $A_{:,I_i}$, is sparse of dimension $|J_i| \times |I_i|$ where its number of rows $|J_i|$ depends on the sparsity structure of $A_{:,I_i}$. Consequently the LSI strategy has a higher computational cost, but it overcomes the rank deficiency drawback of LI because the least squares matrix is always full column rank (as A is full rank).

Proposition 2 *Let $k+1$ be the iteration during which the fault occurs on processor i. The recovered entries of $x_{I_i}^{(LSI)}$ defined in Equation (9) are uniquely defined. Furthermore, let $r^{(k)} = b - Ax^{(k)}$ denote the residual associated with the iterate before the fault occurs, and $r^{(LSI)} = b - Ax^{(LSI)}$ be the residual associated with the initial guess generated with the LSI strategy* (9)*, we have:*

$$\|r^{(LSI)}\|_2 \leq \|r^{(k)}\|_2.$$

Proof 2

1. *Uniquely defined: because A is non singular, $A_{:,I_i}$ has full column rank.*
2. *Monotonic residual norm decrease: the proof is a straightforward consequence of the definition of $x_{I_i}^{(LSI)} = \underset{x_{I_i}}{\operatorname{argmin}} \|(b - \sum_{j \neq i} A_{:,I_j} x_j^{(k)}) - A_{:,I_i} x_{I_i}\|$*

Remark 1 *Notice that the LSI recover-restart technique is exact in the sense that if the fault occurs at the iteration where the stopping criterion based on a scaled residual norm is detected, this recovery will regenerate an initial guess that also complies with the stopping criterion.*

2.4 Multiple faults

So far, we have introduced two policies to handle a single fault occurrence; but multiple processors may fail during the same iteration especially when a huge number of processors will be used. At the granularity of our approach, these faults may be considered as simultaneous. To our knowledge, the multiple fault situation has not been addressed by other authors. We present here two strategies to deal with such multiple faults in the context of both the LI and LSI approaches.

2.4.1 Global recovery techniques

The approach described in this section consists in recovering multiple faults all at once. With this global recovery technique, the linear system is permuted so that the equations relative to the failed processors are grouped into one block. Therefore the recovery technique falls back to the single fault case. For example, if processors i and j fail, the global linear interpolation (LI-G) solves the following linear system (similar to Equation (2))

$$\begin{pmatrix} A_{I_i,I_i} & A_{I_j,I_i} \\ A_{I_j,I_i} & A_{I_j,I_j} \end{pmatrix} \begin{pmatrix} x_{I_i}^{(LI-G)} \\ x_{I_j}^{(LI-G)} \end{pmatrix} = \begin{pmatrix} b_{I_i} - \sum_{\ell \notin \{i,j\}} A_{I_i,I_\ell} x_{I_\ell}^{(k)} \\ b_{I_j} - \sum_{\ell \notin \{i,j\}} A_{I_j,I_\ell} x_{I_\ell}^{(k)} \end{pmatrix}.$$

Following the same idea, the global least squares interpolation (LSI-G) solves

$$\begin{pmatrix} x_{I_i}^{(LSI-G)} \\ x_{I_j}^{(LSI-G)} \end{pmatrix} = \underset{x_{I_i}, x_{I_j}}{\operatorname{argmin}} \| (b - \sum_{\ell \notin \{i,j\}} A_{:,I_\ell} x_\ell^{(k)}) - A_{(:,I_i \cup I_j)} \begin{pmatrix} x_{I_i} \\ x_{I_j} \end{pmatrix} \|.$$

2.4.2 Local recovery techniques

Alternatively, if processors i and j fail simultaneously, x_{I_i} and x_{I_j} can be interpolated independently from each other. Using the LI strategy, the entries of x_{I_i} can be computed using Equation (2) assuming that the quantity x_{I_j} is equal to its initial value $x_{I_j}^{(0)}$. At the same time processor j recover x_{I_j} assuming that $x_{I_i} = x_{I_i}^{(0)}$. We call this approach uncorrelated linear interpolation (LI-U). For example we recover x_{I_i} via

1: $x_{I_j}^{(k)} = x_{I_j}^{(0)}$,
2: $x_{I_\ell}^{(LI-U)} = x_{I_\ell}^{(k)}$ for $\ell \notin \{i,j\}$,
3: $x_{I_i}^{(LI-U)} = A_{I_i,I_i}^{-1}(b_{I_i} - \sum_{i \neq \ell} A_{I_i,I_\ell} x_{I_\ell}^{(k)})$.

Although better suited for a parallel implementation, this approach might suffer from a worse interpolation quality when the off-diagonal blocks A_{I_i,I_j} or A_{I_j,I_i} are non zero (it of course reduces to LI if both extra diagonal blocks are zero, i.e., processor i and j are not neighbor). Similar idea can be applied to LSI to implement an uncorrelated LSI (LSI-U). However, the flexibility of LSI can be further exploited to reduce the potential bad effect of considering $x_{I_j}^{(0)}$ when recovering x_{I_i}. Basically, to recover x_{I_i}, each equation that involves x_{I_j} is discarded from the least squares system and we solve the following equation

$$x_{I_i}^{(LSI-U)} = \underset{x_{I_i}}{\operatorname{argmin}} \| (b_{J_i \setminus J_j} - \sum_{\ell \notin \{i,j\}} A_{J_i \setminus J_j, I_\ell} x_{I_\ell}^{(k)}) - A_{J_i \setminus J_j, I_i} x_{I_i} \|, \tag{10}$$

where the set of row-column indices $(J_i \setminus J_j, I_\ell)$ denotes the set of rows of block column I_ℓ of A that have non zero entries in row J_i and zero entries in row J_j (if the set $(J_i \setminus J_j, I_\ell) = \emptyset$ then $A_{J_i \setminus J_j, I_\ell}$ is a zero matrix).

We denote this approach by decorrelated LSI (LSI-D). The heuristic beyond this approach is to avoid perturbing the recovery of x_{I_i} with entries in the right-hand sides that depends on x_{I_j} that are unknown. A possible drawback is that discarding rows in the least squares problem might lead to an under-determined or to a rank deficient problem. In such a situation, the minimum norm solution might be meaningless with respect to the original linear system. Consequently the computed initial guess to restart the Krylov method might be poor and could slow down the overall convergence.

3 Recovery for Krylov solvers

In this section we briefly describe the main two subspace Krylov techniques that we consider. We recall their main numerical/computational properties and discuss how they are affected by the recovery techniques introduced in the previous sections.

3.1 The conjugate gradient method

The conjugate gradient method (CG) is the method of choice for the solution of linear systems involving SPD matrices. It can be expressed via short term recurrences with a recurrence for the iterate as depicted in Algorithm 1.

Algorithm 1 Conjugate gradient (CG)

1: Compute $r_0 = b - Ax^{(0)}$,
2: $p_0 = r_0$
3: **for** $j = 0, 1, \ldots,$ until convergence, **do**
4: $\quad \alpha_j = r_j^T r_j / p_j^T A p_j$
5: $\quad x^{(j+1)} = x^{(j)} + \alpha_j p_j$
6: $\quad r_{j+1} = r_j - \alpha_j A p_j$
7: $\quad \beta_j = \frac{r_{j+1}^T r_{j+1}}{r_j^T r_j}$
8: $\quad p_{j+1} = r_{j+1} + \beta_j p_j$
9: **end for**

The CG algorithm enjoys the unique property to minimize the A-norm of the forward error on the Krylov subspaces, i.e., $\|x^{(k)} - x\|_A$ is monotonically decreasing along the iterations k (see for instance [27]). This decreasing property is still valid for the preconditioned conjugate gradient (PCG) method. Consequently, an immediate consequence of Proposition 1 reads:

Corollary 1 *The initial guess generated by either LI or LI-G after a single or a multiple failure does ensure that the A-norm of the forward error associated with the recover-restart strategy is monotonically decreasing for CG and PCG.*

3.2 GMRES

The GMRES method is one of the most popular solver for the solution of unsymmetric linear systems. It belongs to the class of Krylov solvers that minimize the 2-norm of the residual associated with the iterates built in the sequence of Krylov subspaces (MinRES is another example of such a solver [25]). In contrast to many other Krylov methods, GMRES does not update the iterate at each iteration but only either when it has converged or when it restarts every other m steps (see Algorithm 2, lines 14-16) in the so-called restarted GMRES (GMRES(m)). When a fault occurs, the approximate solution is not available. However, in most of the classical parallel GMRES implementations, the Hessenberg matrix $\bar{H}_m$ is replicated on each processor and the least squares problem is also solved redundantly. Consequently, each individual still running processor ℓ can compute its entries I_ℓ of the iterate when a failure occurs.

The property of residual norm monotony of GMRES and GMRES(m) is still valid in case of failure for the recover-restart strategies LSI (for single fault) and LSI-G (even for multiple faults).

Algorithm 2 GMRES

1: Set the initial guess x^0;
2: **for** $k = 0, 1, \ldots,$ until convergence, **do**
3: $r_0 = b - Ax^0$; $\beta = \|r_0\|$
4: $v_1 = r_0/\|r_0\|$;
5: **for** $j = 1, \ldots, m$ **do**
6: $w_j = Av_j$
7: **for** $i = 1$ `to` j **do**
8: $h_{i,j} = v_i^T w_j$; $w_j = w_j - h_{i,j} v_i$
9: **end for**
10: $h_{j+1,j} = \|w_j\|$
11: If $(h_{j+1,j}) = 0$; $m = j$; goto 14
12: $v_{j+1} = w_j / h_{j+1,j}$
13: **end for**
14: Define the $(m+1) \times m$ upper Hessenberg matrix $\bar{H}_m$
15: Solve the least squares problem $y_m = \arg\min \|\beta e_1 - \bar{H}_m y\|$
16: Set $x^0 = x^0 + V_m y_m$
17: **end for**

Corollary 2 *The recover-restart strategies LSI and LSI-G do ensure the monotonic decrease of the residual norm of minimal residual Krylov subspace methods such as GMRES, Flexible GMRES and MinRES after a restart due to a failure.*

We should point out that this corollary does not translate straightforwardly to preconditioned GMRES as it was the case for PCG in Corollary 1. For instance for left preconditioned GMRES, the minimal residual norm decrease applies to the linear system $MAx = Mb$ where M is the preconditioner. To ensure the monotonic decrease of the preconditioned residual, the least squares problem should involve matrices that are part of MA, which might be complicated to build depending on the preconditioner used. In that case, because GMRES computes iterates $x^{(k)}$ one might compute a recovery of x using only A but we loose the monotonicy property. For right preconditioned GMRES, $AMu = b$ with $x = Mu$ similar comments can be made, except for block diagonal preconditioner where the property holds. Indeed, similarly to the unpreconditioned case, in the block diagonal right preconditioner case, after a failure all the entries of u but those allocated on the failed processors can be computed, so can the corresponding entries of x (that are computed locally as the preconditioner is block diagonal); therefore, the new initial guess constructed by LSI or LSI-G still complies with Proposition 2. Finally, the possible difficulties associated with general preconditioners for GMRES disappear when Flexible GMRES is considered. In that latter case, the generalized Arnoldi relation $AZ_k = V_{k+1}\bar{H}_k$ holds (using the classical notation from [27]), so that the still alive processors can compute their part of x_k from their piece of Z_k.

4 Numerical experiments

In this section we investigate first the numerical behavior of the Krylov solvers restarted after a failure when the new initial guess is computed using the strategies discussed above. For the sake of simplicity of exposure, we organized this numerical experiment section as follows. We first present in Section 4.2 numerical experiments where at most one fault occurs during one iteration. In Section 4.3, we consider examples where multiple faults occur during some iterations

to illustrate the numerical robustness of the different variants we exposed in Section 2.4. For the sake of completeness and to illustrate the possible numerical penalty induced by the restarting procedure after the failures we compare in Section 4.4 the convergence behaviour of the different Krylov solvers with and without failure. For the recovery calculations, we use sparse direct solvers (Cholesky or LU) for the LI variants and QR factorization for the LSI variants. We investigate the additional computational cost associated with this "exact" recovery in Section 4.5.

4.1 Experimental framework

We have simulated a faulty parallel distributed platform in Matlab. In that respect, the matrix of the linear system to be solved is first reordered to minimize the number of off-diagonal entries associated with the block row partitioning. This reordering actually corresponds to the one we would have performed if we had run the experiments in parallel; it attempts to minimize the communication volume required by the parallel matrix-vector product.

For the fault injection, we generate fault dates independently on the p processors using the Weibull probability distribution that is admitted to provide realistic distribution of faults. Its probability density function is:

$$f(T;\lambda,k) = \begin{cases} \frac{k}{\lambda}(\frac{T}{\lambda})^{k-1}e^{-(\frac{T}{\lambda})^k} & \text{if } T \geq 0, \\ 0 & \text{if } T < 0, \end{cases} \tag{11}$$

where T is the operating time or age that can we express in floating point operations (Flop) in our experiments. The parameter k, $(k > 0)$ is the shape parameter, related to the variation of the fault rate. If $k < 1$, the fault rate decreases over time. The case $k = 1$ induces a constant fault rate and thus corresponds to an exponential distribution. Finally, $k > 1$ means that the fault rate increases over time. The parameter λ is the scale parameter, it can be viewed as a function of MTBF precisely in the case of an exponential distribution, $MTBF = \frac{1}{\lambda}$. For our simulations, we use $k \approx 0.7$ [5], the value of MTBF is a function of cost of iterations in terms of Flop. For example $MTBF = \alpha \times IterCost$ means that a fault is expected to occur every other α iterations.

We have performed extensive numerical experiments and only report on qualitative numerical behaviour observed on a few examples that are representative of our observations (more experiments are available in the appendix). Most of the matrices come from the University of Florida test suite. The right-hand sides are computed for a given solution generated randomly. Finally to ensure a reasonable convergence rate, we generally used a preconditioner.

To study the numerical features of the proposed recover-restart strategies, we display the convergence history as a function of the iterations. For the unsymmetric solver, we depict the scaled residual, while for the symmetric positive definite case (SPD) we depict the A-norm of the error. For the sake of comparison, we systematically display the convergence history of a cheap checkpoint strategy that consists in checkpointing only the iterate at each iteration. In that latter case, when a fault occurs we restart the Krylov method from the latest computed entries of the lost iterate. We refer to this strategy as Selective Checkpointing and denote it SC. We also depict in red (Reset) a straightforward strategy where the lost entries of the iterate are replaced by the corresponding ones of the first initial guess.

4.2 Numerical behavior in single fault cases

In this section we first examine the situation where only one fault occurs during an iteration. We present the convergence history for the LI and LSI recover-restart strategies in Figure 2-5.

The first overall observation is that the reset strategy does not work when many faults occur. After each fault, the convergence criterion moves up to a value close to the initial one and does not succeed to decrease enough before the next failure. The convergence history of this approach is very noisy with essentially a peak after each fault. The second global observation is that the other three (LI, LSI, SC) do enable us to get convergence even when a significant amount of failures occur.

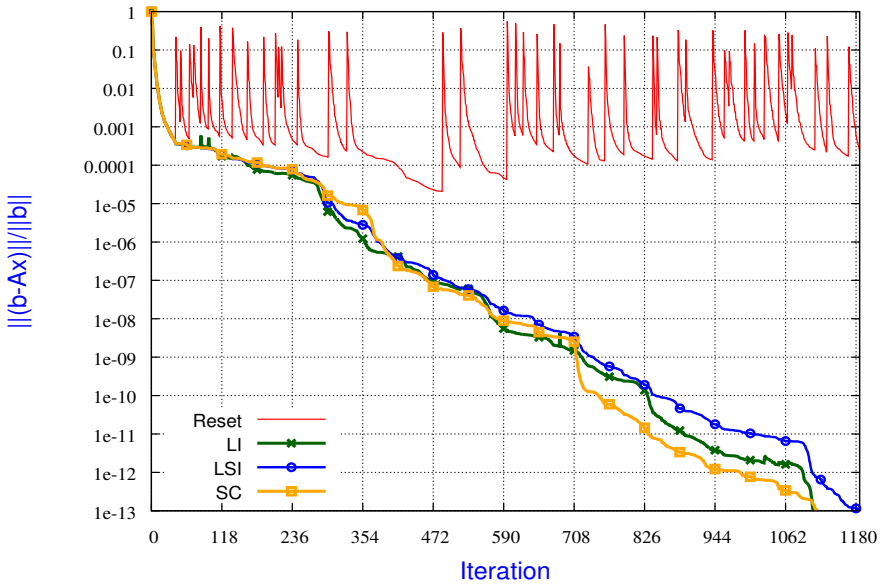


Figure 2: Right block diagonal preconditioned GMRES on UF Averous/epb0 using 16 processors with 44 single faults

For GMRES (CG), the SC curves are monotonically decreasing as they correspond to the convergence of GMRES with variable restart [2] (resp. A-norm minimization of the error with CG). For GMRES with right block-Jacobi preconditioner we can see in Figure 2 that the residual norm with LSI is monotonically decreasing as indicated by Corollary 2, while LI does exhibit a few (local) increases. When left preconditioner is used, because the recovery is computed based on A the monotony is no longer observed for LSI as shown in Figure 3.

In Figure 4, we display the A-norm of the error for the three recover-restart strategies. Although not visible on the curves, LSI does have a few small increases while LI does converge monotonically. For that example SC performs better than the other two, but we observed the reverse on other examples (some are available in the appendix).

As in many situations, BiCGStab exhibits a highly oscillatory convergence behaviour of the residual norm, this is also observed with our recover-restart strategies as it can be seen in Figure 5. Nevertheless, as for the other examples with GMRES and CG, the recover-restart strategies based on either of the two interpolation strategies have similar behaviour and comparable with a light checkpointing such as SC. From the extensive numerical experiments we have performed, none of the three recover-restart policies has shown to be the best nor the worse, even though on the graphs reported here SC is often slightly better than the others.

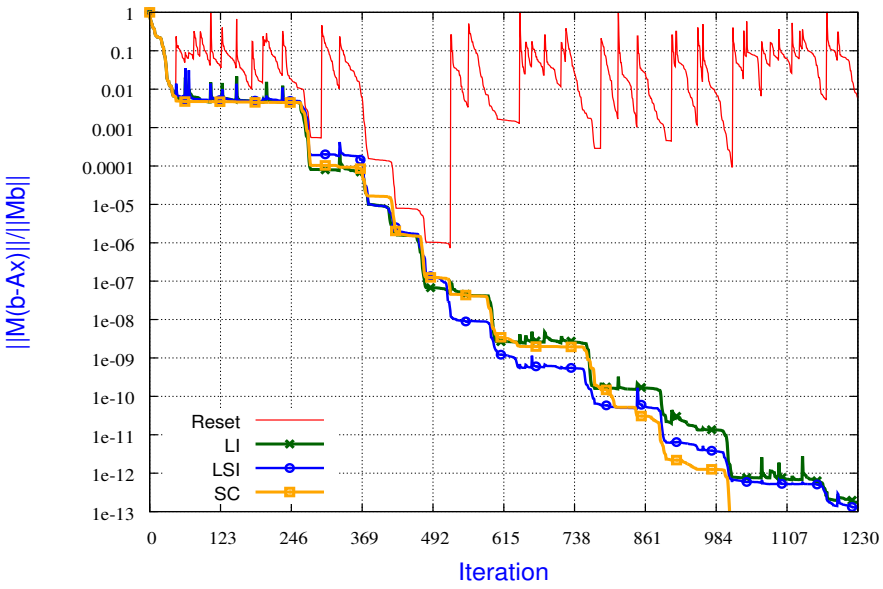


Figure 3: Left preconditioned GMRES on UF Averous/epb0 using 16 processors with 44 single faults

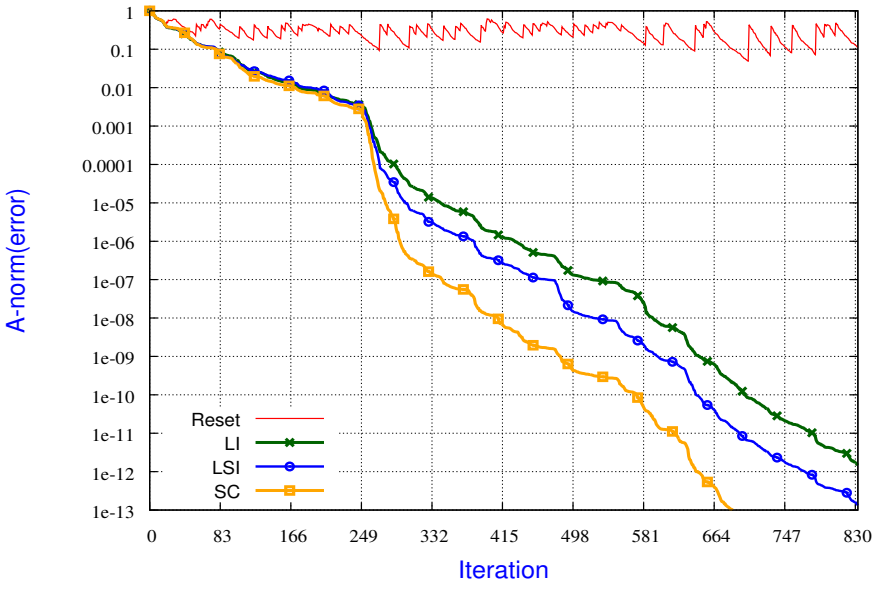


Figure 4: PCG on a 7-point stencil 3D Poisson equation using 16 processors with 70 single faults

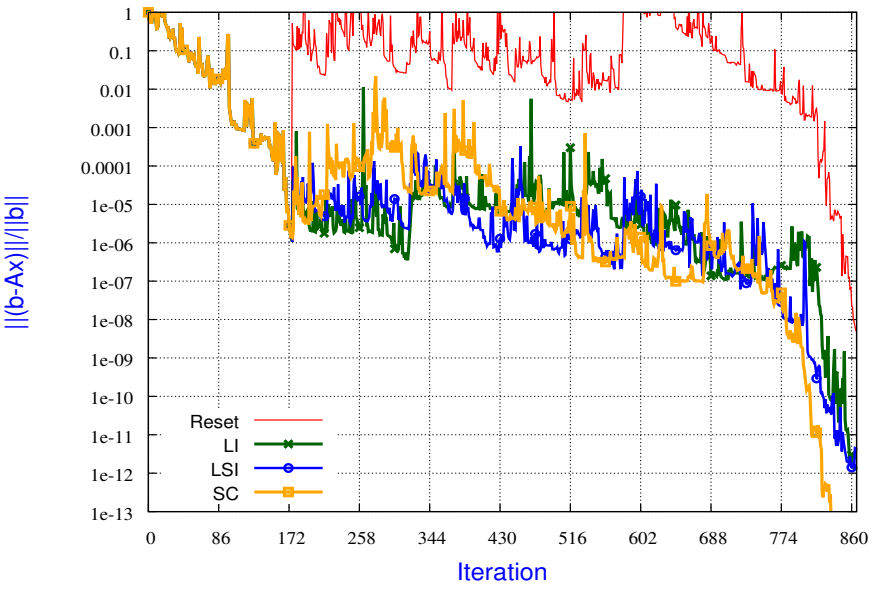


Figure 5: BiCGStab on UF Averous/epb0 using 16 processors with 15 single faults

4.3 Numerical behavior in multiple fault cases

In this section we illustrate the numerical behaviour of the various recover-restart strategies described in Section 2.4. We made a selection of a few numerical experiments that are reported in the Figures 6-9. What is referred to as a multiple fault corresponds to the situation where the entries of x_{I_i} and x_{I_j} are lost at the same iteration and either the block A_{I_i,I_j} or the block A_{I_j,I_i} is non zero (i.e., processors i and j are neighbor), consistently with Section 2.4. In that respect, among the faults that are considered as simple, some might still occur during the same iteration but since they are uncorrelated they only account for one single fault. Furthermore, to be able to observe a few multiple faults using our fault injection probability law we had to generate a large number of faults.

In Figure 6-9, the multiple fault occurrences are characterized by a significant jump of the residual norm for GMRES and of the A-norm of the error for PCG for the two recover-restart strategies LI-U and LSI-U; that are almost as bad as the straightforward reset approach. The underlying idea to design these heuristics was to interpolate lost entries by fully ignoring other simultaneous failures. Those experiments show that the penalty to pay is very high and that a special treatment deserves to be implemented.

The first possibility is to consider the LI-G or the LSI-G recover-restart policy, where all the lost entries are recovered at once as if a "large" single fault occurred. It can be seen in these figures that the numerical behaviour is consequently very similar to the ones we observed in the previous section where only single faults were considered. More interesting is the behaviour of the LSI-D strategy whose behaviour seems to vary a lot from one example to another. In Figures 7 and 9, this policy enables a convergence similar to the two robust strategies LI-G and LSI-G, while in Figures 6 and 8 large jumps are observed with this recover-restart strategy. Actually, this latter bad behaviour occurs when the least squares problem, that is solved once the correlated rows have been discarded, becomes rank deficient. In that case, the recovered initial guess is poor. In order to remove this drawback, one could switch to LI-G or LSI-G when a rank deficiency

in the least squares matrix is detected. Such an hybrid scheme would conciliate robustness and speed of the recover-restart approach and would thus certainly represent a strategy of choice for a production code but is out of the scope of this study and we do not consider it in the rest of the paper.

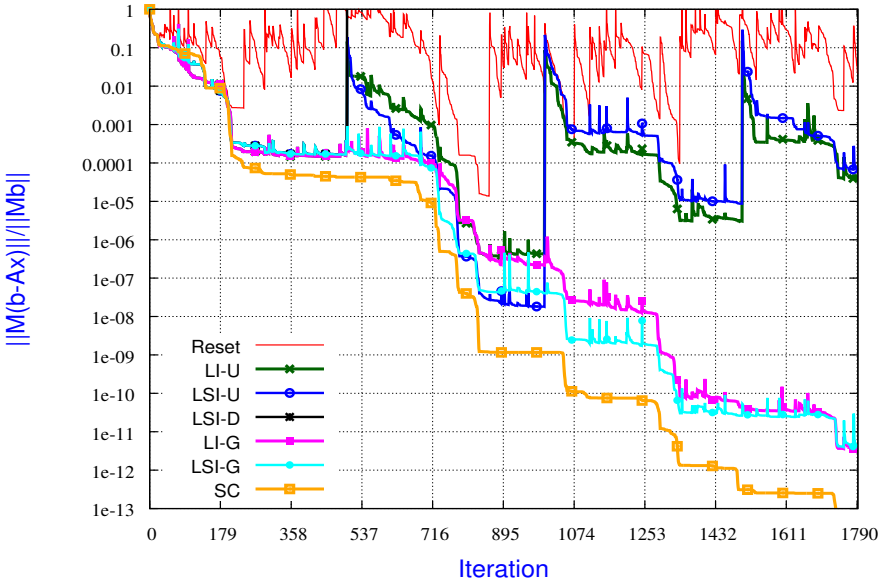


Figure 6: Left preconditioned GMRES on UF Averous/epb0 using 16 processors with 103 single fault and 3 multiple faults

4.4 Penalty of the recover-restart strategy on convergence

One of the main feature of the resilient numerical scheme of the algorithms described in this paper is to restart once meaningful entries have been interpolated to replace the lost entries. When restarting, the Krylov subspace built before the failure is lost and a new sequence of Krylov subspaces is computed. To reduce the computational resource consumption, such a restarting mechanism is implemented in GMRES that it is known to delay the convergence compared to full-GMRES. This delay can be observed in Figure 10, where the convergence history of full-GMRES is the curve denoted "REF" and the one of GMRES(50) is denoted "Restart". Although the convergence history of the faulty executions are much slower than the one of full-GMRES they are not that far (and some even outperform [2]) the convergence of GMRES(50).

On the contrary, CG and BiCGStab do not need to be restarted. In order to evaluate, how the restarting affects the convergence of these two short-term recurrence solvers we display in Figure 11 (Figure 12) the convergence history of CG (resp. BiCGStab) of the method with and without fault. For the 3D Poisson problem, it can be seen that faulty restarted CG (with 70 single faults) does converge twice as slow as classical CG. For BiCGStab, on the Averous/epb0 matrix, the penalty induced by the restarting is even larger while the number of faults is smaller.

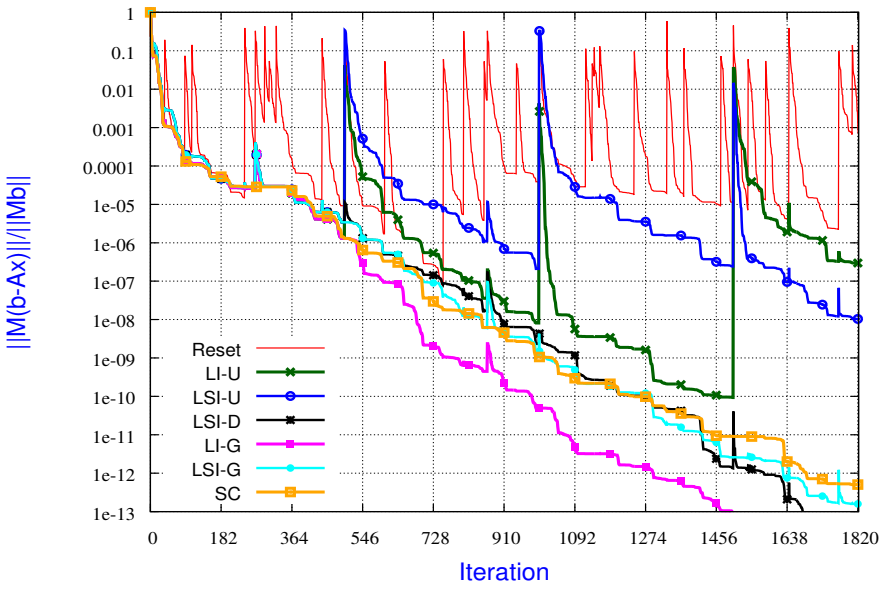


Figure 7: Left preconditioned GMRES on UF Boeing/nasa1824 using 32 processors with 32 single faults and 3 multiple faults

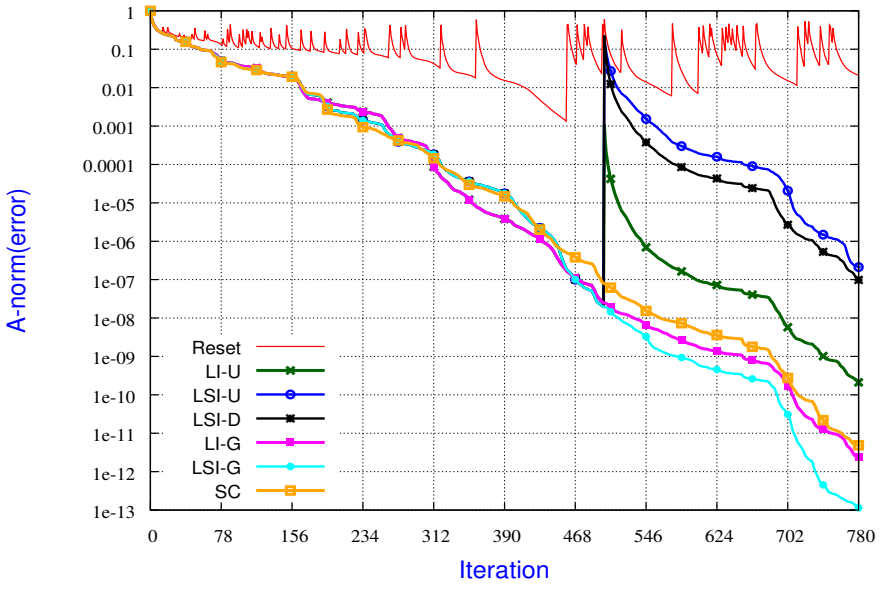


Figure 8: PCG on UF MathWorks/Kuu using 128 processors with 70 single faults and 1 multiple fault

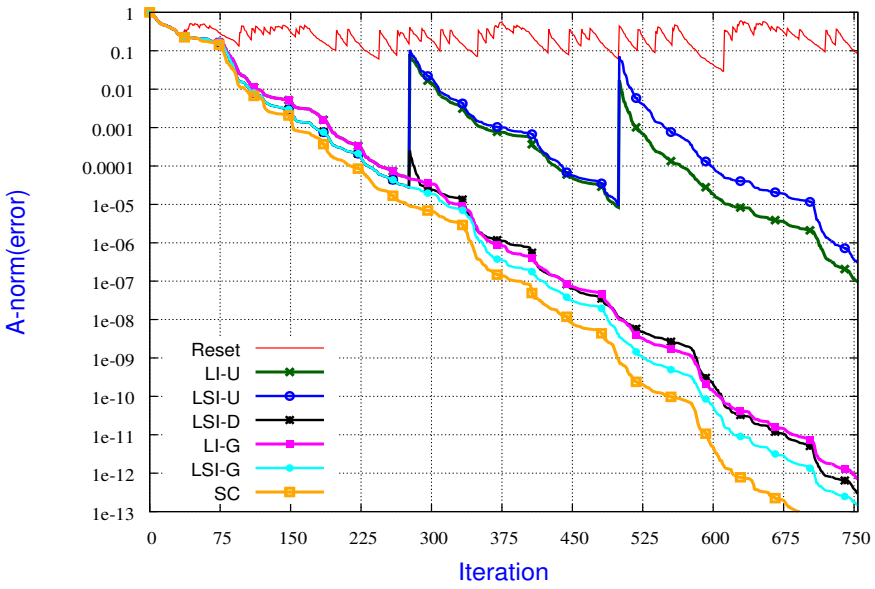


Figure 9: PCG on a 7-point stencil 3D Poisson equation using 32 processors with 67 single faults and 2 multiple faults

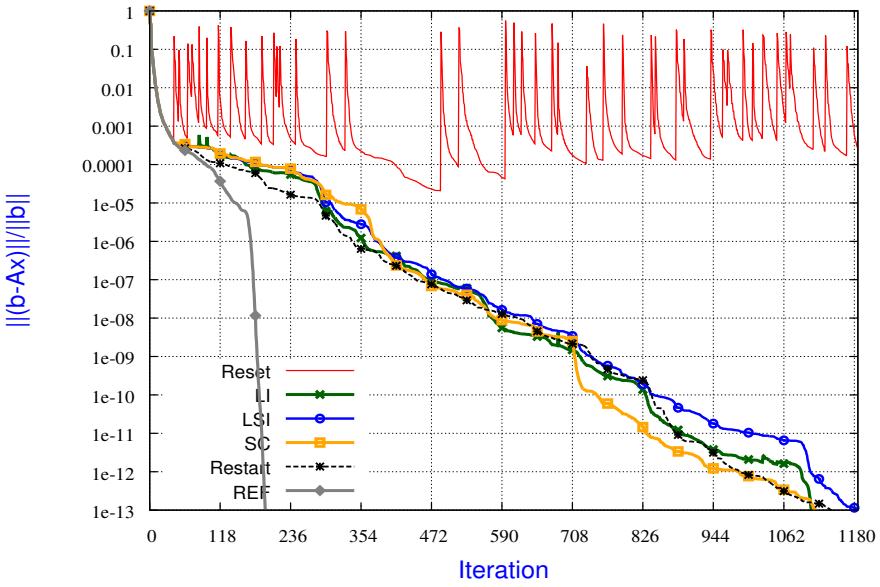


Figure 10: Block diagonal right preconditioned GMRES on Averous/epb0 using 16 processors with 44 single faults

4.5 Cost of interpolation methods

The objective of this paper is to give some qualitative information on the numerical behaviour of recover-restart procedures to enable the Krylov solvers surviving to faults. Nevertheless we also

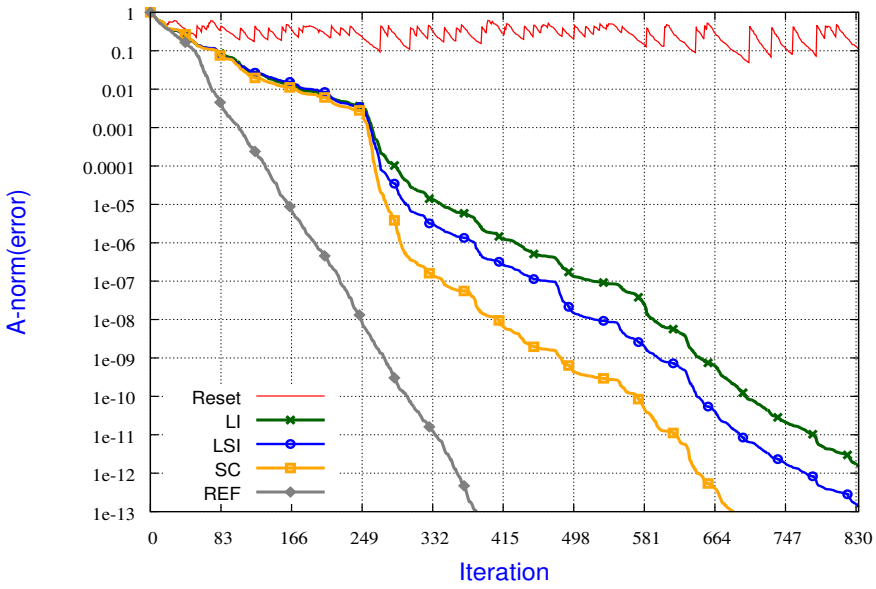


Figure 11: PCG on a 7-point stencil 3D Poisson equation using 16 processors with 70 single faults

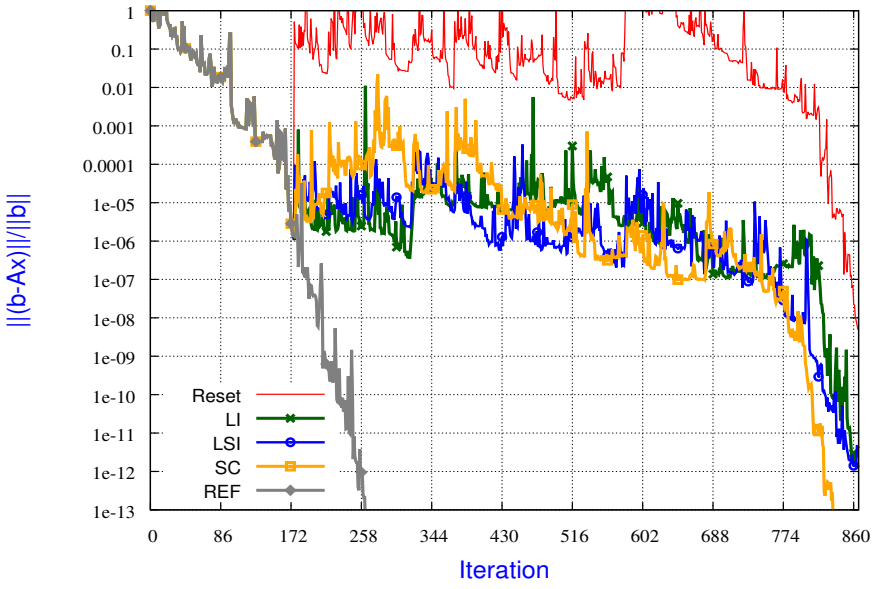


Figure 12: BiCGStab on Averous/epb0 using 16 processors with 15 single faults

look at the computational cost associated with each of the interpolation alternative that should remain affordable to be applicable. In that respect we measure the computational complexity in terms of Flop for the various Krylov solvers as well as for the solution of the sparse linear or least squares problems required by the interpolations. For these latter two kernels we used the Matlab

interface to the UF packages QR-Sparse [17] and Umfpack [16] to get their computational cost. We did not account for the communication in the Krylov solver, but account for the possible imbalance of the work load, i.e., essentially the number of non zeros per block rows. When a fault occurs, we neglect the time to start a new processor and make the assumption that all the processors are involved in the interpolation calculation. We furthermore assume that the parallel sparse LU or sparse QR is ran with a parallel efficiency of 50 %.

We report in Figure 13-15 the convergence history of the Krylov solvers as a function of the Flop count performed. Those figures are the counterparts of Figures 3-5 where the convergence is given as a function of iterations. In can be seen that the qualitative behaviours are comparable, as the extra computational cost associated with the direct solution of the sparse linear algebra problems only represent a few percents of the overall computational effort. On the problems we have considered, the parallel LI (LSI) recovery costs vary from 1 to 8 % (respectively 12 up to 64 %) of one Krylov iteration. The higher cost of LSI with respect to LI accounts for the higher computational complexity of QR compared to LU or Cholesky.

Finally, it is worth mentioning that the SC strategy assumes that the data associated with the lost entries of the iterates have to be recovered from some devices where they are written at each iteration. Depending on the storage device, the time to access the data corresponds to a few thousands/millions of Flop so that the convergence curves in Figures 13-15 should have to be shifted slightly to the left to account for this penalty.

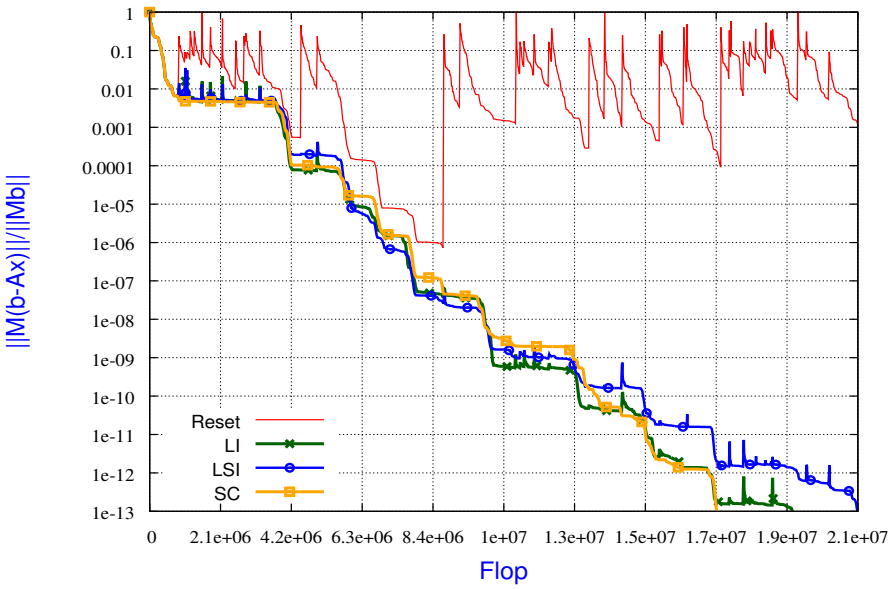


Figure 13: Left preconditioned GMRES on Averous/epb0 using 16 processors with 44 single faults

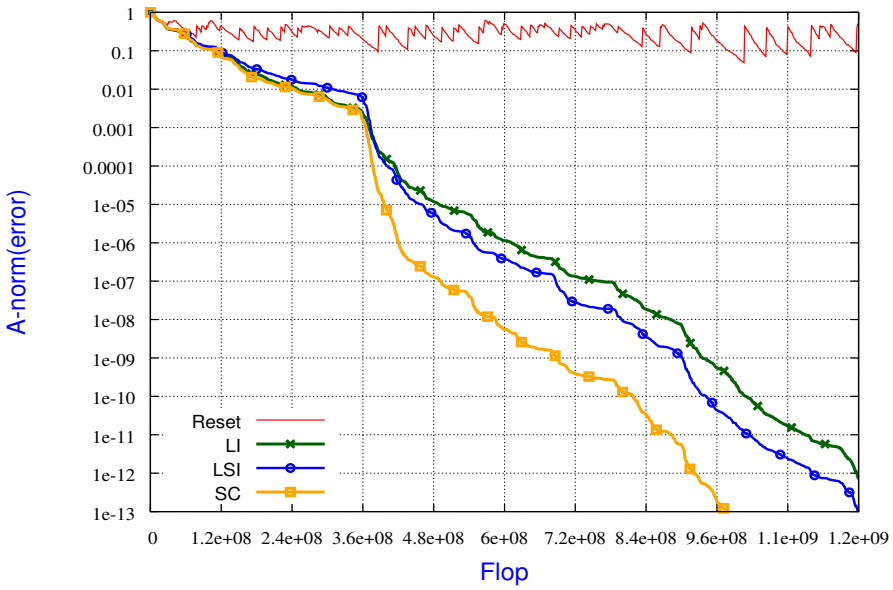


Figure 14: PCG on EDP-SPD using 16 processors with 70 single faults

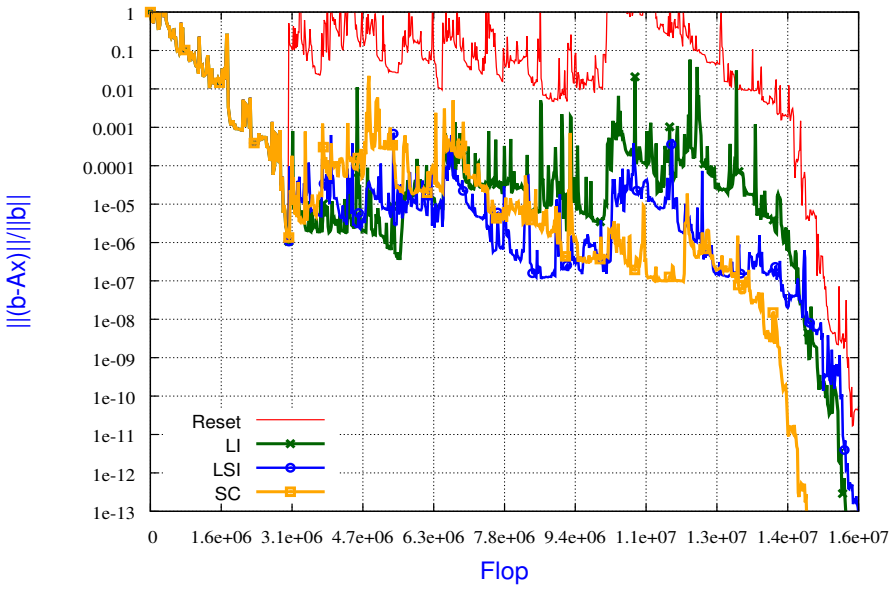


Figure 15: BICGSTAB on Averous/epb0 using 16 processors with 15 single faults

5 Concluding remarks

In this paper we have investigated some recover-restart techniques to design resilient parallel Krylov subspace methods. The recovery techniques are based on interpolation approaches that compute meaningful entries of the iterate lost when a processor fails. We have shown that for SPD matrices the linear interpolation does preserve the A-norm error monotony of the iterates generated by CG and PCG. We have also demonstrated that the least squares interpolation does guarantee the residual norm monotony decrease generated by GMRES and Flexible GMRES as well as for preconditioned GMRES for some class of preconditioners. Because we have considered a restarting procedure after the recovery phase, we have illustrated the numerical penalty induced by the restarting on short terms recurrence Krylov approaches. For CG and Bi-CGStab the convergence delay remains acceptable. For GMRES, where a restarting strategy is usually implemented to cope with the computational constraints related to the computation and storage of the orthonormal Krylov basis, the numerical penalty induced by the recover-restart techniques is negligible and can be beneficial in some cases.

For all the recovery techniques, we have considered a direct solution technique. Altenatively an iterative scheme might be considered with a stopping criterion related to the accuracy level of the iterate when the fault accurs; such a study will be the focus of a future work. Finally, it would be worth assessing the proposed interpolation strategies in efficient fixed point iteration schemes such as multigrid, where the penalty associated with the Krylov restarting would vanish.

Acknowledgements

This work was partially supported by the French research agency ANR in the framework of the RESCUE project (ANR-10-BLANC-0301), in particular the PhD thesis of the fifth author was funded by this project. This research also benefited from the G8-ECS project.

References

[1] J. Anfinson and F. T. Luk. A linear algebraic model of algorithm-based fault tolerance. *IEEE Trans. Comput.*, 37:1599–1604, December 1988. ISSN 0018-9340. doi:10.1109/12.9736.

[2] A. H. Baker, E. R. Jessup, and Tz. V. Kolev. A simple strategy for varying the restart parameter in GMRES(m). *J. Comput. Appl. Math.*, 230(2):751–761, August 2009. ISSN 0377-0427. doi:10.1016/j.cam.2009.01.009.

[3] Prithviraj Banerjee, Joe T. Rahmeh, Craig Stunkel, V. S. Nair, Kaushik Roy, Vijay Balasubramanian, and Jacob A. Abraham. Algorithm-based fault tolerance on a hypercube multiprocessor. *IEEE Trans. Comput.*, 39:1132–1145, September 1990. ISSN 0018-9340. doi:http://dx.doi.org/10.1109/12.57055.

[4] Daniel L. Boley, Richard P. Brent, Gene H. Golub, and Franklin T. Luk. Algorithmic fault tolerance using the Lanczos method. *SIAM J. Matrix Anal. Appl.*, 13:312–332, January 1992. ISSN 0895-4798. doi:http://dx.doi.org/10.1137/0613023.

[5] Marin Bougeret, Henri Casanova, Mikael Rabie, Yves Robert, and Frederic Vivien. Checkpointing strategies for parallel jobs. Rapport de recherche RR-7520, INRIA, April 2011.

[6] Patrick G. Bridges, Kurt B. Ferreira, Michael A. Heroux, and Mark Hoemmen. Fault-tolerant linear solvers via selective reliability. *CoRR*, abs/1206.1390, 2012.

[7] Greg Bronevetsky and Bronis de Supinski. Soft error vulnerability of iterative linear algebra methods. In *Proceedings of the 22nd annual international conference on Supercomputing*, ICS'08, pages 155–164. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-158-3.

[8] Greg Bronevetsky, Daniel Marques, Keshav Pingali, and Paul Stodghill. Automated application-level checkpointing of MPI programs. In *Proceedings of the ninth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '03, pages 84–94. ACM, New York, NY, USA, 2003. ISBN 1-58113-588-2. doi:http://doi.acm.org/10.1145/781498.781513.

[9] Greg Bronevetsky, Daniel Marques, Keshav Pingali, and Paul Stodghill. A system for automating application-level checkpointing of MPI programs. In *LCPC'03*, pages 357–373. 2003.

[10] Greg Bronevetsky, Keshav Pingali, and Paul Stodghill. Experimental evaluation of application-level checkpointing for OpenMP programs. In *Proceedings of the 20th annual international conference on Supercomputing*, ICS '06, pages 2–13. ACM, New York, NY, USA, 2006. ISBN 1-59593-282-8. doi:http://doi.acm.org/10.1145/1183401.1183405.

[11] Franck Cappello, Henri Casanova, and Yves Robert. Preventive migration vs. preventive checkpointing for extreme scale supercomputers. *Parallel Processing Letters*, pages 111–132, 2011.

[12] G. Chen, M. Kandemir, M. J. Irwin, and G. Memik. Compiler-directed selective data protection against soft errors. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, ASP-DAC '05, pages 713–716. ACM, New York, NY, USA, 2005. ISBN 0-7803-8737-6. doi:http://doi.acm.org/10.1145/1120725.1121000.

[13] Zizhong Chen. Algorithm-based recovery for iterative methods without checkpointing. In *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, pages 73–84. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-0552-5. doi:http://doi.acm.org/10.1145/1996130.1996142.

[14] Zizhong Chen and Jack Dongarra. Algorithm-based checkpoint-free fault tolerance for parallel matrix computations on volatile resources. In *Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, pages 97–97. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 1-4244-0054-6.

[15] Teresa Davies and Zizhong Chen. Fault tolerant linear algebra: Recovering from fail-stop failures without checkpointing. In *IPDPS Workshops'10*, pages 1–4. 2010.

[16] T. A. Davis. Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):196–199, 2004.

[17] Timothy A. Davis. Algorithm 915, SuiteSparseQR: Multifrontal multithreaded rank-revealing sparse QR factorization. *ACM Trans. Math. Softw.*, 38(1):1–22, 2011.

[18] J. Dongarra, G. Bosilca, Z. Chen, V. Eijkhout, G. E. Fagg, E. Fuentes, J. Langou, P. Luszczek, J. Pjesivac-Grbovic, K. Seymour, H. You, and S. S. Vadhiyar. Self-adapting numerical software (SANS) effort. *IBM J. Res. Dev.*, 50:223–238, March 2006. ISSN 0018-8646.

[19] Graham E. Fagg and Jack Dongarra. FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world. In *Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 346–353. Springer-Verlag, London, UK, 2000. ISBN 3-540-41010-4.

[20] John A. Gunnels, Robert A. Van De Geijn, Daniel S. Katz, and Enrique S. Quintana-ortÃ. Fault-tolerant high-performance matrix multiplication: Theory and practice. In *Dependable Systems and Networks*, pages 47–56. 2001.

[21] Kuang-Hua Huang and J. A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Trans. Comput.*, 33:518–528, June 1984. ISSN 0018-9340.

[22] Julien Langou, Zizhong Chen, George Bosilca, and Jack Dongarra. Recovery patterns for iterative methods in a parallel unstable environment. *SIAM J. Sci. Comput.*, 30:102–116, November 2007. ISSN 1064-8275. doi:10.1137/040620394.

[23] Yudan Liu, Raja Nassar, Chokchai Leangsuksun, Nichamon Naksinehaboon, Mihaela Paun, and Stephen Scott. An optimal checkpoint/restart model for a large scale high performance computing system. *IEEE Trans. Comput.*, 33:1–9, june 2008. ISSN 1530-2075.

[24] Konrad Malkowski, Padma Raghavan, and Mahmut T. Kandemir. Analyzing the soft error resilience of linear solvers on multicore multiprocessors. In *IPDPS'10*, pages 1–12. 2010.

[25] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numerical Analysis*, 12:617 – 629, 1975.

[26] J. S. Plank, Y. Kim, and J. Dongarra. Fault tolerant matrix operations for networks of Workstations Using Diskless Checkpointing. *Journal of Parallel and Distributed Computing*, 43(2):125–138, June 1997.

[27] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003. ISBN 0898715342.

A More experiments

A.1 Numerical behavior in single fault cases

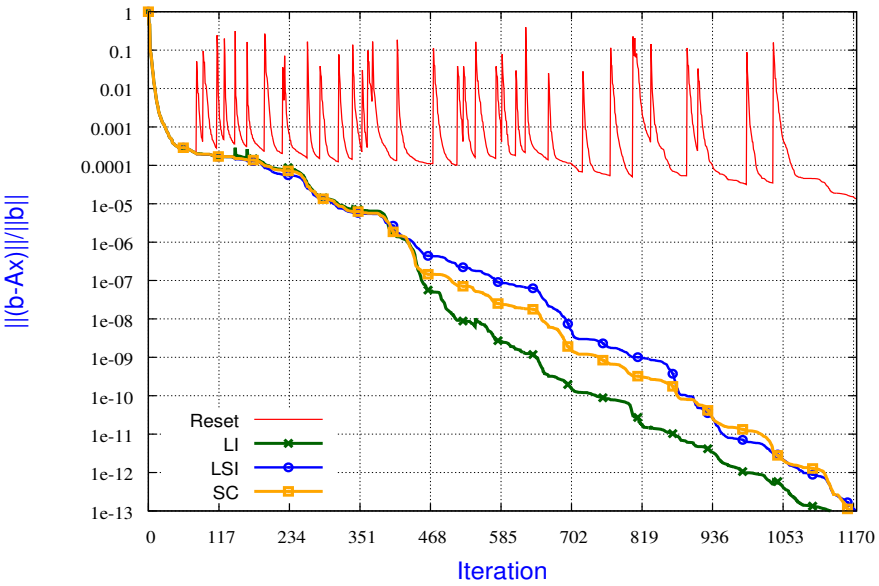


Figure 16: Right block diagonal preconditioned GMRES on UF Averous/epb0 using 32 processors with 36 single faults

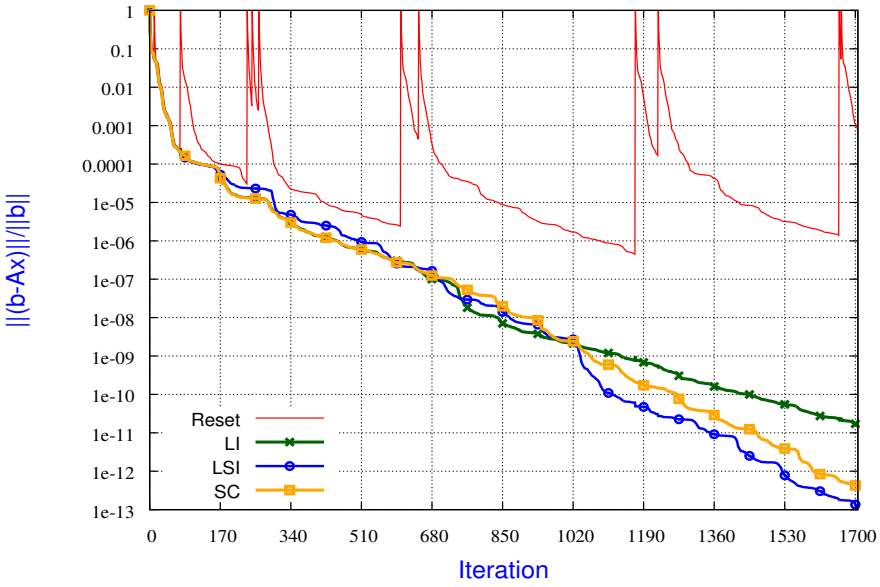


Figure 17: Right block diagonal preconditioned GMRES on UF Boeing/nasa1824 using 8 processors with 11 single faults

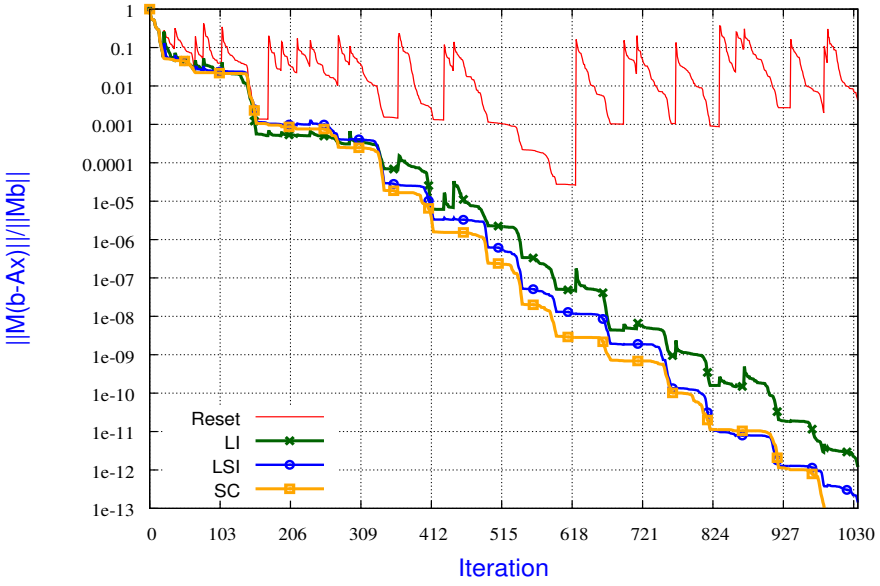


Figure 18: Left preconditioned GMRES on UF HB/jagmesh9 using 32 processors with 25 single faults

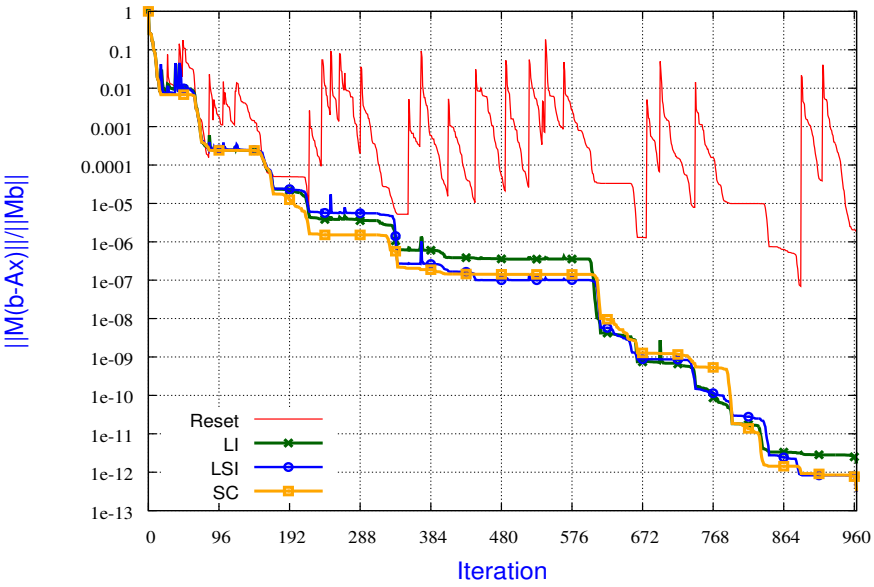


Figure 19: Left preconditioned GMRES on UF Nasa/barth4 using 256 processors with 33 single faults

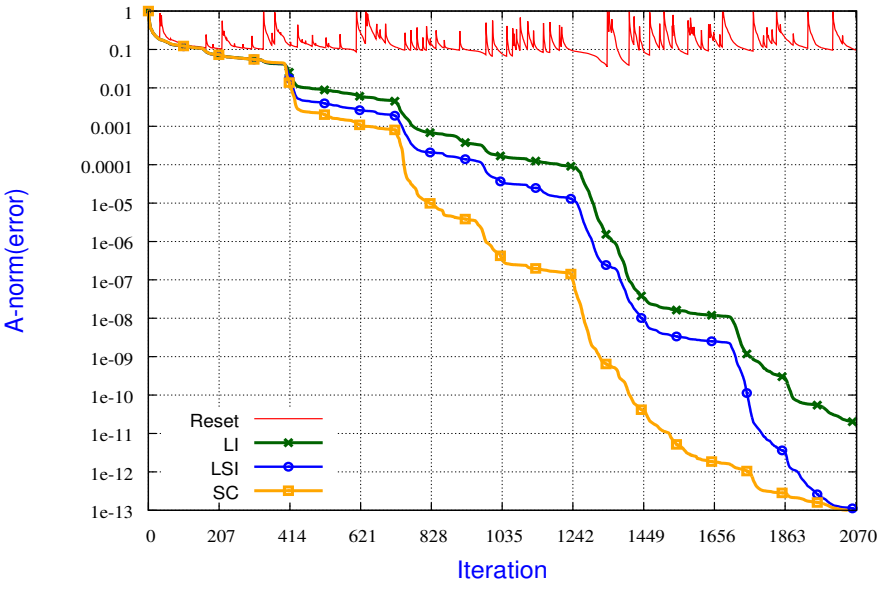


Figure 20: PCG on UF ACUSIM/Pres/Poisson using 16 processors with 99 single faults

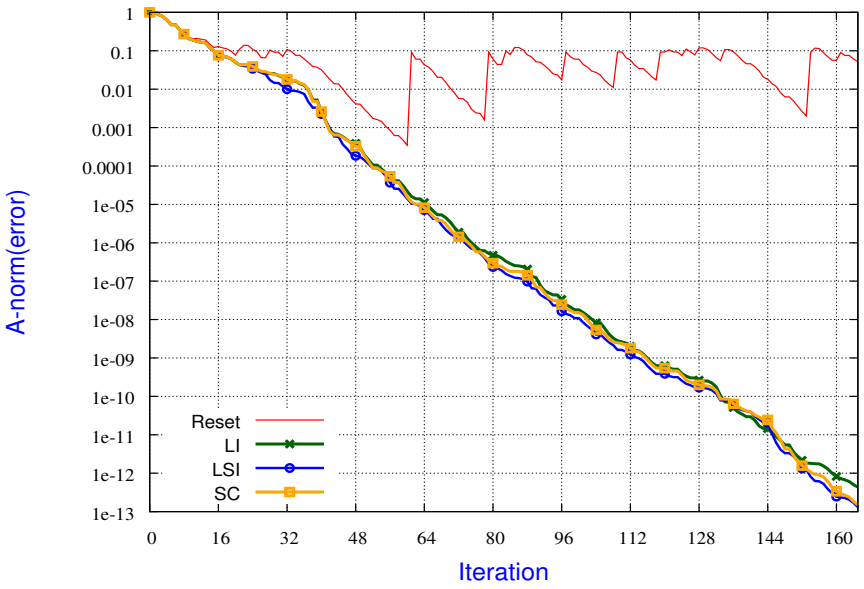


Figure 21: PCG on UF Norris/fv1 using 256 processors with 21 single faults

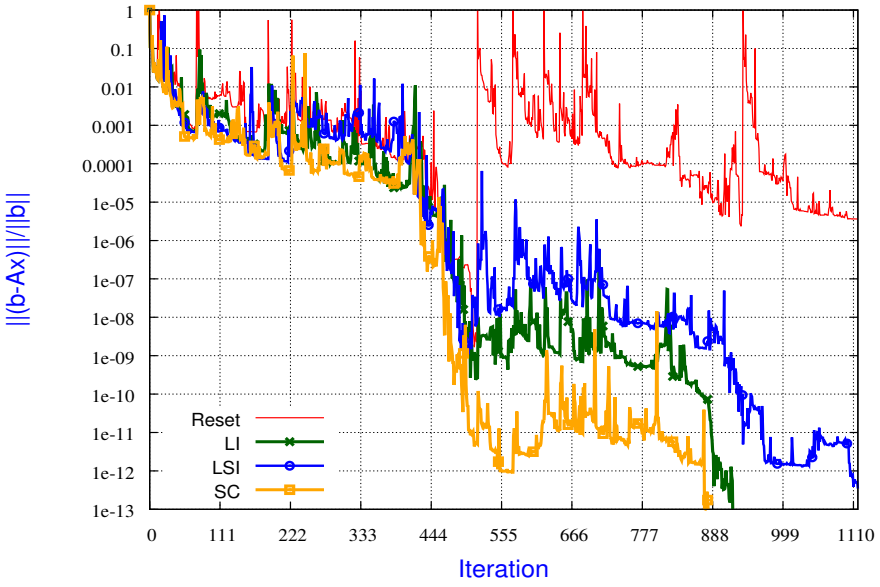


Figure 22: BICGSTAB on UF Boeing/nasa1824 using 32 processors with 12 single faults

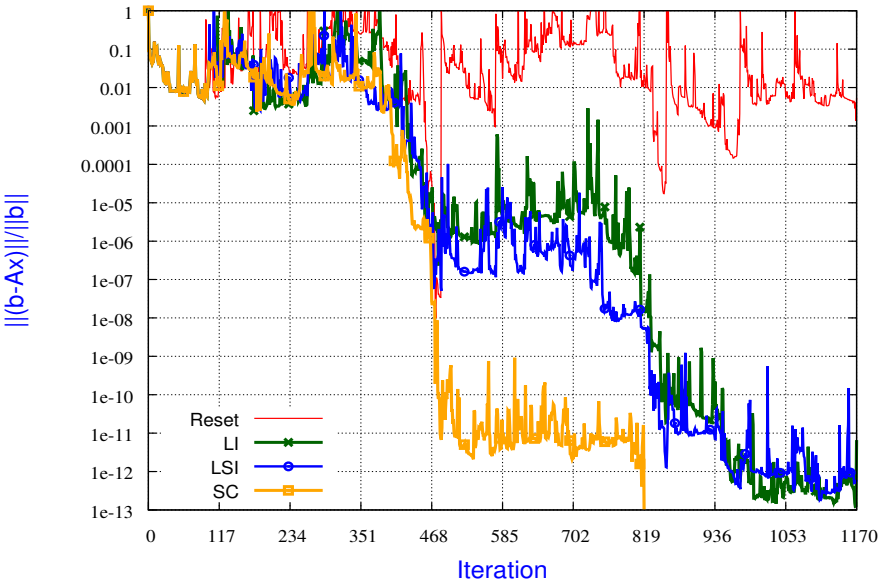


Figure 23: BICGSTAB on UF Bai/olm2000 using 32 processors with 12 single faults

A.2 Numerical behavior in multiple fault cases

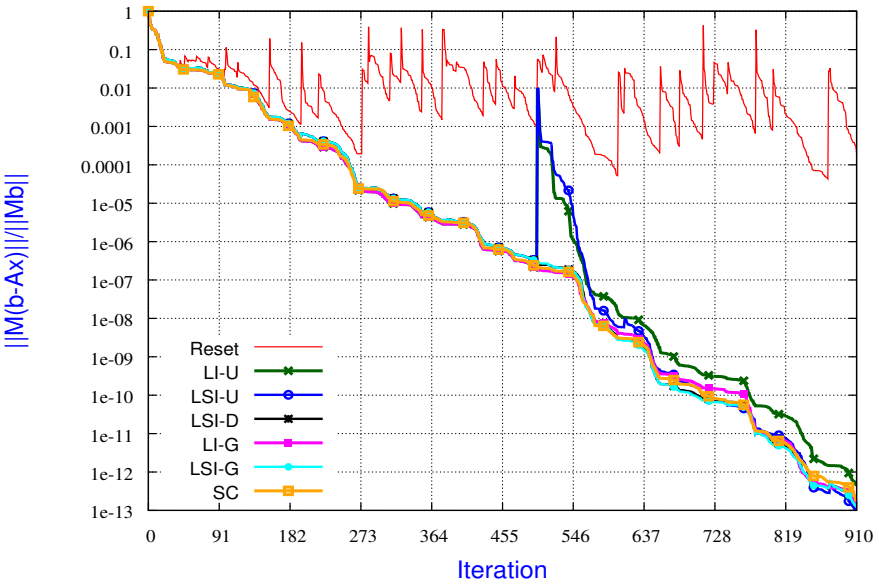


Figure 24: Left preconditioned GMRES on UF Rajat/rajat03 using 256 processors with 41 single faults and 1 multiple fault

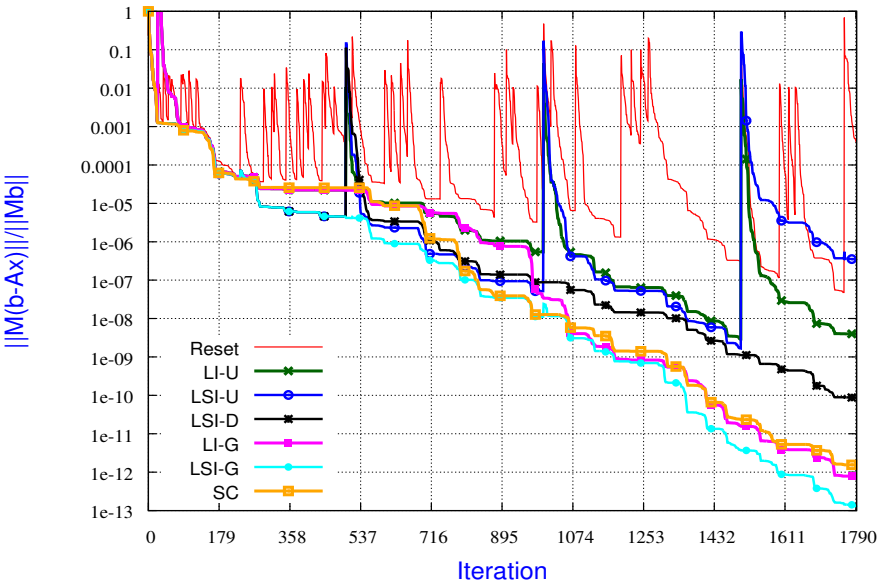


Figure 25: Left preconditioned GMRES on UF Boeing/nasa1824 using 256 processors with 46 single faults and 3 multiple faults

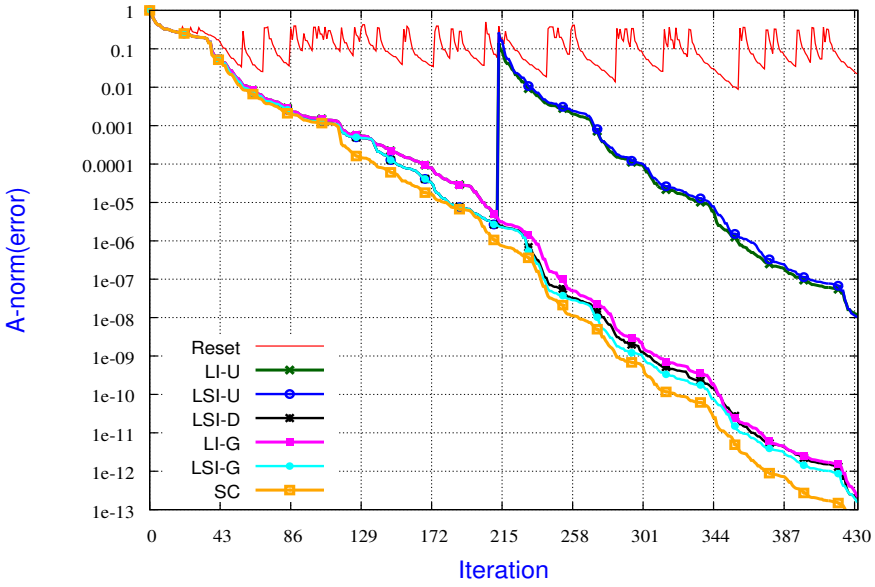


Figure 26: PCG on UF Nasa/nasa2146 using 64 processors with 52 single faults and 1 multiple fault

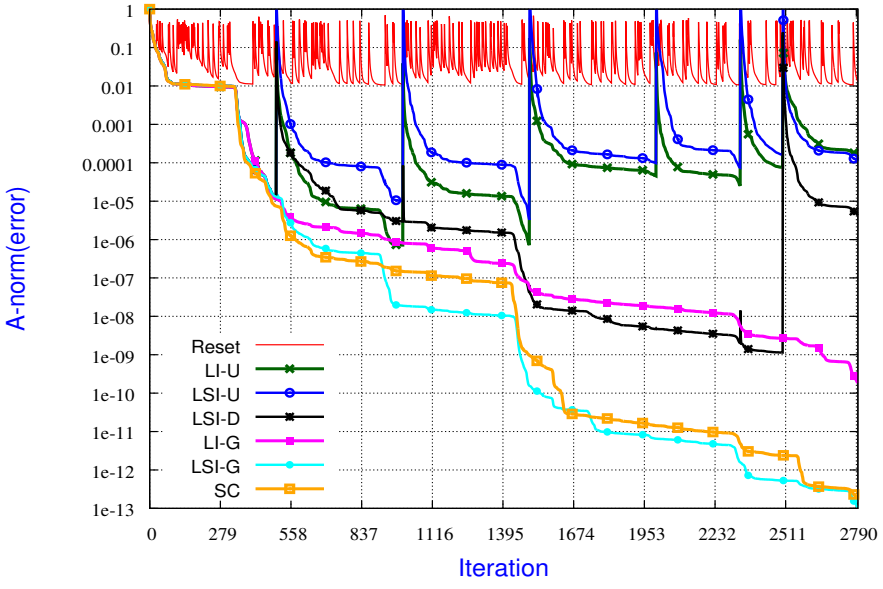


Figure 27: PCG on UF Cylshell/s1rmq4m1 using 64 processors with 209 single faults and 6 multiple faults

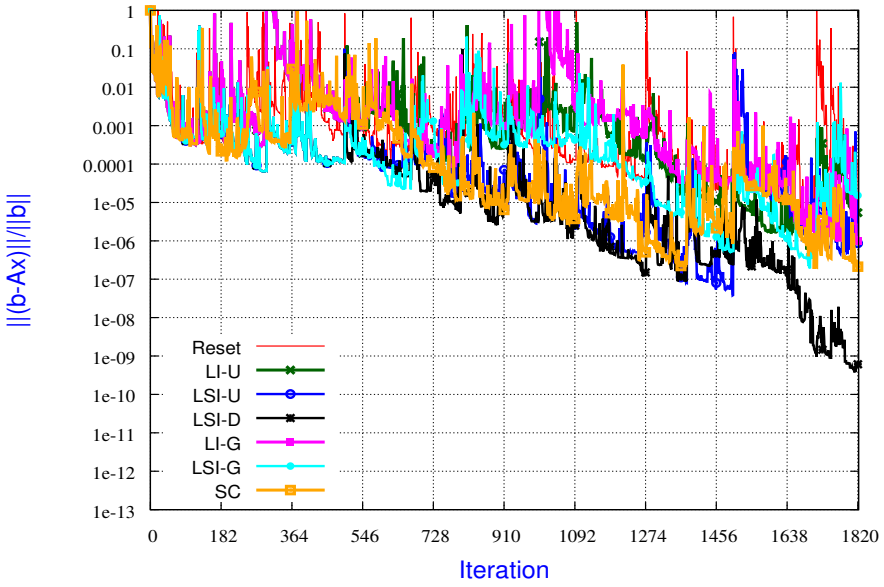


Figure 28: BICGSTAB on UF Boeing/nasa1824 using 64 processors with 22 single faults and 1 multiple fault

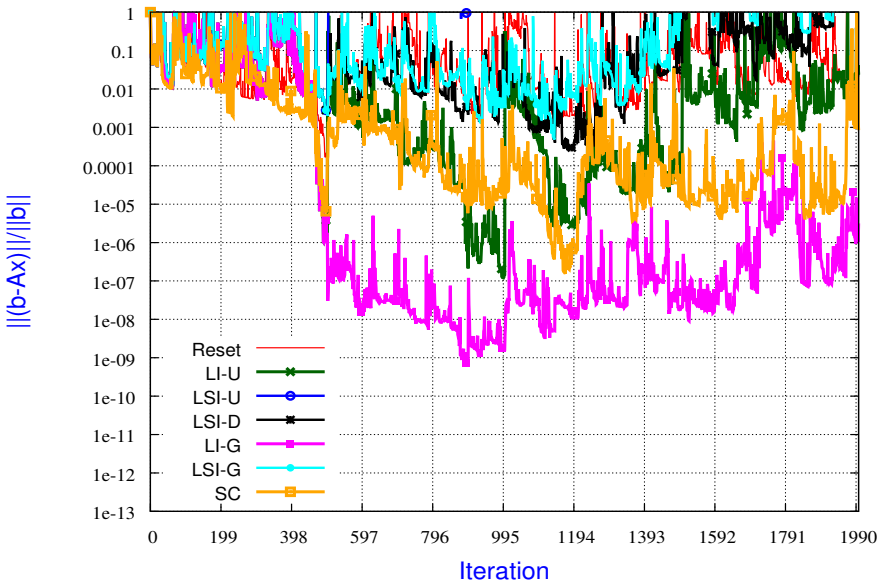
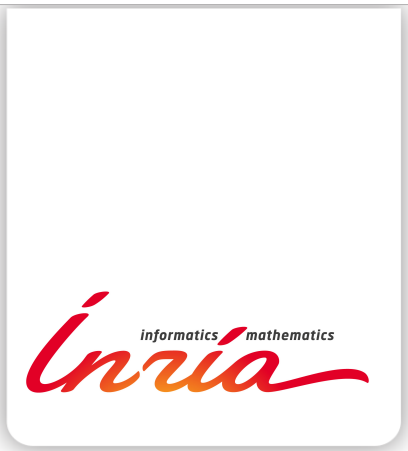


Figure 29: BICGSTAB on UF Boeing/nasa1824 using 256 processors with 38 single faults and 1 multiple fault



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