On numerical resilience in linear algebra

Mawussi Zounon

université de BORDEAUX

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Mawussi Zounon**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

On numerical resilience in linear algebra

Soutenue le : 1er avril 2015

Membres du jury :

Frédéric Vivien .	Président
Mike Heroux ..	Rapporteur
Peter Arbenz ..	Rapporteur
Karl Meerbergen	Examinateur
Emmanuel Agullo	Directeur de Thèse
Luc Giraud ...	Directeur de Thèse

Acknowledgments

My advisor Luc Giraud and my co-advisor Emmanuel Agullo deserve thanks for many things. Notably, for their confidence, all their support and the nice research environment during my graduate studies. They have provided guidance at key moments in my work while also allowing me to work independently the majority of the time. I would like to express my thanks to Professor Jean Roman and Abdou Guermouche. Your suggestions have been precious for the development of this thesis content. I would like to extend my sincerest thanks and appreciation to Professors Peter Arbenz and Mike Heroux, Karl Meerbergen, Frédéric Vivien for having accepted to act as referee or as regular committee member for my defense, and for their precious suggestions.

Because of the HiePACS research team environment, I have crossed paths with many graduate students and postdocs who have influenced and enhanced my research. I take this opportunity to thank all graduate students and postdocs of the HiePACS research team, without forgetting our kind team assistant, Chrystel Plumejeau.

Last but not least, my deepest thanks goes to my sweet wife, Anna, who has consistently supported and encouraged me through the ups and downs, all these years.

Conception d'algorithmes numériques pour la résilience en algèbre linéaire

Résumé

Comme la puissance de calcul des systèmes de calcul haute performance continue de croître, en utilisant un grand nombre de cœurs CPU ou d'unités de calcul spécialisées, les applications hautes performances destinées à la résolution des problèmes de très grande échelle sont de plus en plus sujettes à des pannes. En conséquence, la communauté de calcul haute performance a proposé de nombreuses contributions pour concevoir des applications tolérantes aux pannes. Cette étude porte sur une nouvelle classe d'algorithmes numériques de tolérance aux pannes au niveau de l'application qui ne nécessite pas de ressources supplémentaires, à savoir, des unités de calcul ou du temps de calcul additionnel, en l'absence de pannes. En supposant qu'un mécanisme distinct assure la détection des pannes, nous proposons des algorithmes numériques pour extraire des informations pertinentes à partir des données disponibles après une pannes. Après l'extraction de données, les données critiques manquantes sont régénérées grâce à des stratégies d'interpolation pour constituer des informations pertinentes pour redémarrer numériquement l'algorithme. Nous avons conçu ces méthodes appelées techniques d'Interpolation-restart pour des problèmes d'algèbre linéaire numérique tels que la résolution de systèmes linéaires ou des problèmes aux valeurs propres qui sont indispensables dans de nombreux noyaux scientifiques et applications d'ingénierie. La résolution de ces problèmes est souvent la partie dominante; en termes de temps de calcul, des applications scientifiques. Dans le cadre solveurs linéaires du sous-espace de Krylov, les entrées perdues de l'itération sont interpolées en utilisant les entrées disponibles sur les nœuds encore disponibles pour définir une nouvelle estimation de la solution initiale avant de redémarrer la méthode de Krylov. En particulier, nous considérons deux politiques d'interpolation qui préservent les propriétés numériques clés de solveurs linéaires bien connus, à savoir la décroissance monotone de la *norme-*$\mathcal{A}$ de l'erreur du gradient conjugué ou la décroissance monotone de la norme résiduelle de GMRES. Nous avons évalué l'impact du taux de pannes et l'impact de la quantité de données perdues sur la robustesse des stratégies de résilience conçues. Les expériences ont montré que nos stratégies numériques sont robustes même en présence de grandes fréquences de pannes, et de perte de grand volume de données. Dans le but de concevoir des solveurs résilients de résolution de problèmes aux valeurs propres, nous avons modifié les stratégies d'interpolation conçues pour les systèmes linéaires. Nous avons revisité les méthodes itératives de l'état de l'art pour la résolution des problèmes de valeurs propres creux à la lumière des stratégies d'Interpolation-restart. Pour chaque méthode considérée, nous avons adapté les stratégies d'Interpolation-restart pour régénérer autant d'informations spectrale que possible. Afin d'évaluer la performance de nos stratégies numériques, nous avons considéré un solveur parallèle hybride (direct/itérative) pleinement fonctionnel nommé MaPHyS pour la résolution des systèmes linéaires creux, et nous proposons des solutions numériques pour concevoir une version tolérante aux pannes du solveur. Le solveur étant hybride, nous nous concentrons dans cette étude sur l'étape de résolution itérative, qui est souvent l'étape dominante dans la pratique. Les

solutions numériques proposées comportent deux volets. A chaque fois que cela est possible, nous exploitons la redondance de données entre les processus du solveur pour effectuer une régénération exacte des données en faisant des copies astucieuses dans les processus. D'autre part, les données perdues qui ne sont plus disponibles sur aucun processus sont régénérées grâce à un mécanisme d'interpolation. Pour évaluer l'efficacité des solutions numériques proposées, elles ont été mises en œuvre dans le solveur parallèle MaPHyS pour résoudre des problèmes de grande échelle sur un grand nombre de ressources de calcul (allant jusqu'à 12288 coeurs CPU).

Mots-clés

Tolerance aux pannes, résilience, interpolation, restauration de donnée, robustesse, méthodes itératives, méthodes de type Krylov, sous espaces de Krylov, problèmes de valeurs propres, systèmes linéaires, GMRES, CG, PCG, Bicgstab, preconditionnement, méthode de puissance, itération de sous espace, Arnoldi, IRAM, Jacobi-Davidson, Simulation numérique, parallélisme, calcul scientifique.

On numerical resilience in linear algebra

Abstract

As the computational power of high performance computing (HPC) systems continues to increase by using huge number of cores or specialized processing units, HPC applications are increasingly prone to faults. This study covers a new class of numerical fault tolerance algorithms at application level that does not require extra resources, i.e., computational unit or computing time, when no fault occurs. Assuming that a separate mechanism ensures fault detection, we propose numerical algorithms to extract relevant information from available data after a fault. After data extraction, well chosen part of missing data is regenerated through interpolation strategies to constitute meaningful inputs to numerically restart the algorithm. We have designed these methods called Interpolation-restart techniques for numerical linear algebra problems such as the solution of linear systems or eigen-problems that are the inner most numerical kernels in many scientific and engineering applications and also often ones of the most time consuming parts. In the framework of Krylov subspace linear solvers the lost entries of the iterate are interpolated using the available entries on the still alive nodes 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 linear solvers, namely the monotony decrease of the A-norm of the error of the conjugate gradient or the residual norm decrease of GMRES. We assess the impact of the fault rate and the amount of lost data on the robustness of the resulting linear solvers. For eigensolvers, we revisited state-of-the-art methods for solving large sparse eigenvalue problems namely the Arnoldi methods, subspace iteration methods and the Jacobi-Davidson method, in the light of Interpolation-restart strategies. For each considered eigensolver, we adapted the Interpolation-restart strategies to regenerate as much spectral information as possible. Through intensive experiments, we illustrate the qualitative numerical behavior of the resulting schemes when the number of faults and the amount of lost data are varied; and we demonstrate that they exhibit a numerical robustness close to that of fault-free calculations. In order to assess the efficiency of our numerical strategies, we have considered an actual fully-featured parallel sparse hybrid (direct/iterative) linear solver, MaPHyS, and we proposed numerical remedies to design a resilient version of the solver. The solver being hybrid, we focus in this study on the iterative solution step, which is often the dominant step in practice. The numerical remedies we propose are twofold. Whenever possible, we exploit the natural data redundancy between processes from the solver to perform an exact recovery through clever copies over processes. Otherwise, data that has been lost and is not available anymore on any process is recovered through Interpolation-restart strategies. These numerical remedies have been implemented in the MaPHyS parallel solver so that we can assess their efficiency on a large number of processing units (up to $12,288$ CPU cores) for solving large-scale real-life problems.

Keywords

Fault tolerance, resilience, interpolation, recovery, robustness, iterative methods, Krylov methods, Krylov subspaces, eigenvalue problems, linear systems, GMRES, flexible GMRES, CG, power method, subspace iteration, Arnoldi, IRAM, Jacobi-Davidson, HPC, large scale numerical simulations.

Contents

Nomenclature

ℓ	Index to enumerate eigenpairs $\ell \in [1, nev]$
ϵ	Threshold
λ	Eigenvalue
$\mathcal{I}$	Identity matrix
$\mathcal{U}$	Subspace
$\mathcal{X}$	Eigenspace
C	Rayleigh quotient
$\mathcal{N}$	Number of nodes involved in a simulation
τ	Target value for convergence of eigenvalues
$\tilde{m}$	Size of a restarted basis (in IRAM and Jacobi-Davidson
A	General square matrix
D	Diagonal matrix which diagonal entries are eigenvalues of a given square matrix
e	Column of identity matrix
ER	Enforced recovery
G	Matrix of eigenvectors $g_1, g_2, \ldots$
g	Eigenvector of C, or H or any factorized form of A
H	Hessenberg matrix
i	First inner loop iteration counter
I_p	Set of rows mapped to node p
IR	Interpolation-restart

j Second inner loop iteration counter

k Outer loop iteration counter

m Restart parameter related to variants of Arnoldi algorithm

n Number of rows in a given matrix

$nconv$ Number of converged Schur vectors (in J-D)

nev Number of eigenvector to converge

p Faulty node index

q Non faulty node index

s Number of additional recovered Ritz vectors

T Triangular matrix from partial Schur decomposition A

U Subspace basis

u Eigenvector

U^H Transpose conjugate of U

V Subspace basis

v Vector of basis V

Z Matrix of Schur vectors

z Schur vector

B-CGSTAB Biconjugate gradient stablized

CG Conjugate gradient

GMRES Generalized minimal residual

PCG Preconditioned conjugate gradient

Résumé en Français

Introduction

Il est admis aujourd'hui que la simulation numérique est le troisième pilier pour le développement de la découverte scientifique au même niveau que la théorie et l'expérimentation. Au cours de ces dernières décennies, il y a eu dans d'innombrables domaines scientifiques, ingénierie et sociétaux des avancées des simulations à grande échelle grâce à l'élaboration des calculs haute performance(HPC), des algorithmes et des architectures. Ces outils de simulation puissants ont fourni aux chercheurs la capacité de trouver des solutions efficaces pour certaines des questions scientifiques plus difficiles et des problèmes de la médecine de l'ingénierie et de la biologie, la climatologie, des nanotechnologies, l'énergie et l'environnement. Ces simulations numériques nécessitent des machines à grande puissance de calcul. Comme la puissance de calcul des systèmes de calcul haute performance continue de croître, en utilisant un grand nombre de cœurs CPU ou d'unités de calcul spécialisées, les applications hautes performances destinées à la résolution des problèmes de très grande échelle sont de plus en plus sujettes à des pannes. En conséquence, la communauté de calcul haute performance a proposé de nombreuses contributions pour concevoir des applications tolérantes aux pannes. Ces contributions peuvent être orientées système, théorique ou numérique.

Problème de pannes dans les solveurs numériques d'algèbre linéaire

L'algèbre linéaire numérique joue un rôle central dans la résolution de nombreux problèmes de nos jours. Afin de comprendre les phénomènes ou pour résoudre les problèmes, les scientifiques utilisent des modèles mathématiques. Les solutions obtenues à partir de modèles mathématiques sont souvent des solutions satisfaisantes aux problèmes complexes dans des domaines tels que la prévision météorologique, la trajectoire d'un engin spatial, la simulation de crash de voiture, etc. Dans de nombreux domaines scientifiques tels que l'électrostatique, l'électrodynamique, l'écoulement du fluide, l'élasticité, ou la mécanique quantique, les problèmes sont largement modélisés par des équations aux dérivées partielles

(EDP). Le moyen commun pour résoudre les EDP est d'approcher la solution qui est continue par des équations discrètes qui impliquent un nombre fini d'inconnus mais souvent très grand [136, Chapitre 2]. Cette stratégie est appelée discrétisation. Les stratégies de discrétisation conduisent à de grandes matrices creuses. Ainsi les problèmes du monde réel deviennent des problèmes numériques d'algèbre linéaire. Il y a beaucoup de problèmes d'algèbre linéaire, mais ce travail se concentre sur la résolution de systèmes d'équations linéaires $\mathcal{A}x = b$ et des problèmes aux valeurs propres $\mathcal{A}x = \lambda x$. Les solveurs parallèles d'algèbre linéaire sont souvent les noyaux numériques indispensables dans de nombreuses applications scientifiques; par conséquent, l'une des parties dominantes en terme de temps de calcul. En outre, dans les systèmes à grande échelle, le temps entre deux pannes consécutives peut être inférieur au temps requis par les solveurs pour finir les calculs. Par conséquent, il devient critique de concevoir des solveurs parallèles d'algèbre linéaire qui peuvent survivre aux pannes. De nombreuses études se concentrent sur la conception de systèmes HPC fiables, mais avec le nombre croissant de composants impliqués dans ces systèmes, les pannes deviennent de plus en plus fréquentes. Les études [33, 41] tendent à démontrer qu'avec l'augmentation permanente des pannes, les stratégies de sauvegardes classiques peuvent être insuffisantes pour pallier les problèmes de pannes des systèmes HPC. Nous proposons d'étudier certains solveurs linéaires afin d'exploiter leur propriétés numériques pour la conception des algorithmes numériques de résilience. Dans cette thèse, nous étudions une nouvelle classe d'algorithmes numériques de résilience au niveau de l'application qui ne nécessitent pas de ressources supplémentaires, à savoir des unités de calcul ou de temps de calcul, en l'absence de pannes. En supposant qu'un mécanisme distinct dans la pile logiciel assure la détection des pannes, et nous proposons des algorithmes numériques appelés des techniques d'interpolation-Restart, pour survivre aux pannes.

Première contribution: Stratégies d'Interpolation-Restart pour les solveurs linéaires de Krylov

Dans la première (Part I), nous avons conçu les techniques d'interpolation-Restart pour des solveurs linéaires de sous-espace de Krylov et des solveurs de problèmes aux valeurs propres bien connus. Dans le cadre des solveurs linéaires de sous-espace de Krylov, dans le chapitre 2, les données perdues sont interpolées en utilisant les données disponibles sur les nœuds de calcul restants pour définir une nouvelle estimation de la solution initiale avant de redémarrer la méthode de Krylov. En particulier, nous avons considéré deux methodes d'interpolation qui préservent les propriétés numériques clés de solveurs linéaires bien connus, à savoir la décroissance monotone de la *norme-*$\mathcal{A}$ de l'erreur du gradient conjugué ou la décroissance monotone de la norme résiduelle de GMRES. Nous avons évalué l'impact du taux de pannes et l'impact de la quantité de données perdues sur la robustesse des stratégies de résilience conçues. Les expériences ont montré que nos stratégies numériques sont robustes même en présence de grandes fréquences de pannes, et de perte de grand volume de données.

Deuxième contribution: Stratégies d'Interpolation-Restart pour les solveurs de problèmes aux valeurs propres

Le calcul des paires propres (valeurs propres et vecteurs propres) des matrices creuses de grande taille est requis dans de nombreuses applications scientifiques et d'ingénierie, par exemple dans les problèmes d'analyse de stabilité. Cependant, dans les systèmes de calcul haute performance à grande échelle à venir, il est prévu que l'intervalle de temps entre deux pannes consécutives soit plus petit que le temps requis par les solveurs pour finir les calculs. Il est donc nécessaire de concevoir des solveurs parallèles de problèmes aux valeurs propres qui peuvent survivre aux pannes. Dans le but de concevoir des solveurs résilients pour la résolution de problèmes aux valeurs propres, nous avons modifié les stratégies d'interpolation conçues pour les systèmes linéaires. Le Chapitre 3 est donc dédié à des stratégies d'interpolation-Restart pour des méthodes itératives de résolution de problèmes aux valeurs propres où il y a plus de flexibilités pour adapter les idées Interpolation-Restart. Par exemple, pour le solveur Jacobi-Davidson, les interpolations sont appliquées aux vecteurs de Schur qui ont convergé ainsi que les meilleurs vecteurs propres de l'espace de recherche. Après une panne, ce nouvel ensemble de vecteurs est utilisé comme estimation initiale pour redémarrer les itérations de Jacobi-Davidson. La plupart des méthodes itératives de résolution de problèmes aux valeurs propres couramment utilisés tels que Arnoldi, IRAM ou l'algorithme d'itération de sous-espaces ont été revisités à la lumière des stratégies numériques de tolérance aux pannes. Nous illustrons la robustesse des stratégies proposées par de nombreuses expériences numériques.

Troisième contribution: Implémentation parallèle des stratégies résilience numérique dans un solveur hybride (direct/itératif)

Dans la deuxième partie II de cet travail, nous avons considéré un solveur parallèle hybride (direct/itérative) pleinement fonctionnel nommé MaPHyS pour la résolution des systèmes linéaires creux, et nous avons proposé des solutions numériques pour concevoir une version tolérante aux pannes du solveur. Le solveur étant hybride, nous nous concentrons dans cette étude sur l'étape de résolution itérative, qui est souvent l'étape dominante dans la pratique. Nous supposons en outre qu'un mécanisme distinct assure la détection des pannes et qu'une couche système fournit un support qui remet l'environnement (processus, ...) dans un état fonctionnel. Ce manuscrit se focalise donc sur (et seulement sur) des stratégies pour la régénération des données perdues après que la panne ait été détectée (la détection de panne est un autre problème non traité dans cette étude) et que le système soit à nouveau fonctionnel (un autre problème orthogonal non étudié ici). Les solutions numériques proposées comportent deux volets. A chaque fois que cela est possible, nous exploitons la redondance de données entre les processus du solveur pour effectuer une régénération exacte des données en faisant des copies astucieuses dans les processus. D'autre part, les données

perdues qui ne sont plus disponibles sur aucun processus sont régénérées grâce à un mécanisme d'interpolation. Ce mécanisme est dérivé des stratégies d'interpolation en prenant en compte les propriétés spécifiques au solveur hybride cible. Pour évaluer l'efficacité des solutions numériques proposées, elles ont été mises en œuvre dans le solveur parallèle MaPHyS pour résoudre des problèmes de grande échelle sur un grand nombre de ressources de calcul (allant jusqu'à 12288 coeurs CPU).

Conclusion et perspectives

L'objectif principal de cette thèse était d'explorer les schémas numériques pour la conception de stratégies de résilience qui permettent aux solveurs numériques parallèles algèbre linéaire de survivre aux pannes. Dans le contexte des solveurs d'algèbre linéaire, nous avons étudié des approches numériques pour régénérer des données dynamiques significatives afin de redémarrer numériquement le solveur. Nous avons présenté de nouveaux algorithmes de résilience numérique appelés des stratégies d'Interpolation-Restart pour la résolution de systèmes d'équations linéaires au chapitre 2. En particulier, nous avons examiné deux méthodes pertinentes qui préservent les propriétés numériques clés des solveurs linéaires bien connus. Nous avons démontré que nos stratégies sont très robustes indépendamment du taux de pannes et du volume de données perdues. Dans le chapitre 3, nous avons adapté les stratégies d'Interpolation-Restart pour concevoir des techniques numériques de résilience pour des solveurs de résolution de problèmes aux valeurs propres. Les caractéristiques numériques de ces derniers offrent la flexibilité de concevoir des méthodes de résilience assez efficaces et robustes.

Après avoir évalué la robustesse de nos stratégies numériques dans des conditions stressantes simulées par des taux de pannes élevés et la perte de grand volume de données, nous nous sommes concentrés sur leur extension à un solveur hybride parallèle d'algèbre linéaire existant, MaPHyS. Avons avons exploité les propriétés d'implémentation parallèle de MaPHyS à savoir la redondance de données aussi bien que les propriétés numériques du préconditionneur, pour concevoir une version résiliente du solveur. Toutes les expériences ont montré que la stratégie conçue a un comportement numérique extrêmement robuste avec un très faible surcoût en terme de temps de calcul.

Enfin, nos stratégies de résilience numériques peuvent être efficacement combinées avec des mécanismes de tolérance aux pannes existants tels que les techniques de checkpoint/restart pour concevoir une boite à outils de tolérance aux pannes pour les simulations à grande échelle.

Introduction

It is admitted today that numerical simulation is the third pillar for the development of scientific discovery at the same level as theory and experimentation. Over the last few decades, there have been innumerable science, engineering and societal breakthroughs enabled by large-scale simulations thanks to the development of high performance computing (HPC) applications, algorithms and architectures. These powerful simulation tools have provided researchers with the ability to computationally find efficient solutions for some of the most challenging scientific questions and problems in engineering medicine and biology, climatology, nanotechnology, energy and environment.

In this landscape, parallel sparse linear algebra solvers are often the innermost numerical kernels in many scientific and engineering applications; consequently, one of the most time consuming parts. Furthermore in today's large-scale systems, the time between two consecutive faults may be smaller than the time required by linear algebra solvers to complete. Consequently, it becomes critical to design parallel linear algebra solvers which can survive to faults. Many studies focus on designing reliable HPC systems, but with the increasing number of components involved in these systems, faults become more frequent. Studies [33, 41] tend to demonstrate that with the permanent increase of faults, classical checkpoint strategies may be insufficient to recover from HPC system faults. We propose to revisit some linear solvers to exploit their numerical properties and design scalable numerical resilience algorithms. In this dissertation, we study a new class of numerical resilience algorithms at application level that do not require extra resources, i.e., computational units or computational time, when no fault occurs. Assuming that a separate mechanism in the software stack ensures fault detection, we propose numerical algorithms called Interpolation-Restart (IR) techniques, to survive to faults.

In Part I, we design IR techniques for Krylov subspace linear solvers and well-known eigensolvers. In the framework of Krylov subspace linear solvers in Chapter 2, the lost entries of the iterate are interpolated using the available entries on the still alive nodes to define a new initial guess before restarting the Krylov method. In particular, we consider two rational policies that preserve key numerical properties of well-known linear solvers, namely the monotony decrease of the A-norm of the error of the conjugate gradient or the residual norm decrease of GMRES. We assess the impact of the interpolation-restart techniques, the fault rate and the amount of lost data on the robustness of the resulting resilient Krylov subspace solvers.

Chapter 3 is dedicated to IR techniques for iterative eigensolvers where there are more freedom to adapt Interpolation-Restart ideas. For instance, for the Jacobi-Davidson solver, the interpolations are applied to the converged Schur vectors as well as to the best direction candidates of the current search space. After a fault, this new set of vectors are used as initial guess to restart the Jacobi-Davidson iterations. Most of the commonly used eigensolvers such as Arnoldi, Implicitly restarted Arnoldi or subspace iteration algorithm have been revisited in the light on faults. We illustrate the robustness of the proposed schemes through extensive numerical experiments.

In Part II we consider an actual fully-featured sparse hybrid (direct/iterative), namely the Massively Parallel Hybrid Solver (MaPHyS[1]) [4, 75, 83]. This solver is based on non-overlapping domain decomposition techniques applied in a fully algebraic framework. Such a technique leads to the iterative solution of a considered linear system defined on the interface between the algebraic subdomains (subset of equations). We derive this parallel solver for making it resilient, focusing on the iterative solve step, objective of this thesis. The derivation is twofold and aims at exploiting the properties of the particular hybrid method considered. First, we show that data redundancy can be efficiently used to recover single fault cases. Second, we propose a variant of the IR strategies proposed in Part I and tuned according to the properties of the preconditioner onto which the hybrid solver relies. We show that this IR strategy is efficient to survive neighbor processes fault cases. To illustrate our discussion, we have modified the fully-featured MaPHyS solver. We focused on the cost induced at algorithm level (extra computation and communication) and neglected the cost due to the necessary system software stack required for supporting it.

Finally we conclude this manuscript with some perspectives for future research in the field of resilient numerical schemes.

[1] https://project.inria.fr/maphys/

Chapter 1

General Introduction

Contents

Resilience is accepting your new reality, even if it's less good than the one you had before. You can fight it, you can do nothing but scream about what you've lost, or you can accept that and try to put together something that's good

Elizabeth Edwards

1.1 Introduction

The main objective of this chapter is to introduce numerical linear algebra problems and review a large spectrum of known fault tolerance researches in the HPC community. Consequently, this chapter does not focus on numerical fault tolerance approaches which are the target of this thesis and introduced further in Chapter 2. It presents a more general framework which explores most of fault researches in HPC community but is not exhaustive.

The remainder of the chapter is structured as follows: Section 1.2 is a brief introduction to numerical linear algebra problems. It explains the limitation of direct methods for the solution of large sparse linear algebra problems namely linear systems problems and eigenvalue problems. Alternatively, it introduces iterative numerical methods, more precisely, Krylov subspace methods which have attractive properties for fault tolerance. Section 1.3 aims at presenting fault issues in the HPC systems. It mainly present fault nomenclature in the HPC community, and emphasizes on the origin of the permanent increase of fault rate in HPC systems before presenting fault injection protocols commonly used in fault tolerance researches. In Section 1.4, we present different fault tolerant approaches before giving a short summary of concepts covered in this chapter in Section 1.5.

1.2 Brief introduction to numerical linear algebra

Numerical linear algebra plays a central role in solving many real-world problems. To understand phenomena or to solve problems, scientists use mathematical models. Solutions obtained from mathematical models are often satisfactory solutions to complex problems in field such as weather prediction, trajectory of a spacecraft, car crashes simulation, etc. In many scientific fields such as electrostatics, electrodynamics, fluid flow, elasticity, or quantum mechanics, problems are broadly modeled by partial differential equations (PDEs). The common way to solve PDEs is to approximate the solution which is continuous by discrete equations that involve a finite but often large number of unknowns [136, Chapter 2]. This strategy is called discretization. There are many ways to discretize a PDE, the three most widely used being the finite element method (FEM), finite volume methods (FVM) and finite difference methods (FDM). These discretization strategies lead to large and sparse matrices. Thus real-word problems become numerical linear algebra problems. There are

many linear algebra problems, but this work focuses on the resolution of linear systems of equations $\mathcal{A}x = b$ and on eigenvalue problems $\mathcal{A}x = \lambda x$.

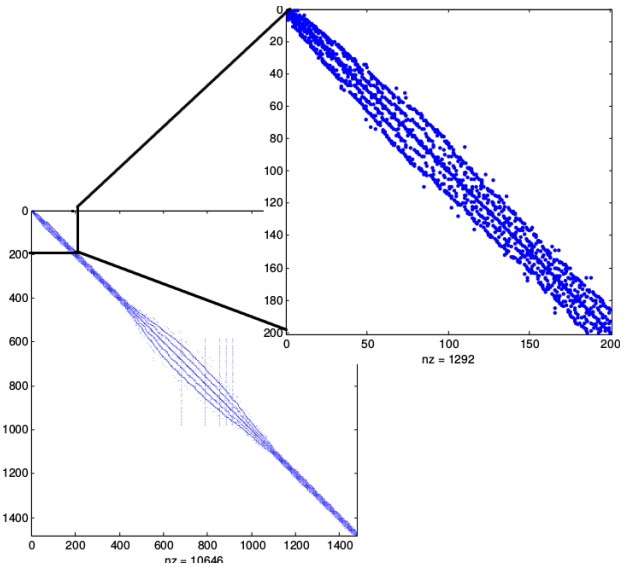


Figure 1.1 – Sparse matrix `OP` from modelization of thermoacoustic problems.

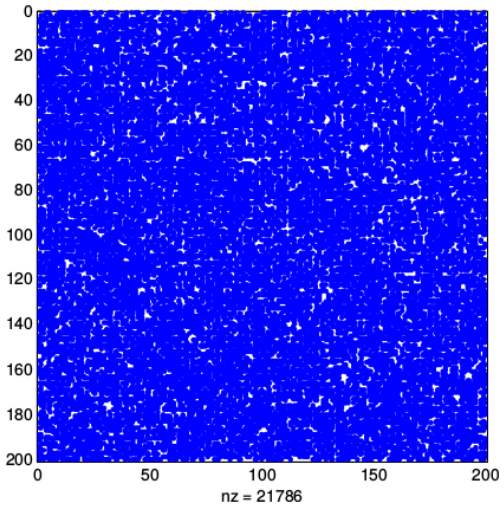
For example, the sparse matrix `OP` (Figure 1.1) from the PhD of P. Salas [142] is obtained by modeling combustion instabilities in complex combustion chambers. More precisely, the matrix is obtained by discretizing the Helmholtz equation using a finite volume formulation. The computation of eigenvalues and eigenvectors of the sparse matrix `OP`, which is a pure numerical linear algebra problem, allows one to study combustion instabilities problems.

1.2.1 Sparse matrices

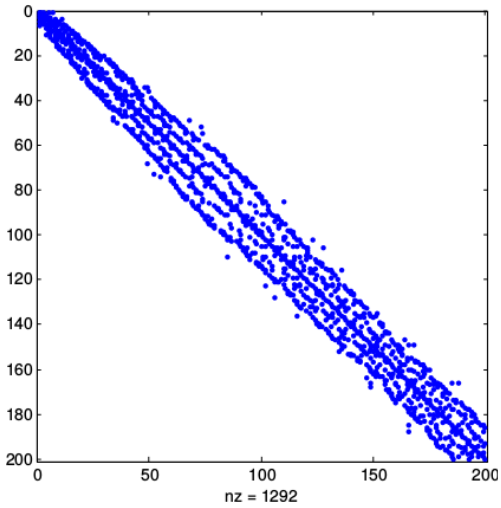
As reported for example by Yousef Saad [136, Chapter 2], partial differential equations are among the most important sources of sparse matrices. A matrix is said to be sparse if it contains only very few nonzero elements, as depicted in Figure 1.2b, where nonzero elements are represented in blue color. There is no accurate definition of the proportion of nonzero elements in sparse matrices. However, a matrix can be considered as sparse when one can take advantage computationally of taking into account its nonzero elements. Even if the matrix presented in Figure 1.2a contains 54% of nonzero elements it cannot be termed sparse. Matrices from numerical simulations are not only sparse, but they may also be very large, which leads to a storage problem. For example, a matrix $\mathcal{A} \in \mathbb{C}^{n \times n}$, of order $n = 10^6$, contains $n \times n = 10^{12}$ elements (zero and nonzero elements). In double precision arithmetic, 16 terabytes[1] are necessary to store all its entries. There are special data structures to store sparse matrices and the basic idea is to store only nonzero elements.

The main goal of these data structures is to store only nonzero elements and at the same time facilitate sparse matrix operations. The most general sparse matrix storage is called

[1] $10^{12} \times 2 \times 8$ bytes $= 16 \times 10^{12}$ bytes. Each complex element requires 2×8 bytes, 8 bytes for imaginary part and 8 for real part, in double precision



(a) This matrix contains 54% of nonzero elements



(b) This matrix contains 3% of nonzero elements

Figure 1.2 – Sparse matrices contains only a very few percentage of nonzero elements. With 54% of nonzero elements the matrix in (a) cannot be referred as sparse whereas, with only 3% of non zero elements, the matrix in (b) satisfies a sparsity criterion.

coordinate (COO) format and consists of three arrays of size nnz, where nnz is the number of nonzero elements. As illustrated in Fig 1.3, the first array (`AA`) contains the nonzero elements of the sparse matrix, the second array (`JR`) contains the corresponding row indices and the third array (`JC`) contains the corresponding column indices.

$$A = \begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0 & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix}$$

AA	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.
JR	1	1	2	2	2	3	3	3	3	4	4	5
JC	1	4	1	2	4	1	3	4	5	3	4	5

Figure 1.3 – Coordinate (COO) format for sparse matrix representation.

The COO format is very flexible but possibly not optimized since row indices and column indices may be stored redundantly. In the example depicted in Fig 1.3, the row index "3", is stored 4 times in `JR`, and the column index "4" is also stored 4 times. It is possible to compress row indices, which leads to compressed sparse row (CSR) format. Alternatively the column indices can also be compressed, this format is called compressed sparse column (CSC). Other sparse data structures exist to further exploit particular situations. We refer to [136, Chapter 2] for a detailed description of possible data structures for sparse matrices.

1.2.2 Solutions for large sparse linear algebra problems

To illustrate problems related to large sparse linear algebra problems, we take the particular case of a linear system of equations. To solve a linear system of equations of form $\mathcal{A}x = b$, where $\mathcal{A}$ is a square non-singular matrix of order n, b, the right-hand side vector, and x the unknown vector, as illustrated by Equation (1.1), Gaussian elimination is broadly used,

$$\begin{pmatrix} 1. & 0. & 0. & 2. & 0. \\ 3. & 4. & 0. & 5. & 0. \\ 6. & 0 & 7. & 8. & 9. \\ 0. & 0. & 10. & 11. & 0. \\ 0. & 0. & 0. & 0. & 12. \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 5 \\ 4 \\ 3 \\ 2 \\ 1 \end{pmatrix}. \tag{1.1}$$

One variant decomposes the coefficient matrix of the linear system (here A) into a product of a lower triangular matrix L (diagonal elements of L are unity) and an upper triangular matrix U such that A=LU. This decomposition is called LU factorization.

$$L = \begin{pmatrix} 1.00 & 0 & 0 & 0 & 0 \\ 3.00 & 1.00 & 0 & 0 & 0 \\ 0 & 1.50 & 1.00 & 0 & \\ 0 & 0 & 0.56 & 1.00 & 0 \\ 0 & 0 & 0 & 0.77 & 1.00 \end{pmatrix} \quad U = \begin{pmatrix} 1.00 & 0 & 2.00 & 0 & 0 \\ 0 & 4.00 & -6.00 & 5.00 & 0 \\ 0 & 0 & 16.00 & 14.21 & -4.50 \\ 0 & 0 & 0 & -7.50 & 8.00 \\ 0 & 0 & 0 & 0 & 15.48 \end{pmatrix}.$$

Once the LU factorization is performed, the linear system solution consists of two steps:

1: *The forward substitution* solves the triangular systems $Ly = b$.
In our example, it computes $y = (5.00, -11.00, 19.50, -8.96, 7.93)^T$.

2: *The backward substitution* solve $Ux = y$,
which leads to the solution $x = (3.51, -1.05, .074, -0.46, 0.51)^T$.

The advantage of this approach is that most of the work is performed in the decomposition ($\mathcal{O}(n^3)$ for dense matrices) and very little in the forward and backward substitutions ($\mathcal{O}(n^2)$). The solution of successive linear systems using the same matrix but with different right-hand sides, often arising in practice, is then relatively cheap. Furthermore, if the matrix is symmetric (or SPD), an LDL^T (or Cholesky) factorization may be performed. In finite arithmetics, direct methods enable one to solve linear systems in practice with a high accuracy in terms of backward error [88]. However, this numerical robustness has a cost. First, the number of arithmetic operations is very large. Second, in the case of a sparse matrix $\mathcal{A}$, the number of nonzeros of L and U is often much larger than the number of nonzeros in the original matrix. This phenomenon, so-called fill-in, may be prohibitive in terms of memory usage and computational time. Solving large sparse linear algebra problems using direct methods is very challenging because of memory limitation.

In order to minimize computational cost and guarantee a stable decomposition and limited fill-in, some direct sparse linear systems solvers have been implemented such as `CHOLMOD` [39], `MUMPS` [8, 9], `PARDISO` [144], `PaStiX` [85], `SuperLU` [110], to name a few. These direct sparse methods work well for 2D PDE discretizations, but they may be very penalizing in terms of memory usage and computational time, especially for 3D test cases.

To solve very large sparse problems, iterative solvers are more scalable and better suited for parallel computing. Iterative methods produce a sequence of approximates to the solution. Successive iterations implemented by iterative methods require a small amount of storage and floating point operations, but might converge slowly or not converge at all. It is important to notice that iterative methods are generally less robust than direct solvers for general sparse linear systems. There are many variants of iterative methods for both linear system of equations and eigenvalues problems.

1.2.3 Iterative methods for linear systems of equations

Iterative methods for linear systems are broadly classified into two main types: stationary and Krylov subspace methods.

1.2.3.1 Stationary methods for solving linear systems

Consider solving the linear system $\mathcal{A}x = b$, stationary methods can be expressed in the simple form

$$Mx^{(k+1)} = Nx^{(k)} + b, \tag{1.2}$$

where $x^{(k)}$ is the approximate solution at the k^{th} iteration. The matrices N and M do not depend on k, and satisfy $\mathcal{A} = M - N$ with M non singular. These methods are called stationary because the solution to a linear system is expressed as finding the stationary fixed point of Equation (1.2) when k will go to infinity. Given any initial guess $x^{(0)}$, the stationary method described in Equation (1.2) converges if and only if $\rho(M^{-1}N) < 1$. Note that the spectral radius $\rho(A)$ of a given matrix $\mathcal{A}$ with eigenvalues λ_i is defined by $\rho(\mathcal{A}) = max(|\lambda_i|)$. Typical iterative methods for linear systems are Gauss-Seidel, Jacobi,

M	N	Method
D	$(L+U)$	Jacobi
$(D-L)$	U	Gauss-Seidel
$((\frac{1}{\omega})D-L)$	$(((\frac{1}{\omega})-1)D+U)$	Successive over relaxation

Table 1.1 – Stationary iterative methods for linear systems. D, $-L$ and $-U$ are the diagonal, strictly lower-triangular and strictly upper-triangular parts of $\mathcal{A}$, respectively.

successive over relaxation etc., as described in Table 1.1 according to the choice of M and N.

1.2.3.2 Krylov subspaces

Another approach to solve linear systems of equations consists in extracting the approximate solution from a subspace of dimension much smaller than the size of the coefficient matrix $\mathcal{A}$. This approach is called projection method. These methods are based on projection processes: orthogonal and oblique projection onto Krylov subspaces, which are subspaces

spanned by vectors of form $p(\mathcal{A})v$, where p is a polynomial [136]. Let $\mathcal{A} \in \mathbb{C}^{n\times n}$ and $v \in \mathbb{C}^n$, let $m \leq n$, the space denoted by $\mathcal{K}_m(A, v) = Span\{v, Av, ..., A^{m-1}v\}$ is referred to as the Krylov space of dimension m associated with $\mathcal{A}$ and v. In order words, these techniques approximate $\mathcal{A}^{-1}v$ by $p(\mathcal{A})v$, where p is a specific polynomial. Based on a minimal polynomial argument, it can be shown that these methods should converge in less than n steps compared to *"infinity"* for stationary schemes.

In practice, minimal residual methods are based on orthogonal basis using Arnoldi procedure depicted in Algorithm 1

Algorithm 1 Arnoldi procedure to build orthogonal basis of Krylov subspace $\mathcal{K}_m(A, m)$.

1: $v_1 = \frac{v}{\|v\|_2}$
2: **for** $j = 1, \ldots, m$ **do**
3: $\quad w_j = Av_j$
4: $\quad$ **for** $i = 1$ `to` j **do**
5: $\quad\quad h_{i,j} = v_i^T w_j$; $w_j = w_j - h_{i,j}v_i$
6: $\quad$ **end for**
7: $\quad h_{j+1,j} = \|w_j\|_2$
8: $\quad v_{j+1} = w_j/h_{j+1,j}$
9: **end for**

We denote $V_m \in \mathbb{C}^{n\times m}$ the matrix with column vectors $v_1, \ldots, v_m$ and H_m the $m \times m$ Hessenberg matrix whose nonzero entries are defined by Algorithm 1. The following equalities are satisfied:

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T, \tag{1.3}$$

$$V_m^T A V_m = H_m. \tag{1.4}$$

The Hessenberg matrix H_m is the projection of $\mathcal{A}$ onto $\mathcal{K}_m(\mathcal{A}, v)$, with respect to the orthogonal basis V_m [136]. If $\mathcal{A}$ is a symmetric matrix, then H_m is reduced to a tridiagonal symmetric matrix and the corresponding algorithm is called Lanczos algorithm. The main idea of Krylov subspace methods is to project the original problem onto the Krylov subspace and then solve the problem in that Krylov subspace.

1.2.3.3 Krylov subspace methods for linear systems

Krylov subspace methods are currently widely used iterative techniques for solving large sparse linear systems. Given an initial guess $x^{(0)}$, to approximate the linear systems of equations $\mathcal{A}x = b$, Krylov subspace methods approximate the solution $x^{(m)}$ from the subspace $x^{(0)} + \mathcal{K}_m(A, r^{(0)})$, where $r^{(0)} = b - Ax^{(0)}$. There are different variants of Krylov subspace methods for linear systems of equations. The Ritz-Galerkin approach constructs $x^{(m)}$ such that $b - Ax^{(m)} \perp \mathcal{K}_m(A, r^{(0)})$. The CG algorithm belongs to this class. The Petrov-Galerkin approach constructs $x^{(m)}$ such that $b - Ax^{(m)} \perp \mathcal{L}_m$, where $\mathcal{L}_m$ is another subspace of dimension m. If $\mathcal{L}_m = A\mathcal{K}_m(A, r^{(0)})$, then $x^{(m)}$ minimizes the residual norm $\|b - Ax^{(m)}\|_2$ over

all candidates from the Krylov subspace [140]. A typical example is the GMRES algorithm. The biconjugate gradient method (BiCG) is obtained when $\mathcal{L}_m = \mathcal{K}_m(A^T, r^{(0)})$.

The convergence of Krylov subspace methods depends on the numerical properties of the coefficient matrix $\mathcal{A}$. To accelerate the convergence, one may use a non-singular matrix M such that $M^{-1}\mathcal{A}$ has *better* convergence properties for the selected solver. The linear systems $M^{-1}Ax = M^{-1}b$, has the same solution as the original linear system. This method is called preconditioning and the matrix M is called an implicit (i.e., M attempts to somehow approximate $\mathcal{A}$) left preconditioner. On the other hand, linear systems of equations can also be preconditioned from the right: $\mathcal{A}M^{-1}y = b$, and $x = M^{-1}y$. One can also consider split preconditioning that is expressed as follows: $M_1^{-1}AM_2^{-1}y = M_1^{-1}b$, and $x = M_2^{-1}y$, where the preconditioner $M = M_1M_2$. It important to notice that Krylov subspace methods do not compute explicitly the matrices $M^{-1}\mathcal{A}$ and $\mathcal{A}M^{-1}$, in order to minimize computational cost and to preserve sparsity. Preconditioner are commonly applied by performing sparse matrix-vector product or solving simple linear systems.

1.2.3.4 Stopping criterion for convergence detection

As mentioned above, iterative methods construct a series of approximate solutions that converges to the exact solution of the linear system. The *normwise* perturbation model [163] is an appropriated method that can be used to check if the approximation is good enough to stop the iterations. Let $x^{(k)}$ be an approximate solution of the linear system $\mathcal{A}x = b$. The quantity defined by Equation (1.5) is the backward error associated with $x^{(k)}$:

$$\begin{aligned}\eta_{A,b}(x^{(k)}) &= \min_{\Delta A, \Delta b}\{\tau > 0 : \|\Delta A\| \leq \tau\|A\|, \|\Delta b\| \leq \tau\|b\| \text{ and } (A+\Delta A)x^{(k)} = b + \Delta b\}, \\ &= \frac{\|Ax^{(k)} - b\|}{\|A\|\|x^{(k)}\| + \|b\|}.\end{aligned} \tag{1.5}$$

This quantity is used to define stopping criteria for iterative methods. It measures the norm of the smallest perturbations ΔA on $\mathcal{A}$ and Δb on b such that $x^{(k)}$ is the exact solution of $(A+\Delta A)x^{(k)} = b+\Delta b$. The lower the backward error the better, ideally the backward error is expected to be of the order of the machine precision. In practice, computing $\|\mathcal{A}\|$ might not be feasible or too expensive. Alternatively, one can consider another perturbation model that only involves perturbations in the right-hand side to define the following backward error:

$$\begin{aligned}\eta_b(x^{(k)}) &= \min_{\Delta b}\{\tau > 0 : \|\Delta b\| \leq \tau\|b\| \text{ and } Ax^{(k)} = b + \Delta b\}, \\ &= \frac{\|Ax^{(k)} - b\|}{\|b\|}.\end{aligned} \tag{1.6}$$

The backward error defined by Equation (1.6) is the stopping criterion commonly used for iterative methods. Given a target accuracy ϵ, the algorithm finishes with success if the current approximation $x^{(k)}$ satisfies the criterion $\eta_b(x^{(k)}) \leq \epsilon$.

We notice that for GMRES, the location of the preconditioner has an impact on the backward error that is commonly checked during the computation. This backward error is depicted in Table 1.2. It can be seen that a right preconditioning is interesting because for right preconditioning, the backward error is the same for both the preconditioned and original system. Indeed, $\|AMy^{(k)} - b\| = \|Ax^{(k)} - b\|$ because $x^{(k)} = My^{(k)}$.

Preconditioning	Left precond.	Right precond.	Split precond.
Backward error	$\frac{\|MAx^{(k)}-Mb\|}{\|Mb\|}$	$\frac{\|AMy^{(k)}-b\|}{\|b\|}$	$\frac{\|M_2Ay^{(k)}M_1-M_2b\|}{\|M_2b\|}$

Table 1.2 – Backward errors associated with preconditioned linear systems in GMRES.

1.2.4 Iterative methods for eigenvalue problems

1.2.4.1 Basic definition of eigenvalue problems

The eigenvalue problem for a matrix $\mathcal{A}$ consists in the determination of nontrivial solution (i.e., $u \neq 0$) of $\mathcal{A}u = \lambda u$, where λ is an eigenvalue of $\mathcal{A}$, u is the corresponding eigenvector, and (λ, u) is called an eigenpair. In some way, one can say that all the algorithms for the computation of eigenvalues of large matrices are iterative methods. We distinguish on one hand, algorithms for the computation of few eigenvalues like power method, subspace iteration and its variants described in Section 3.3. On the other hand, there are algorithms for the computation of all eigenvalues like similarity transformations. The full set $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ of eigenvalues of A is called the spectrum of $\mathcal{A}$ and denoted $\sigma(\mathcal{A})$. It is important to notice that the spectrum of $\mathcal{A}$ is invariant by similarity transformation. this means that if $B = G^{-1}\mathcal{A}G$, then $\sigma(A) = \sigma(B)$ [141].

The basic scheme to compute all the eigenpairs of matrices is to transform them into matrices that have simpler forms, such as diagonal or bidiagonal, or triangular etc.

If a matrix $\mathcal{A}$ is similar to a diagonal matrix, then $\mathcal{A}$ is diagonalizable and we have $\mathcal{A} = GDG^{-1}$, where $G = [g_1, g_2, \ldots, g_n]$ if formed by the eigenvectors of $\mathcal{A}$, and $D = diag(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is a diagonal matrix formed by the corresponding eigenvalues. Alternatively, another decomposition can be considered that is the Schur form of a matrix $\mathcal{A} = Z^H T Z$, where T is upper triangular and Z is unitary. The diagonal entries of T are the eigenvalues of $\mathcal{A}$. The columns of Z are called the Schur vectors. This decomposition is often used on projected problem is attractive because it involves unitary transformations that are stable in finite precision.

1.2.4.2 Krylov subspace methods for eigenvalue problems

Krylov subspace methods are also successful for the solution of large sparse eigenvalue problems. The basic idea is to compute eigenvalues using similarity transformation processes [138]. Given a Krylov subspace $\mathcal{K}$, spanned by orthonormal vectors $V_m = [v_1, v_2 \ldots, v_m]$, a projection process onto $\mathcal{K}$ computes an approximate eigenpair $(\tilde{\lambda} \in \mathbb{C}, \tilde{u} \in \mathcal{K})$ that satisfies the Galerkin condition, $(A - \tilde{\lambda}\mathcal{I})\tilde{u} \perp \mathcal{K}$ [139]. The approximate

eigenvalues $\tilde{\lambda}_i$ are the eigenvalues of $C \in \mathbb{C}^{m\times m}$, where $C = V_m^T A V_m$. The corresponding approximate eigenvectors are $\tilde{u_i} = V y_i$, where y_i are the eigenvectors of C. Well-known Krylov subspace methods for eigenvalue problems are the Hermitian Lanczos method, the Arnoldi method and the nonhermitian Lanczos method. Both Arnoldi method and Lanczos method are based on orthogonal projection methods whereas the nonsymmetric Lanczos algorithm is an oblique projection method [141]. Furthermore, Newton type schemes can be considered as well. This leads to the well-known and effective Jacobi-Davidson class of methods.

1.2.4.3 Stopping criterion for convergence detection

Following the same philosophy as for linear system solution, backward error can be considered for eigenproblems. An eigenpair $(\lambda_k, u^{(k)})$ computed by an eigensolver can be considered as the exact eigenpair of a nearby perturbed problem. For $\|u^{(k)}\| = 1$, the backward error reads [164]

$$\begin{aligned} \eta_A(\lambda_k, u^{(k)}) &= \min_{\Delta A}\{\tau > 0 : \|\Delta A\| \leq \tau \|A\| \text{ and } (A + \Delta A)u^{(k)} = \lambda_k u^{(k)}\}, \\ &= \frac{\|Au^{(k)} - \lambda_k u^{(k)}\|}{\|A\|}. \end{aligned} \tag{1.7}$$

In practice, computing $\|A\|$ can be prohibitive in term of computation cost. An often used alternative is to consider an upper bound of the backward error that is built using the following inequalities

$$|\lambda_k| \leq \rho(A) \leq \|A\|$$

where $\rho(A)$ denotes the spectral radius of A. The corresponding stopping criterion is $\frac{\|Au^{(k)} - \lambda_k u^{(k)}\|}{|\lambda_k|} \leq \epsilon,$, if $\lambda \neq 0$, that ensures $\eta_A(\lambda_k, u^{(k)}) < \epsilon$. In this dissertation, we use this stopping criterion that is widely considered in many libraries for the solution of large eigenproblems, such as ARPACK [104] or SLEPC[2]. For more details, a study on stopping criteria for general sparse matrices is reported in [103].

1.2.5 Parallel implementation of large sparse linear algebra solvers

Numerical algorithms for large sparse linear algebra solvers are based on few basic routines such as sparse matrix-vector multiplications (SpMV), dot products, vector norm, vector update functions, etc. These routines, except SpMV are part of Basic Linear Algebra Subroutines (BLAS) [16]. Thus, the parallelization of sparse Krylov solvers for linear systems and solvers for eigenvalue problems can be achieved by investigating how to execute BLAS on parallel distributed environments. Furthermore, preconditioning is often a potential bottleneck in Krylov subspace solvers for linear systems [136]. There is a large literature on preconditioning techniques that are very much problem dependent and their parallel

[2] http://www.grycap.upv.es/slepc/

implementation constrained by the targeted architecture. It is a challenge to find a preconditioner with favorable numerical properties for better convergence, and a good parallelism quality. This challenge is out of the scope of this work. In this work, we consider a block-row partition. For the sake of illustration, we give a brief description of SpMV ($y \leftarrow Ax$), vector update ($y(1:n) \leftarrow a \times x(1:n)$).

SpMV: Let p be the number of partitions, such that each block-row is mapped to a computing node. For all i, $i \in [1, p]$, I_i denotes the set of rows mapped to node i. With respect to this notation, node i stores the block-row $\mathcal{A}_{I_i,:}$ and x_{I_i} as well as the entries of all the vectors involved in the Krylov solver associated with the corresponding row indices of this block-row. If the block $\mathcal{A}_{I_i,I_j}$ contains at least one nonzero entry, node j is referred to as neighbor of node i as communication will occur between those two nodes to perform a parallel matrix-vector product. As illustrated in Figure 1.4 node 1 and node 3 are neighbors so they will exchange data during the computation, whereas node 1 and node 2 are not neighbors.

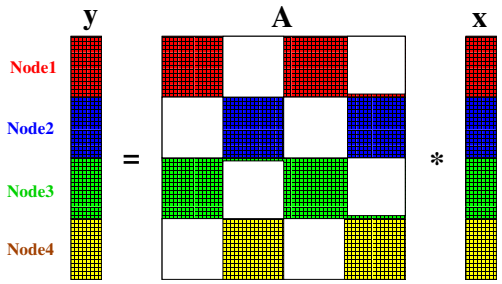


Figure 1.4 – Block-row distribution of sparse matrix-vector multiplication on 4 nodes.

Vector update: Vector updates are straightforward because they do not require communication with neighbors. Since vectors x and y are distributed in the same manner in nodes, i.e, the indices of the components of any vector that are mapped to a given node are the same, each node $i \in [1, p]$ performs $y_{I_i} = y_{I_i} + a \times x_{I_i}$ simultaneously.

1.3 Quick introduction to faults in HPC systems

1.3.1 Understanding faults in HPC systems

In HPC systems, unexpected behaviors are often called faults, failures or errors [55]. A fault refers to an abnormal behavior of system components whereas a failure refers to an incorrect output of a routine [121]. However, all faults do not lead to an incorrect result, since error detection and correcting features can eliminate the impact of a fault. Some faults occur in a non-crucial part of the system. Some faults are not detected and repaired and cause incorrect behavior, which is called an error. Errors are manifested in a particular behavior called a failure. The most common way to express the difference between fault, failure and error is as follows: *a failure is the impact of a fault that caused an error.* Even

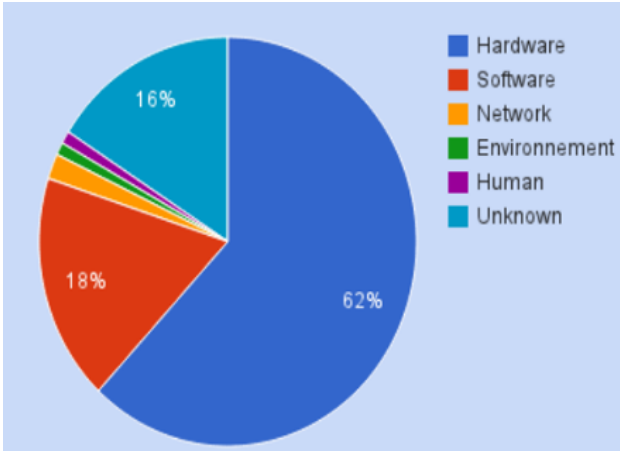


Figure 1.5 – Root causes of faults according to [147].

though, there is a difference between fault, failure and error, in the remainder if this work, we distinguish only between fault and failure, for the sake of simplicity.

Faults may also be classified in *soft* and *hard* faults, based on their impact on the system. A hard fault, also called fail-stop is a fault that causes immediate routine interruption. On the other hand, a soft fault is an inconsistency that does not lead *directly* to routine interruption. Typical soft faults are: bit flip, data corruption [27]. Soft faults are likely to induce hard faults or failures. For example, if a processor memory becomes defective due to high temperature, it may experiment bit corruption without permanent physical defect; it is a transient fault. If the memory continues functioning but induces bit corruption more frequently at the slightest increase in temperature, it is an intermittent fault. In the worst case, the memory may have physical damage which may lead to interrupt or series of data corruption, it is a persistent fault.

One of the disturbing issues when studying faults in HPC systems is the increase of fault frequency, because common sense expect less faults with technology advance. In HPC systems, performance is currently improved by the use of largest number of components. Hardware becomes more complex, heterogeneous [152], while circuit size and voltage shrink. The reduction of circuit size and voltage increases thermal neutrons and alpha particle emission, which perturbs the circuits [56]. However, circuit size and voltage are not the only source of fault in devices. In [118], studies on memory chips show that cosmic rays also represent a significant source of faults.

According to the studies reported in [147], faults may emanate from many sources such as hardware, software, network, human and environment as depicted in Figure 1.5. Among those faults, hardware faults are predominant and more detailed studies [72, 148] showed that faults related to memory represent the most significant part of hardware faults in modern HPC systems.

Even though it is commonly accepted that HPC systems face higher fault rates, HPC system fault logs are hardly accessible while they are necessary for fault studies. To face this issue, Bianca Schroeder and Garth Gibson, have created the computer failure data

repository (CFDR)[3] and encourage HPC labs and end-users to report faults that occur on their systems [146]. Nowadays, the repository contains fault data of 4750 nodes and 24101 processors, and represents an interesting database for a fault study.

The basic metric to analyze the sensitivity to system fault is the availability. The availability is the percentage of time a system is really functioning correctly. The amount of time of malfunctioning includes also the time to repair the system. This metric gives a general insight into the availability of the whole system, but it does not give information on the frequency of faults. The metric commonly used is the mean time between faults (MTBF)[2] [94]. It corresponds to the average time between two consecutive detected faults assuming that the system is repaired instantly. Generally speaking, the MTBF of a system is the average amount of time that it can work without any fault report. Here, it is important to note that the MTBF focuses only on detected faults. Given the MTBF θ of a system, $\lambda = \frac{1}{\theta}$, is called the fault rate. It is the frequency of faults, and it is expressed in faults per unit of time. In [147], Schroeder and Gibson analyzed the fault rate of 22 large-scale parallel systems. Different fault rates have been observed and they range from 17 to 1159 faults per year, meaning a MTBF of 8 hours in the worst case. This result shows that the fault rate varies largely across systems. The study has also confirmed two interesting insights as follows. The fault rate increases linearly with the size of systems and varies also across nodes. The results are consistent with recent forecasts that expect the MTBF of extreme-scale systems to be less than one hour [15].

1.3.2 Fault distribution models

The fault rate and the MTBF of systems are good metrics to give insights on the frequency of faults, but they are still a statistical average, with less information on the distribution of faults across systems lifetime. Faults seem to occur randomly and independently, but they may fit well known probabilistic models. If the fault rate is constant over time, it is an exponential distribution with the following cumulative distribution function (CDF):

$$F(t) = \begin{cases} 1 - e^{-\lambda t} & \text{if } t \geq 0, \\ 0 & \text{if } t < 0. \end{cases} \tag{1.8}$$

We recall that λ is the fault rate, while t is the operating time. In the case of an exponential distribution we have: $MTBF = \frac{1}{\lambda}$. However, fault analysis on systems shows that, the fault rate varies over the lifetime of the same system. The lifetime of a system can be divided into three periods as follows. The first period corresponds to the infant mortality. This period is often chaotic, faults are more frequent and even occur randomly but things improve progressively. The second period is the useful lifetime, faults are less frequent and the fault rate stabilizes. At the end, the system experiments the wear out period where the fault rate increases again. Schroeder and Gibson [147], and Raju et al. [132] showed that the Weibull distribution [162] is the most realistic probabilistic model that characterizes the normal

[3] https://www.usenix.org/cfdr

[2] MTBF often stands for Mean Time Between Failures, but this definition would not be consistent with our distinction between faults and failures.

behavior of large-scale computational platforms. The CDF of the Weibull distribution is:

$$F(t) = \begin{cases} 1 - e^{-(\frac{t}{c})^k} & \text{if } t \geq 0, \\ 0 & \text{if } t < 0. \end{cases} \tag{1.9}$$

Here, c is the scale parameter and k (with $k > 0$) is the shape parameter. In probability theory and statistics, a scale parameter allows one to set the scale of a distribution. A large scale parameter corresponds to a relatively spread out distribution whereas a small scale parameter corresponds to a more condensed distribution. The shape parameter is related to the variation of the fault rate. A value of $k < 1$ implies that the fault rate decreases, it is the infant mortality period. With $k = 1$, we fall back to an exponential distribution, it is the useful lifetime. Finally, a value of $k > 1$ means that the fault rate increases over time, the wear out period. In this particular case of a Weibull distribution, the $MTBF$ is computed as follows [7]: $MTBF(c, k) = c\Gamma(1 + \frac{1}{k})$, where Γ is the gamma function:

$$\Gamma(x) = \begin{cases} (x-1)! & \text{if } x \in \mathbb{N}, \\ \int_0^{+\infty} t^{x-1} e^{-t}\, dt & \text{if } x \notin \mathbb{N}. \end{cases}$$

There are also different models that do not describe necessarily the fault distribution but help for fault prediction. In [72, 81, 111] the authors investigate algorithms to analyze correlations between reported faults in order to forecast future faults. These fault prediction models, based on system log analysis, can be very useful to implement appropriate fault tolerant mechanisms. Most of the fault prediction models help to determine fault occurrence in the whole system, but they cannot specify the faulty component [72]. For efficient fault management, the end-user needs insights on the fault occurrence on each component involved in a computation. Studies reported in [77,132,147,165] conclude that faults in each individual node fit Weibull distribution, but researchers do not agree on fault correlation among nodes. Fault propagation between nodes is highly related to the system topology and the robustness of the architecture. Schroeder and Gibson [147] report very strong correlation between nodes, only in the first lifetime of systems, while no evident correlation is observed after that period. Depending on the target system, some studies [23,77,132] use the simple model without correlation while other design fault correlation models [72,129].

In practice, it is very difficult to design a reliable fault prediction system. However, fault distribution models are useful to optimize fault tolerance algorithms. They may also be useful to simulate fault injection in HPC systems in order to evaluate fault tolerance strategies.

1.3.3 Fault injection models

Fault injection protocols are commonly designed for better understanding of fault impact on system behaviors, to reproduce faults and validate fault tolerance strategies [10, 96]. The fault injection protocols vary according to the target component (software, network, memory, CPU, node, etc.). Three main classes of fault injection protocol can be considered. These are: hardware fault injection, software fault injection and the simulation approach [34].

Hardware fault injection protocol: This approach lies at hardware level, and consists in physically changing the normal behavior of the hardware. Typical hardware fault injection are heavy-ion radiation injection into integrated circuits [96], voltage and temperature manipulation [98]. Hardware fault injection is useful to test hardware resiliency, however, its implementation may require specialized resources, without guarantee of reproducibility.

Software fault injection protocol: It consists of simulating faults using fault injection libraries. It is often achieved by changing the computational state or modifying the computational data of applications. For example, libfiu [13] is a fault injection software for testing the behavior of resilient applications without having to modify the application's source code. Linux, one of the most used operating system in HPC systems, provides also fault injection mechanisms. The fault injection module is available in Linux kernel since the version 2.6.20 [1]. It allows to simulate faults in memory slab, memory page and disk I/O. The fault injection tool is very flexible allowing users to define the fault rate as well as the maximal number of faults authorized during the simulation. This module is mainly implemented for fault injection in kernels. However, it can be used for the simulation of memory page corruption in distributed systems as reported by Naughton et al. [117].

Simulation-based fault injection: This approach is often used at application level, by directly modifying the source code of applications to simulate a faulty behavior. For example, one can overwrite computation data to simulate data corruption or reallocate memory pages to simulate memory page corruptions. To simulate process crashes, one can simply reduce the number of processes involved in a parallel computation. For example, in Unix/Linux, each process has its own address space through the memory management unit (MMU), which manages the mapping of virtual memory addresses and physical addresses. The Linux system call `mprotect()` [3] allows processes to specify the desired protection for the memory pages. The prototype of `mprotect()` is

```
int mprotect(const void *addr, size_t len, int prot);
```

where `[addr,addr+len]` is the address of the target and the values of `prot` are described in Table 1.3. Fault simulation may be achieved by invalidating a given set of memory pages with `prot = PROT_NONE` or just changing readable memory to unreadable with `prot = PROT_WRITE`. It is important to mention that `mprotect()` only changes the virtual address space. As a consequence, depending on the fault tolerance strategy, the process may recover the invalidated data and may request the same virtual address thanks to memory allocation mechanisms, more precisely using `mmap` [2]. This approach has been successfully used in [114] to simulate memory pages corruption in iterative solvers.

The simulation approach is more flexible and the most used in the HPC community.

Tag	Description
PROT_NONE	The memory cannot be accessed at all
PROT_READ	The memory can be read
PROT_WRITE	The memory can be written to
PROT_EXEC	The memory can contain executing code

Table 1.3 – Description of protection options of `mprotect`.

1.4 Overview of fault detection and correction models

1.4.1 Fault tolerance and resilience

The main two approaches to correct faults in systems are fault tolerance and resilience. These two concepts are commonly used in the HPC community to characterize the ability of a system or application to give a correct result despite faults. To avoid confusion we distinguish as follows. Fault tolerance is more related to the property of anticipating faults and providing rescue resources to deal with faulty components or data corruption. This is commonly achieved by component redundancy and data replication. On the other hand, resilience refers to the ability of keeping running trough a disaster to deliver a correct result. Resilience strategies enjoy the property of computing a result that satisfies the success criterion, but the result may differ from the fault-free one, whereas fault tolerance techniques are more likely to give the same result. Strategies are chosen depending on applications sensitivity. For example, transaction must be fault tolerant, whereas resilience may be satisfactory for scientific simulation applications. In the rest of this chapter, almost all the fault correction models from related work are fault tolerant models because they recover from fault thanks to data redundancy. Resilience approach based on interpolation techniques in numerical linear algebra solvers are briefly introduced in Section 1.4.7, and further detailed in Chapter 2.

1.4.2 Replication and redundancy

Replication is the most natural alternative among all fault tolerant techniques. The basic idea is to increase reliability by executing an entire application, or selected instructions on independent computational resources. When a fault occurs, the surviving computational resources will provide the correct output [23, 57]. As on HPC systems the soft faults are sometimes silent, duplication strategy is commonly used for fault detection [90, 161] by comparing the outputs, while triple modular redundancy (TMR) is used for fault detection and correction [145, 157]. However, the additional computational resources required by replication strategies may represent a severe penalty. Instead of replicating computational resources, studies [12, 156] propose a time redundancy model for fault detection. It consists in repeating computation twice on the same resource. The time redundancy model is more likely to detect transient faults but intermittent or persistent faults may remained undetected. The advantage of time redundancy models is the flexibility at application

level; software developers can indeed select only a set of critical instructions to protect. Recomputing only some instruction instead of the whole application lowers time redundancy overhead [119]. [116] proposes a selective replication strategy based on a combination of resource replication for critical instructions and time redundancy. Experimental results have demonstrated that the selective approach may reduce the overhead of duplication strategy by 59%.

The redundancy strategy is not only designed for computational resources, it is also useful to detect faults in storage units such as DRAM and cache memories or to detect faults during data transmission. A naive idea may consist in all bits replication, but the strategy commonly used consists in encoding redundant information in the parity bit [150] for single bit-flip detection. The Hamming code [84] is often used for double bit-flip detection and single bit-flip correction. Error correcting code (ECC) implemented in ECC memories, increases significantly memory reliability, however additional memory bits and memory checkers slow down the memory performance, and lead to the increase of the price of ECC memories and power consumption compared to classical memories. In [37] an interesting trade-off is proposed for the use of unreliable memories and ECC memories. They have proposed hierarchical memories with different levels of reliability together with an annotation-based model that allow users to store data according to the required protection. Thus critical data is stored in high reliability memory whereas classical memory stores non critical data.

The replication technique is very robust and most of its complexity is handled at the operating system level, making it very convenient for application developers. However, its extra usage of computational resources may be prohibitive in many real-life cases, especially for large scale simulations.

1.4.3 Checkpoint/restart techniques

Checkpoint/restart is the most studied protocol of fault tolerant strategies in the context of HPC systems. The common checkpoint/restart scheme consists in periodically saving data onto a reliable storage device such as a remote disk. When a fault occurs, a roll back is performed to the point of the most recent and consistent checkpoint.

Checkpoint/restart strategies can be classified into two categories: system-level checkpoint/restart and user-level checkpoint/restart. The system-level checkpoint/restart is also commonly known as kernel-level checkpoint/restart. It provides checkpoint/restart features without any change to applications and libraries. The main advantage of system-level checkpoint/restart strategies is that they have fine granularity access to the memories so they can optimize the data to checkpoint with a very low overhead. System-level checkpoint/restart does not take advantage of application properties because it aims at handling all applications. To design an efficient checkpoint/restart for a given application, user-level checkpoint/restart is commonly used. User-level checkpoint/restart consists in exploiting knowledge of the application to identify the best checkpoint frequency, the best consistent checkpoint state, the appropriate data to checkpoint, etc. User-level checkpoint/restart has to be at least partially managed by application programmers, and it can be external or internal. In an external user-level checkpoint/restart, a different process performs the

checkpoints, and checkpoint libraries are commonly used for that purpose. In an internal user-level checkpoint/restart, the process itself performs the checkpoint, so that the checkpoint/restart strategy becomes intrinsic to the application, it is also known as application-level checkpoint/restart.

1.4.3.1 Coordinated versus uncoordinated checkpoint

According to the implemented checkpoint strategy, all processes may perform the periodical record simultaneously. It is called a coordinated checkpoint. Coordinated checkpoint is widely used for fault tolerance because it is a very simple model. However, it presents some weaknesses. All the computation performed from the most recent checkpoint until the fault is lost. In addition, in parallel distributed environments, synchronizations due to coordination may significantly degrade application performance [63, 112]. Non-blocking coordinated checkpoints have been proposed [46, 62, 63] to limit synchronizations. The basic idea is to perform consistent checkpoint without blocking all the processes during the checkpoint. The checkpoint can be managed by a coordinator, and the active processes may delay the checkpoint and perform it as soon as possible without interrupting ongoing computation.

To fully avoid synchronization, uncoordinated checkpoint may be employed combined with message logging protocols. Message logging protocol consists in saving exchanged messages and their corresponding chronology on external storage [78]. When a process fails, it can be restarted from its initial state or from a recent checkpoint. The logged messages can be replayed in the same order to guarantee identical behavior as before the fault. The main challenge of message logging protocols is an efficient management of non-deterministic messages and best strategy to prevent inconsistencies. Inconsistencies currently occur if a process receives or sends a message and does not succeed to record it before it fails, creating thus an orphan process. A typical example from [6] is the following. Suppose a process p receives a message m_q from a process q, then sends a message m_p to the process q. If the process p crashes before it logs the message m_q, information about m_q would be lost. Suppose a new process p_{new} is spawned to replace p. While replaying the logged messages, p_{new} waits for the lost message m_q before sending m_p. In this case, p_{new} will never send m_p. The process q is thus called orphan process because it would have received a message that was not sent by the process p (p_{new}).

There are three classes of message logging protocols namely, optimistic, pessimistic, and causal protocols. In optimistic message logging protocol [48, 153], processes are not constrained to log exchanged messages before continuing communication, what may lead to orphan processes. The optimistic approach lowers the fault-free overhead, but in the presence of orphan processes, the overhead may be significantly high due to extra rollbacks to reach a consistent state. On the other hand, the pessimistic message logging protocol [20, 93] never creates orphan processes because the log protocol is performed synchronously on stable storage, and each process ensures that all received messages are logged before sending another message. The pessimistic approach may lead to high overhead and performance penalties due to synchronizations. A trade-off between the optimistic approach and the pessimistic approach is called causal message logging [6]. Causal message logging protocol prevents orphan processes and avoids some synchronizations [133].

1.4.3.2 Full versus incremental checkpoint

A checkpoint of a given process consists of saving the running state of the process including address space, register states, information on allocated resources. One well-known drawback of this approach is the congestion on I/O resources [80]. The main solutions proposed to decrease full checkpoint/restart overhead are compression strategies to reduce checkpoint size [106, 127], copy-on-write strategies [66, 107] to perform memory-to-memory copies with low latency, and diskless checkpoint strategies to avoid overhead of disk writing [128].

On the other hand, incremental checkpoint optimizes the size of the checkpoint by writing only modified data at page granularity to the stable storage. The first backup remains a full checkpoint, then, successively, the next checkpoint first identify modified pages since the previous checkpoint and only saves them [125]. For efficient reduction of checkpoint size, incremental checkpoint is commonly performed more frequently compared to full checkpoint. However, the increase of the incremental checkpoint frequency and checkpoint at page granularity may lead to global inconsistency and the restart procedure may be more complex than the full checkpoint. In [158], Wang et al. propose a clever balance between full and incremental checkpoints that may significantly reduce the overhead.

1.4.3.3 Some examples of Checkpoint/restart implementations

Most of HPC platforms consist of clusters running Linux operating system. However Linux kernels do not provide checkpoint/restart features to HPC applications. To overcome the absence of checkpoint/restart, some user-level checkpoint/restart implementations like `Libckp` [160] and `Libtckpt` [58] have been developed. `Libckp` is a Linux checkpointing library that provides user applications with checkpoint and rollback routines (`chkpnt()` and `rollback()`, respectively). `Libckp` is naturally designed for fault tolerance and uses an appropriate coordinated checkpoint strategy based on lazy checkpoint coordination [159] in order to perform checkpoint/restart with affordable overhead. `Libtckpt` is another user-level fault tolerance checkpointing library extended to multithreaded applications.

Berkeley Lab Checkpoint/Restart (`BLCR`) [60], and `TICK` [74], provide kernel-level checkpoint/restart libraries for Linux. They provide a checkpoint/restart kernel module for processes, including multithreaded processes, and has also been extended to MPI [68] parallel applications. `BLCR` uses coordinated full checkpoint mechanism, whereas `TICK` implements incremental checkpoint strategy.

Furthermore, Fault Tolerance Interface (`FTI`) [14] implements multilevel checkpoint/restart strategies at application-level. `FTI` consists mainly of four checkpointing interfaces. The first level consists in checkpointing data on local storage and it is mainly designed to recover from soft faults. The second level of checkpoint is called partner-copy checkpoint [167] and it can recover from a single node fault. The third level is a Reed-Solomon encoding checkpoint [134] for multiple node crashes while the fourth level is dedicated to Parallel File System (PFS) checkpoint which can recover from whole system crashes [167]. In addition, `FTI` allows users to specify data to protect.

OpenMP [47] is commonly used for parallel applications in shared memory environments. In [31], checkpointing techniques have been implemented for OpenMP shared-

memory programs based on compiler technology to instrument codes in order to design self-checkpointing and self-restarting applications on any platform. This application-level checkpointing proposed for OpenMP programs is twofold. On the one hand, [31] have developed a pre-compiler to instrument application source code such that checkpoints become intrinsic to the application. On the other hand, they have developed a runtime system to perform the checkpoint/restart procedures. The checkpoint protocol is blocking and may induce a performance penalty. However, ongoing promising studies for non-blocking checkpoint protocol shall limit the overhead. Similar approaches have been proposed for MPI programs in [29, 30].

1.4.4 Diskless checkpoint techniques

Diskless checkpointing techniques [124] are commonly used in parallel distributed environments to checkpoint applications without relying on external storage. Indeed, the most important part of the overhead of the classical checkpoint strategy consists of writing the checkpoint to disk. Since memory access is significantly faster than disk access, the diskless checkpoint technique takes advantage of available memory to store checkpoints. The diskless approach thus aims at limiting the performance penalty of traditional disk-based checkpoint schemes. Several variants of the diskless checkpoint technique are studied to improve application performance. A coordinated diskless checkpoint technique based on extra nodes for checkpointing, first presented in [128], has been implemented in [126] for well-known linear algebra algorithms such as Cholesky factorization, LU factorization, QR factorization, and Preconditioned Conjugate Gradient (PCG). The algorithm forces each node to allocate a certain amount of extra memory for checkpointing. Thus each node writes its local checkpoint in its physical memory, and a parity checkpoint is stored in extra nodes only dedicated for checkpoint using common encoding algorithms, such as RAID algorithm [38]. In this model, failed nodes are recovered using checkpoints in both computing nodes and parity nodes. However, this model can tolerate only a limited number of faults. In addition, when a computing node and a parity node fail simultaneously, lost data cannot be recovered. Another diskless checkpoint technique proposes a model in which, each node stores its local checkpoint in neighbor nodes memory. This approach is called neighbor-based checkpoint.

The neighbor-based checkpointing consists of two steps. First, for each node, a neighbor node is defined among its peers. Then each computational node saves a full local checkpoint in its neighbor memory. If the main advantage of this technique is that it does not require neither additional nodes nor encoding algorithm, it may be very penalizing in terms of memory consumption. A trade-off between the encoding and the neighbor-based approaches is proposed in [42]. In the neighbor-based encoding model, the checkpoint encoding instead of being stored in extra nodes, is stored in neighbor memory. This approach lowers the checkpoint overhead, and when a fault occurs, the recovery involved only the neighbors of the failed nodes. However this technique may exhibit higher overhead if applied to applications that require larger sizes of checkpoints. Finally, to survive the worst case of all node crashes, a clever combination of the traditional disk checkpoint and the diskless approach proposed in [155] could be an affordable trade-off.

1.4.5 Limitation of checkpoint/restart techniques

The traditional disk checkpoint technique is robust but requires additional usage of external storage resources. Furthermore, it is forced to rollback all processes to recover single process faults. On the other hand, it may not scale well in certain cases as reported in [33]. The diskless variant is promising for applications that do not require large sizes of checkpoints, but its real limitation is that it cannot recover from a whole system crash, and it may require extra resources such as memory, node, and network [41].

To overcome checkpoint technique drawbacks, application designers must focus on exploiting application particularities in order to design most appropriate fault tolerance techniques that fit well with each application. Another approach may consist in investigating algorithms which have the property of natural fault tolerance. For example, [73] has proposed new fault tolerant super-scalable algorithms that can complete successfully despite node faults in parallel distributed environments. Even though all applications cannot be re-designed to be naturally fault tolerant, some applications such as meshless finite difference algorithms have demonstrated natural fault tolerance. A possible drawbacks of this approach is that it requires application modification and a deep involvement of application developer for efficient fault tolerance. However to exploit extreme-scale machines, HPC application developers cannot continue to relegate fault tolerance to second place, expecting that general fault tolerance techniques may be used with low overhead, while applications may exhibit nice properties to achieve efficient fault tolerance cheaply.

1.4.6 Checksum-based ABFT techniques for fault detection and correction

Algorithm based fault tolerance (ABFT) is a class of approaches in which algorithms are adapted to encode extra data for fault tolerance at expected low cost. The basic idea is to maintain consistency between encoded extra data and application data. When a fault occurs, encoded extra data are exploited to recover lost data. Checksum-based ABFT was first proposed in 1984 by Abraham et al. [89,95] to design low cost fault tolerant algorithms for matrix-based computations. In recent years, because of the high penalty of traditional fault tolerance techniques, ABFT has been considerably reviewed for many linear algebra applications. The great advantage of ABFT is that it can be easily integrated in existing applications with affordable overhead in terms of time. The basic task of checksum-based ABFT schemes is to encode information in checksums. For a given vector $x \in \mathbb{C}^n$, a checksum of x, denoted x_c may be computed as $x_c = \sum_1^n x_i$, where $x_i =$ is the i^{th} entry of x. For a better understanding of ABFT schemes, let us consider the example of a matrix-vector multiplication $b = Ax$ where $A \in \mathbb{C}^{n \times n}$ is the coefficient matrix, $x \in \mathbb{C}^n$ is a given vector and $b \in \mathbb{C}^n$, is the resulting vector. Their respective checksums may be encoded as illustrated in Figure 1.6.

During the computation, bit-flips may occur in the entries of A, in the input vector x or in the result vector b. For fault detection, extra information may be encoded in additional checksum rows/columns. The encoded row vector A_{cc} and column vector A_{rc} denote the checksum column of A and the checksum row of A respectively, with $A_{cc}(j) = \sum_i^n a_{ij}$ and

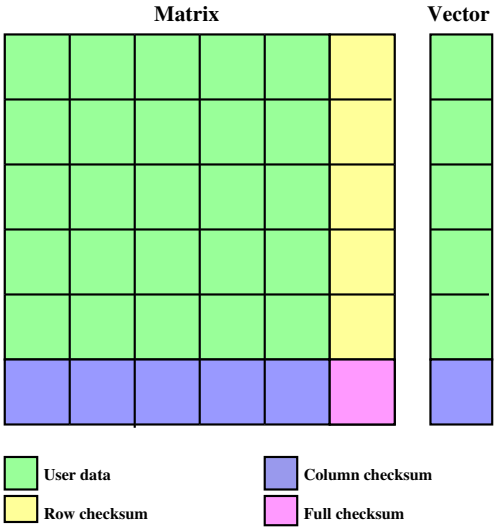


Figure 1.6 – Linear checksum encoding for matrix vector multiplication.

$A_{cr}(i) = \sum_j^n a_{ij}$. In addition, a full checksum consisting of the summation of all the entries of A may also be computed. To check the correctness of the matrix-vector multiplication, the checksum of b ($b_c = \sum_1^n b_i$) is compared to $\tilde{b_c} = A_{cc}x$. In exact arithmetic, b_c is equal to $\tilde{b_c}$, so a difference may be reported as fault. This approach is an efficient scheme for fault detection in matrix vector multiplication. The fault may be accurately located by using the available checksums since each checksum must satisfy a specific property (sum of the data encoded here). The basic checksum described here, may help to correct a single entry corruption while weighted checksum developed in [95] is commonly used for multiple fault detection and correction.

The checksum-based ABFT can be easily extended to recover data associated with failed processes in a parallel distributed environment. Let us consider the example of a scientific application executing n parallel processes $(P_1, P_2, \ldots, P_n)$, in which the critical data of each process P_i necessary for the application completion is denoted D_i. If all critical data satisfy the checksum equality $D_1 + D_2 + \cdots + D_n = D$, then any lost data D_i can be recovered from the checksum as follow, $D_i = D - \sum_{j \neq i} D_j$. This approach has been successfully applied to the ScaLAPACK[4] matrix-matrix multiplication kernel in [41], to recover multiple simultaneous process faults with a very low performance overhead.

In floating point arithmetic, roundoff errors may be indistinguishable from soft faults of small magnitude [19]. In these situations, ABFT techniques for fault detection may consider roundoff errors as faults, and may waste computational time trying to locate corrupted data. Conversely, with propagation of roundoff errors, detected fault may become difficult to locate due to numerical inaccuracy in computation. ABFT techniques must tolerate inaccuracies due to roundoff errors in floating-point arithmetic. [135] discusses how the upper bound of roundoff errors can be used to provide efficient ABFT for fault detection, and how to minimize roundoff error in checksum encoding. In practice, ABFT might

[4]ScaLAPACK is a library of high-performance linear algebra routines for parallel distributed memory machines. ScaLAPACK solves dense and banded linear systems, least squares problems, eigenvalue problems, and singular value problems.

turn out to be very efficient, because application numerical behavior is exploited to set an appropriate threshold, even though distinguishing faults closed to roundoff errors may be very challenging in general.

1.4.7 ABFT techniques without checksums for fault recovery

Even though the overhead of ABFT techniques is often presented as low, the computation of data checksum may possibly increase the fault tolerance scheme overhead. The checksum computation can be avoided if the application data naturally satisfy some mathematical properties, which can be exploited for soft fault detection and/or correction. The basic idea is to report soft fault if the routine output does not satisfy correction solution properties. The basic approach checks the output correctness at the end of the computation. In the particular case of iterative solvers, faults do not only lead to incorrect output, but may also considerably slow down the convergence and increase the computation time [28, 113, 149]. To save computation time, [40] proposed online fault detection techniques for Krylov subspace iterative methods to detect faults as soon as possible during application execution. This approach checks application properties such as the orthogonality between vectors of the Krylov basis or the following equality $r^{(k)} = b - Ax^{(k)}$ between the iterate $x^{(k)}$ and the current residual $r^{(k)}$, that is true in exact arithmetic but only satisfied up to a small relative perturbation in finite precision. If checksum-less ABFT lowers the overhead of fault detection, additional scheme is required to recover lost data. In [101], a checksum-less ABFT technique called lossy approach is proposed for ensuring the recovery of iterative linear methods. The idea also consists in exploiting application properties for data recovery. One attractive feature of this approach is that it has no overhead in fault-free execution. We further detail these mechanisms in the next chapter.

1.4.8 Fault tolerance in message passing systems

Communications between processes in a parallel distributed environment rely on message passing libraries. The dominant message passing library used in the HPC community is the Message Passing Interface (MPI). The default error handler on the `MPI_COMM_WORLD` communicator is `MPI_ERRORS_ARE_FATAL` and without a specific error management policy, the failed process as well as all executing processes will exit. There is another error handler called `MPI_ERRORS_RETURN`. `MPI_ERRORS_RETURN` prevents the exit of all executing processes when a fault occurs. `MPI_ERRORS_RETURN` returns the error and let the user handle it. This can be achieved by replacing the default as follows: `MPI_Errhandler_set(MPI_COMM_WORLD, MPI_ERRORS_RETURN)`. There is no standard specification for fault recovery in MPI applications. Although the MPI forum [5] is actively working on that question, fault tolerance is often achieved in MPI applications by checkpoint/restart schemes and message logging protocols.

[5] http://www.mpi-forum.org

1.4.8.1 Fault tolerance in MPICH

Research studies have led to different specifications of fault tolerant features in MPICH and the most significant effort is MPICH-V [21], an automatic fault tolerant MPI-level mechanism based on an optimal combination of uncoordinated checkpoint/restart and message logging schemes.

The fault tolerant mechanisms proposed by MPICH-V are transparent to the applications. With respect to the MPI standard, they do not require any modification in the existing MPI application codes, except re-linking them with the MPICH-V library. Furthermore, the pessimistic message logging protocol implemented in MPICH-V has been improved in MPICH-V2 [25] to reduce the overhead and performance penalties of the message logging algorithm and the checkpoint protocol. Despite the improvement, MPICH-V2 suffers from the synchronizations intrinsic to pessimistic message logging protocols. This penalty is removed thanks to the causal message logging protocol implemented in the MPICH-V project [26]. MPICH-V project studied how to combine efficiently checkpoint and message logging approaches to design fault tolerant strategies.

In [79], researchers discuss new approaches to design fault tolerance in MPI programs. One idea is to design transparent fault detection and recovery features. This approach is a full MPI-level fault tolerant mechanism, because MPI must decide the best policy to handle faults, and must provide data recovery features. Another approach is to modify MPI semantics to allow MPI to report faults without exiting, and let the user reconfigure the communicator and use her/his own recovery mechanism.

1.4.8.2 FT-MPI

FT-MPI [64] implements a variant of MPI semantics. FT-MPI provides fault tolerant features at MPI level. It can survive process crashes and continue running with few or the same number of processes. FT-MPI provides four options to handle faults. As reported in Table 1.4 the end-user may choose to replace failed processes or continue the execution with surviving processes only depending on the application. Furthermore, this approach is very

Mode	Behavior
ABORT	All processes exit (Default MPI)
BLANK	Surviving processes continue with the same rank in the same communicator
SHRINK	A new communicator of small size is created for surviving processes
REBUILD	Failed processes are replaced in the same communicator

Table 1.4 – FT-MPI modes

flexible as the application developer may provide her/his own recovery modules, however FT-MPI does not conform to the MPI-2 standard. As discussed in [79] an alternative for the FT-MPI implementation is to extend the MPI semantics to add new functions for fault management instead of modifying existing standard semantics. FT-MPI is not in development anymore because it merged with Open MPI. However MPI-level fault tolerance

efforts from FT-MPI have led to the User-Level Failure Mitigation (ULFM) project [17] currently proposed to be integrated in the future MPI Standard (MPI-4.0) [154].

1.4.8.3 User-Level Failure Mitigation

The ULFM project was initially proposed as a specification for fault handling in the MPI-3 Standard. The main focus of ULFM is to provide functions that allow any failed MPI process to report faults successfully without deadlock. Application correct completion also must be guaranteed for non failed processes [18]. ULFM typically provides three main features to handle faults in MPI applications namely, fault detection, notification, and recovery. Contrary to FT-MPI, there is a great standardization effort behind the ULFM project together with a prototype[6] implemented in Open MPI. Preliminary analysis of the prototype demonstrates that it does not decrease existing application performance. Furthermore, ULFM does not require a lot of code modifications to make MPI applications resilient, assuming recovery policies are provided by users [17]. ULFM is a promising project because it opens the door for the exploration of new fault tolerant algorithms to recover fault locally as discussed in [154].

1.5 Faults addressed in this work

In this chapter, we have reviewed concepts in fault-tolerance research in the HPC community. We have mainly emphasized on checkpoint/restart strategies, ABFT techniques and fault tolerant MPI implementations because they represent the current trend in the HPC community. However, the main objective of this thesis is to investigate numerical resilient strategies that do not induce any overhead when no fault occurs.

We have presented the taxonomy of fault namely failure, hard fault and soft fault. Since a failure is related to the final output of a program, our aim is not to correct failures but to avoid failure by providing numerical remedies to address hard faults and soft faults. In parallel applications, a hard fault may lead to the immediate interruption of the whole application. This case is out of the scope of this work. On the other hand, a hard fault may consist in the immediate interruption of a given proportion of processes. In this last case, we investigate numerical remedies to regenerate the data loss due to the hard fault. Soft faults are more likely to induce data corruption, and in a such situation, we focus on strategies for the regenerate the corrupted data. To summarize, in this work, we investigate numerical remedies for data regeneration in the case of partial data corrupted/loss regardless of the type of the fault. Our numerical approaches are furthermore introduced in Chapter 2 and Chapter 3. They could be combined with most of fault tolerance strategies exposed in this chapter to provide a fault tolerance toolkit for resilient large scale simulations. In the remainder of this work, instead of actually crashing a process, we rely on fault injection protocols to simulate its crash.

[6] fault-tolerance.org

Part I

Interpolation-restart Strategies

General framework

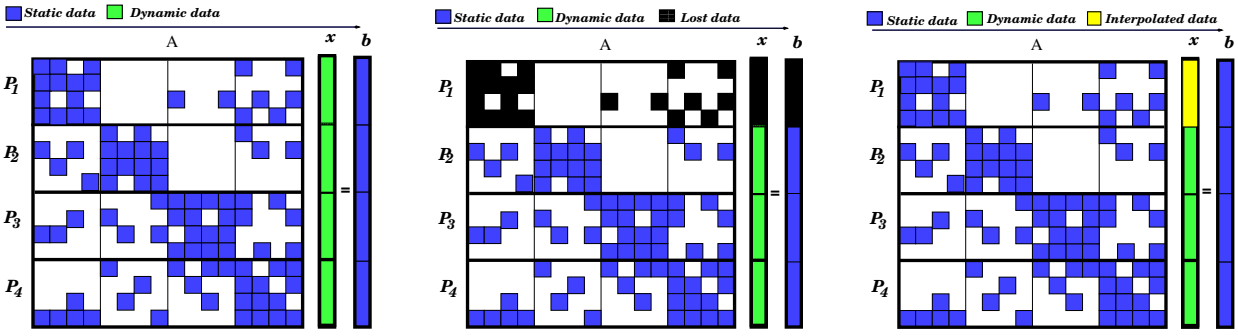
In Chapter 1, we have reviewed a large spectrum of studies on fault tolerance in the HPC community. This part focuses on our own contribution. We present new numerical resilience approaches called interpolation-restart techniques for both large sparse linear system of equations (Chapter 2) and large sparse eigenvalue problems (Chapter 3). In this introducing section, we present the general framework of our approach, the fault model considered in the context of this thesis and the main assumptions we rely on for the design of interpolation-restart (IR) strategies.

Assumption 1. *In our parallel computational context, all the vectors or matrices of dimension n are distributed by blocks of rows in the memory of the different computing nodes. Scalars or low dimensional matrices are replicated.*

According to Assumption 1, scalars or low dimensional matrices are replicated on all nodes while large vectors or matrices of dimension n are distributed according to a block-row partition. Let N be the number of partitions, such that each block-row is mapped to a computing node. For all p, $p \in [1, N]$, I_p denotes the set of row indices mapped to node p. For the sake of illustration, we consider a linear system of equations $\mathcal{A}x = b$. With respect to this notation, node p stores the block-row $A_{I_p,:}$ and x_{I_p} as well as the entries of all the vectors involved in the solver associated with the corresponding row indices of this block-row. If the block A_{I_p,I_q} contains at least one non zero entry, node p is referred to as neighbor of node q as communication will occur between those two nodes to perform a parallel matrix-vector product. By $J_p = \{\ell, a_{\ell,I_p} \neq 0\}$, we denote the set of row indices in the block-column $A_{:,I_p}$ that contain non zero entries and $|J_p|$ denotes the cardinality of this set.

For the sake of simplicity of exposure, we describe IR strategies in the framework of node crashes in a parallel distributed framework. We assume that when a fault occurs on a node, all available data in its memory is lost. This situation is often referred in the literature [27] as hard fault. We consider the formalism proposed in [101] 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 set up during the initialization phase (symbolic step, computation of the preconditioner when it applies, to name a few) and that remain unchanged during the computation. For the sake of illustration, we consider a linear system of equations $\mathcal{A}x = b$. In this particular case static data are the coefficient matrix A, the right-hand side vector b. The Krylov basis vectors (e.g., Arnoldi basis, descent directions, residual, ...) and the iterate are examples of dynamic data. In Figure 1.7a, we depict a block row distribution on four nodes. 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 node 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 1.7b). We further assume that when a fault occurs, the failed node is replaced and the associated computational environment and static data are restored on the new node [101]. In Figure 1.7c for instance, the first matrix block row as well as the corresponding right-hand side are restored as they are static data. However

the iterate, being dynamic data, is definitely lost and we discuss in Chapter 2 strategies for regenerating it.



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

Figure 1.7 – General interpolation scheme. The matrix is initially distributed with a block row partition, here on four nodes (a). When a fault occurs on node P_1, the corresponding data is lost (b). Whereas static is assumed to be immediately restored, dynamic data that has been lost cannot and we investigate numerical strategies for regenerating it (c).

This approach remains valid for eigenvalues problems $\mathcal{A}u = \lambda u$. In the case of eigenvalue problems, the eigenvalues λ being a scalar, is assumed to be replicated (on all nodes) whereas u the eigenvector, being a vector of dimension n, is assumed to be distributed. In general, according to Assumption 1 replicated data can be retrieved from any surviving node and only distributed data need to be regenerated.

We consider iterative methods for the solution of large sparse linear systems and large sparse eigenvalue problems. From a given initial guess, these numerical schemes may converge through successive iterations to a satisfactory solution. Thus IR strategies do not attempt to regenerate all the dynamic data but only the iterate. The interpolated entries and the current values available on the other nodes define the new initial guess to restart the iterations. More precisely we investigate resilience techniques that interpolate the lost entries of the iterates using interpolation strategies making sense for the problem to be solved.

Finally, we assume in the rest of this thesis that a fault occurs during iteration k+1 and the proposed interpolations are thus based on the values of the dynamic data available at iteration k. Here, to minimize the notation, we mix k that is the outer loop in the algorithms description with k the iteration when the fault occurs.

This part is organized as follows. In Chapter 2, we introduce interpolation strategies that are only applied to the current iterate of the linear system solver to define a new initial guess to restart. In Chapter 3, we extend these interpolation ideas to eigensolvers where additional numerical features can be exploited so that more dynamic data can be interpolated.

Summary of general assumptions

1. Large dimensional vectors and matrices are distributed
2. Scalars and low dimensional vectors and matrices are replicated
3. There is a system mechanism to report faults
4. Faulty process is replaced
5. Static data are restored

Chapter 2

Interpolation-restart strategies for resilient parallel linear Krylov solvers

Contents

> ... Iterative methods are not only great fun to play with and interesting objects for analysis, but they are really useful in many situations. For truly large problems they may sometimes offer the only way towards a solution, ...
>
> H.A. van der Vorst

2.1 Introduction

Solving large sparse systems of linear equations is numerically very challenging in many scientific and engineering applications. To solve systems of linear equations, direct methods based on matrix decompositions, are commonly used because they are very robust. However to solve large and sparse systems of linear equations, direct methods may require a prohibitive amount of computational resources (memory and CPU). To overcome the drawbacks of direct solvers, iterative solvers constitute an alternative widely used in many engineering applications. The basic idea is to approximate the solution of large sparse systems of linear equations, through successive iterations that require less storage and less operations. Besides having attractive computational features for solving large sparse systems of linear equations, iterative methods are potentially more resilient. After a "perturbation" induced by a fault, the computed iterate can still serve as an initial guess as long as the static data that define the problem to solve, that are the matrix and the right-hand side, are not corrupted. We exploit the natural resilience potential to design robust resilience iterative solvers which may still converge in the presence of successive faults.

In this chapter, we focus on resilience 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 and analyze the interpolation-restart (IR) strategy introduced for GMRES in [101]. The basic idea consists in computing meaningful values for the lost entries of the current iterate, through a small linear system solution, to build a new initial guess for restarting GMRES. We propose an interpolation approach based on a linear least squares solution that ensures the existence and uniqueness of the regenerated entries without any assumption on the matrix associated with the linear system. Furthermore we generalize the techniques to the situation of multiple concurrent faults. In addition, using simple linear algebra arguments, we show that the proposed IR schemes preserve key monotony properties of CG and GMRES.

The remaining of the chapter is organized as follows. In Section 2.2 we present various IR techniques and analyze their numerical properties. Multiple fault cases are also discussed and we describe different approaches to handle them. We briefly describe the main two subspace Krylov solvers that we consider, namely CG and GMRES methods. For each method, we propose the IR strategy that preserves its numerical properties. We particularly focus on variants of preconditioned GMRES and discuss how the location (right or left) of the preconditioner impacts the properties of our IR strategies. Section 2.3 presents a few numerical experiments where the fault rate and the volume of damaged data are varied to study the robustness of the IR strategies. Some conclusions and perspectives are discussed in Section 2.4.

2.2 Strategies for resilient solvers

Relying on our fault model, we present interpolation strategies designed for Krylov subspace solvers. We first assume that only one node can fail at a time (i.e., iteration) in Sections 2.2.1 and 2.2.2; we relax that assumption in Section 2.2.3 for studying the multiple fault case.

2.2.1 Linear interpolation

The linear interpolation, first introduced in [101] and denoted LI in the sequel, consists in interpolating lost data by using data from non-failed nodes. Let $x^{(k)}$ be the approximate solution when a fault occurs. After the fault, the entries of $x^{(k)}$ are known on all nodes except the failed one. The LI strategy computes a new approximate solution by solving a local linear system associated with the failed node. If node p fails, $x^{(LI)}$ is computed via

$$\begin{cases} x^{(LI)}_{I_q} = x^{(k)}_{I_q} & \text{for } q \neq p, \\ x^{(LI)}_{I_p} = A^{-1}_{I_p,I_p}(b_{I_p} - \sum_{q \neq p} A_{I_p,I_q} x^{(k)}_{I_q}). \end{cases} \tag{2.1}$$

The motivation for this interpolation strategy is that, at convergence (i.e., $x^{(k)} = x$), it regenerates the exact solution ($x^{(LI)} = x$) as long as A_{I_p,I_p} is nonsingular. Furthermore we show that such an interpolation exhibits a property in term of A-norm of the error for symmetric positive definite (SPD) matrices as expressed in the proposition below.

Proposition 1. *Let A be SPD. Let $k+1$ be the iteration during which the fault occurs on node p. The regenerated entries $x^{(LI)}_{I_p}$ defined by Equation (2.1) 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 regenerated using the LI strategy (2.1), we have:*

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

Proof. 1. Uniquely defined $x^{(LI)}_{I_p}$: because A is SPD so is A_{I_p,I_p} 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 node case and assume that the first node fails. Let $A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}$ be an SPD matrix, where $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ denotes the exact solution of the linear equation. The equations associated with the exact solution are:

$$\begin{aligned} A_{1,1}x_1 + A_{1,2}x_2 &= b_1, \\ A_{2,1}x_1 + A_{2,2}x_2 &= b_2. \end{aligned} \tag{2.2a}$$

By linear interpolation (Equation (2.1)), we furthermore have:

$$A_{1,1}x^{(LI)}_1 + A_{1,2}x^{(k)}_2 = b_1, \tag{2.3a}$$

$$x^{(LI)}_2 = x^{(k)}_2. \tag{2.3b}$$

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

$$y^T A z = y_1^T A_{1,1} z_1 + y_1^T A_{1,2} z_2 + y_2^T A_{2,1} z_1 + y_2^T A_{2,2} z_2,$$

$$\begin{aligned} \|y\|_A^2 &= y_1^T A_{1,1} y_1 + y_2^T A_{2,2} y_2 + 2 y_1^T A_{1,2} y_2, \\ \|y-z\|_A^2 &= y^T A y - 2 y^T A z + z^T A z, \qquad (2.4) \\ (y+z)^T A (y-z) &= y^T A y - z^T A z. \qquad (2.5) \end{aligned}$$

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 (2.3b) and (2.4) that:

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

By (2.2a) and (2.5), we have:

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

Because A is SPD, so is $A_{1,1}$ and $A_{1,1}^T A_{1,1}^{-1} = I$. Then by (2.3a), 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 this proof also gives us a quantitative information on the error decrease:

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

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

2.2.2 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. Assuming that

node p has failed, x_{I_p} is interpolated as follows:

$$\begin{cases} x_{I_q}^{(LSI)} = x_{I_q}^{(k)} & \text{for } q \neq p, \\ x_{I_p}^{(LSI)} = \underset{x_{I_p}}{\operatorname{argmin}} \|(b - \sum_{q \neq p} A_{:,I_q} x_q^{(k)}) - A_{:,I_p} x_{I_p}\|. \end{cases} \tag{2.6}$$

We notice that the matrix involved in the least squares problem, $A_{:,I_p}$, is sparse of dimension $|J_p| \times |I_p|$ where its number of rows $|J_p|$ depends on the sparsity structure of $A_{:,I_p}$. 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 node p. The regenerated entries of $x_{I_p}^{(LSI)}$ defined in Equation (2.6) 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 (2.6), we have:*

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

Proof. 1. Uniquely defined: because A is non singular, $A_{:,I_p}$ has full column rank.

2. Monotonic residual norm decrease: the proof is a straightforward consequence of the definition of $x_{I_p}^{(LSI)} = \underset{x_{I_p}}{\operatorname{argmin}} \|(b - \sum_{q \neq p} A_{:,I_q} x_q^{(k)}) - A_{:,I_p} x_{I_p}\|$

□

Remark 1. *Note that the LSI 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 technique will regenerate an initial guess that also complies with the stopping criterion.*

2.2.3 Multiple faults

In the previous section, we have introduced two policies to handle a single fault occurrence. Although the probability of this event is very low, multiple faults may occur during the same iteration especially when a huge number of nodes is used. At the granularity of our approach, these faults may be considered as simultaneous. If two nodes p and q fail simultaneously but they are not neighbors, then x_{I_p} and x_{I_q} can be regenerated independently with LI and LSI as presented above. In this section, on the contrary, we focus more precisely on simultaneous faults on neighbor nodes p and q as illustrated by Equation (2.7) as $\|A_{I_p,I_p}\| \neq$

0 or $\|A_{I_q,I_q}\| \neq 0$.

$$\begin{pmatrix} \cdots & \cdots & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ A_{I_p,I_{p-1}} & A_{I_p,I_p} & \cdots & A_{I_p,I_q} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ A_{I_q,I_{p-1}} & A_{I_q,I_p} & \cdots & A_{I_q,I_q} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \times \begin{pmatrix} \vdots \\ x_{I_p} \\ \vdots \\ x_{I_q} \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ b_{I_p} \\ \vdots \\ b_{I_q} \\ \vdots \end{pmatrix} \tag{2.7}$$

We call multiple fault case this situation of simultaneous faults on neighboring nodes and we present here two strategies to deal with such multiple faults.

2.2.3.1 Global interpolation techniques

We consider here a strategy consisting in regenerating the entries of the iterate after multiple faults all at once. With this global recovery technique, the linear system is permuted so that the equations relative to the failed nodes are grouped into one large block. Then by Equation (2.7), we have Equation (2.8).

$$\begin{pmatrix} \cdots & \cdots & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ A_{I_p,I_{p-1}} & A_{I_p,I_p} & A_{I_p,I_q} & \cdots & \cdots \\ A_{I_q,I_{p-1}} & A_{I_q,I_p} & A_{I_q,I_q} & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \times \begin{pmatrix} \vdots \\ x_{I_p} \\ x_{I_q} \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ b_{I_p} \\ b_{I_q} \\ \vdots \end{pmatrix} \tag{2.8}$$

Therefore the recovery technique falls back to the single fault case. The global linear interpolation (LI-G) solves the following linear system (similar to Equation (2.1))

$$\begin{pmatrix} A_{I_p,I_p} & A_{I_q,I_p} \\ A_{I_q,I_p} & A_{I_q,I_q} \end{pmatrix} \begin{pmatrix} x_{I_p}^{(LI-G)} \\ x_{I_q}^{(LI-G)} \end{pmatrix} = \begin{pmatrix} b_{I_p} - \sum_{\ell \notin \{p,q\}} A_{I_p,I_\ell} x_{I_\ell}^{(k)} \\ b_{I_q} - \sum_{\ell \notin \{p,q\}} A_{I_q,I_\ell} x_{I_\ell}^{(k)} \end{pmatrix}.$$

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

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

2.2.3.2 Local interpolation techniques

Alternatively, if neighbor nodes p and q fail simultaneously, x_{I_p} and x_{I_q} can be interpolated independently from each other. Using the LI strategy, the entries of x_{I_p} can be computed using Equation (2.1) assuming that the quantity x_{I_q} is equal to its initial value $x_{I_q}^{(0)}$. At the same time, node q regenerates x_{I_q} assuming that $x_{I_p} = x_{I_p}^{(0)}$. We call this approach uncorrelated linear interpolation (LI-U). For example we regenerate x_{I_p} via

1: $x_{I_q}^{(k)} = x_{I_q}^{(0)}$,
2: $x_{I_\ell}^{(LI-U)} = x_{I_\ell}^{(k)}$ for $\ell \notin \{p, q\}$,
3: $x_{I_p}^{(LI-U)} = A_{I_p,I_p}^{-1}(b_{I_p} - \sum_{p \neq q} A_{I_p,I_q} x_{I_q}^{(k)})$.

Although better suited for a parallel implementation, this approach might suffer from a worse interpolation quality when the off-diagonal blocks A_{I_p,I_q} or A_{I_q,I_p} define a strong coupling. Similar ideas 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 penalty of considering $x_{I_q}^{(0)}$ when regenerating x_{I_p}. Basically, to regenerate x_{I_p}, each equation that involves x_{I_q} is discarded from the least squares system and we solve the following equation

$$x_{I_p}^{(LSI-U)} = \underset{x_{I_p}}{\operatorname{argmin}} \| (b_{J_p \setminus J_q} - \sum_{\ell \notin \{p,q\}} A_{J_p \setminus J_q, I_\ell} x_{I_\ell}^{(k)}) - A_{J_p \setminus J_q, I_i} x_{I_p} \|, \tag{2.9}$$

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

We denote this approach by decorrelated LSI (LSI-D). The heuristic beyond this approach is to avoid perturbing the regeneration of x_{I_p} with entries in the right-hand sides that depend on x_{I_q} 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 penalize the overall convergence.

2.2.4 Numerical properties of the Interpolation-Restart Krylov solvers

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

the method of choice used for the solution of linear systems involving SPD matrices [87]. It can be expressed via short term recurrence on the iterate as depicted in Algorithm 2.

Algorithm 2 Conjugate gradient algorithm

1: Compute $r_0 = b - Ax^{(0)}$,
2: $p_0 = r_0$
3: **for** $k = 0, 1, \ldots,$ until convergence, **do**
4: $\quad \alpha_k = r_k^T r_k / p_k^T A p_k$
5: $\quad x^{(k+1)} = x^{(k)} + \alpha_k p_k$
6: $\quad r_{k+1} = r_k - \alpha_k A p_k$
7: $\quad \beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
8: $\quad p_{k+1} = r_{k+1} + \beta_k p_k$
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 [136]). 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 fault does ensure that the A-norm of the forward error associated with the IR strategy is monotonically decreasing for CG and PCG.*

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 minimizes 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 [120]).

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 3, lines 14-16) in the so-called restarted GMRES (usually denoted by GMRES(m)) [137]. When a fault occurs, the approximate solution is not available. However, in compliance with Assumption 1 (see p. 37), in most of the classical parallel distributed GMRES implementations the Hessenberg matrix $\bar{H}_m$ is replicated on each node and the least squares problem is also solved redundantly. Consequently, each individual still running node ℓ can compute its entries I_ℓ of the iterate when a fault occurs. The property of residual norm monotony of full and restarted GMRES is still valid in case of fault for the IR strategies LSI (for single fault) and LSI-G (for multiple faults).

Corollary 2. *The IR strategies LSI and LSI-G ensure the monotonic decrease of the residual norm of minimal residual Krylov subspace methods such as GMRES, Flexible GMRES [140] and MINRES after a restart due to a single or to multiple faults.*

Algorithm 3 GMRES

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

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 preconditioned linear system $MAx = Mb$ where M is the preconditioner. To ensure the monotonic decrease of the preconditioned residual, the least squares problem should involve a block-column of MA, which might be complicated to build depending on the preconditioner. In that case, because left GMRES computes iterates $x^{(k)}$, one might regenerate x using only A but we loose the monotony property. For right preconditioned GMRES, $AMu = b$ with $x = Mu$, similar comments can be made except for block diagonal preconditioners where the property still holds. Indeed, similarly to the unpreconditioned case, if a block diagonal right preconditioner is used, all the entries of u but those allocated on the failed nodes can be computed after a fault. After the computation of u on the surviving nodes, the corresponding entries of x can be computed locally as the preconditioner is block diagonal. Therefore, the new initial guess constructed by LSI or LSI-G still complies with Corollary 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 [140]), so that the still alive nodes can compute their part of x_k from their piece of Z_k.

2.3 Numerical experiments

In this section we investigate first the numerical behavior of the Krylov solvers restarted after a fault 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 2.3.2 numerical experiments where at most one fault oc-

curs during one iteration. In Section 2.3.3, we consider examples where multiple faults occur during some iterations to illustrate the numerical robustness of the different variants we exposed in Section 2.2.3. For the sake of completeness and to illustrate the possible numerical penalty induced by the restarting procedure after the faults, we compare in Section 2.3.4 the convergence behavior of the solvers with and without fault. For the computation of interpolation cost, 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 recovery in Section 2.3.5.

2.3.1 Experimental framework

We recall that the goal of this study is to access the numerical behavior of the proposed resilient Krylov solvers. For the sake of flexibility of the experiments, we have developed a simulator to monitor the amount of data lost at each fault as well as the rate of faults. Given an execution with N nodes, the first task is to generate the fault dates of each node, independently using the Weibull probability distribution that is admitted to provide realistic distribution of faults [162]. For our simulations, we use the shape parameter $k \approx 0.7$ [23], the value of MTBF is a function of cost of iterations in terms of Flop. For example $MTBF = \alpha \times IterCost$, where $Itercost$ is the average time of one iteration, means that a fault is expected to occur every α iterations. During the execution, at each iteration, we have a mechanism to estimate the duration of the iteration (Algorithm 4, line 7). A fault is reported if the iteration coincides with the fault date of at least one of the nodes (Algorithm 4, line 9). The whole fault simulation algorithm is depicted in Algorithm 4.

Algorithm 4 Fault Simulation Algorithm

```
1: for Node = 1 : N do
2:    Generation of fault dates of each node.
3: end for
4: BeginningTime = 0
5: for Iteration = 1 : Max_Iterations do
6:    Fault = no
7:    EndTime = BeginningTime + Get_Iteration_Duration(Iteration)
8:    for Node = 1 : Number_Of_Nodes do
9:       if BeginningTime < Next_Fault_Date(Node) < EndTime then
10:         Fault = yes
11:         Add Node to the list of failed nodes
12:         Update the next fault date of Node
13:      end if
14:   end for
15:   if Fault == yes then
16:      Regenerate lost data
17:   end if
18:   BeginingTime = EndTime
19: end for
```

When a node is reported as faulty during an iteration, we simulate the impact of the fault by setting dynamic data in the memory of the corresponding node to zero before calling the interpolation strategies for data regeneration.

We have performed extensive numerical experiments and only report on the qualitative numerical behavior observed on a few examples that are representative of our observations Most of the matrices come from the University of Florida (UF) test suite [52]. The right-hand sides are computed for a given solution generated randomly.

Name	Origin	Properties
EDP-SPD	Discretization of Equation (2.10)	SPD
MathWorks/Kuu	UF	SPD
Averous/epb0	UF	Unsymmetric
Boeing/nasa1824	UF	Unsymmetric

Table 2.1 – Set of matrices considered for the experiments.

To generate test examples where we can easily vary the properties of the matrix associated with the linear systems, we also consider the following 3D Reaction-diffusion operator defined in the unit cube discretized by a seven point finite difference stencil:

$$-\epsilon\Delta u + \sigma u = f, \tag{2.10}$$

with some boundary conditions. To study the numerical features of the proposed IR strategies, we display the convergence history as a function of the iterations, that also coincide with the number of preconditioned matrix-vector products. For the unsymmetric solver, we depict the scaled residual, while for the SPD case we depict the A-norm of the error. To distinguish between the interpolation quality effect and possible convergence introduced by the restart, we consider a simple strategy that consists in restarting the solver using the current iterate $x^{(k)}$ as the initial guess at faulty iterations (We do not inject fault, only restart the solver at iterations corresponding to faulty iterations observed during LI/LSI execution). We refer to this strategy as Enforced Restart and denote it ER. On the other side of the spectrum, we also depict in red a straightforward strategy where the lost entries of the iterate are replaced by the corresponding ones of the first initial guess. This simple approach is denoted "*Reset*". For the sake of simplicity the acronyms used to denote the names of different curves are recalled in the Table 2.2.

2.3.2 Numerical behavior in single fault cases

In this section we first examine the situation where only one fault occurs during an iteration. We first illustrate in Figure 2.1 the numerical behavior of the IR strategies when the amount of lost entries at each fault varies from 3 % to 0.001 % (a single entry in that latter case). We report on experiments with GMRES(100) for the matrix Averous and refer to Table 2.2 for a short summary of the notations used in the legend. For these experiments, in order to enable comparison, the number of faults is identical and they occur at the same iteration

Acronym	Definition	Fault
Reset	Replace lost data by its initial value	Single/Multiple
LI	Linear interpolation	Single
LI-G	Global linear interpolation	Multiple
LI-U	Uncorrelated linear interpolation	Multiple
LSI	Least square interpolation	Single
LSI-G	Global least square interpolation	Multiple
LSI-U	Uncorrelated least square interpolation	Multiple
LSI-D	Decorrelated least square interpolation	Multiple
ER	Enforced restart	Single/Multiple
NF	No faulty execution	–

Table 2.2 – Definition of the acronyms used in the captions of forthcoming plots.

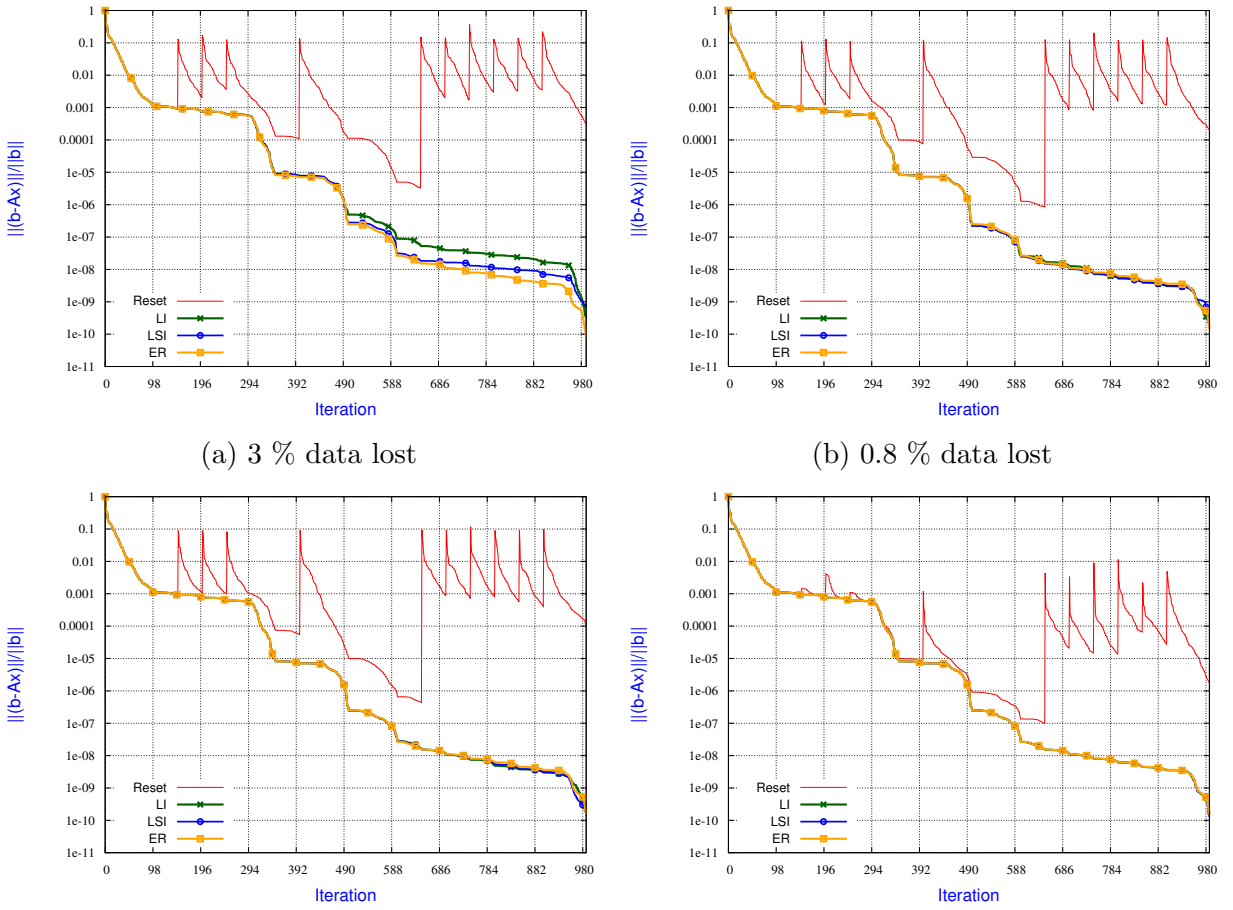
for all the runs. It can first be observed that the straightforward restarting Reset policy does not lead to convergence. Each peak in the convergence history corresponds to a fault showing that the solver does not succeed to converge. In contrast, all the other recovery approaches do ensure convergence and all have very similar convergence behavior; that is, they exhibit similar robustness capabilities. Furthermore, this convergence behavior is not much affected by the amount of data lost.

In Figure 2.2, we investigate the robustness of the IR strategies when the fault rate is varied while the amount of regenerated entries remains the same after each fault, that is 0.2 %. Those experiments are conduced with a GMRES(100) using the kim1 matrix. An expected general trend that can be seen on that example is: the larger the number of faults the slower the convergence. When only a few faults occur, the convergence penalty is not significant compared to the non faulty case. For a large number of faults, the convergence slows down but continues to take place; for instance for an expected accuracy of 10^{-7} the number of iterations with 40 faults is twice the one without fault.

Although not illustrated in the selected numerical experiments reported in Figure 2.1 and 2.2, the LI strategy failed in many of the experiments we ran because of the singularity of the A_{I_p,I_p} block. This constitutes a severe lack of robustness for this approach for non SPD matrices. When LI does not fail, a general trend [5] is that none of the policies LI or LSI appears significantly and consistently superior to the other. The IR strategies based on either of both interpolation strategies have similar numerical behavior and are comparable with ER. This comparison between the IR strategies and ER shows that the regenerated data are numerically as good as lost data in term of approximation of the iterate.

2.3.3 Numerical behavior in multiple fault cases

In this section we illustrate the numerical behavior of the various IR strategies described in Section 2.2.3. We made a selection of a few numerical experiments and reported them in Figure 2.3. We recall that what is referred to as a multiple fault corresponds to the situation



(a) 3 % data lost

(b) 0.8 % data lost

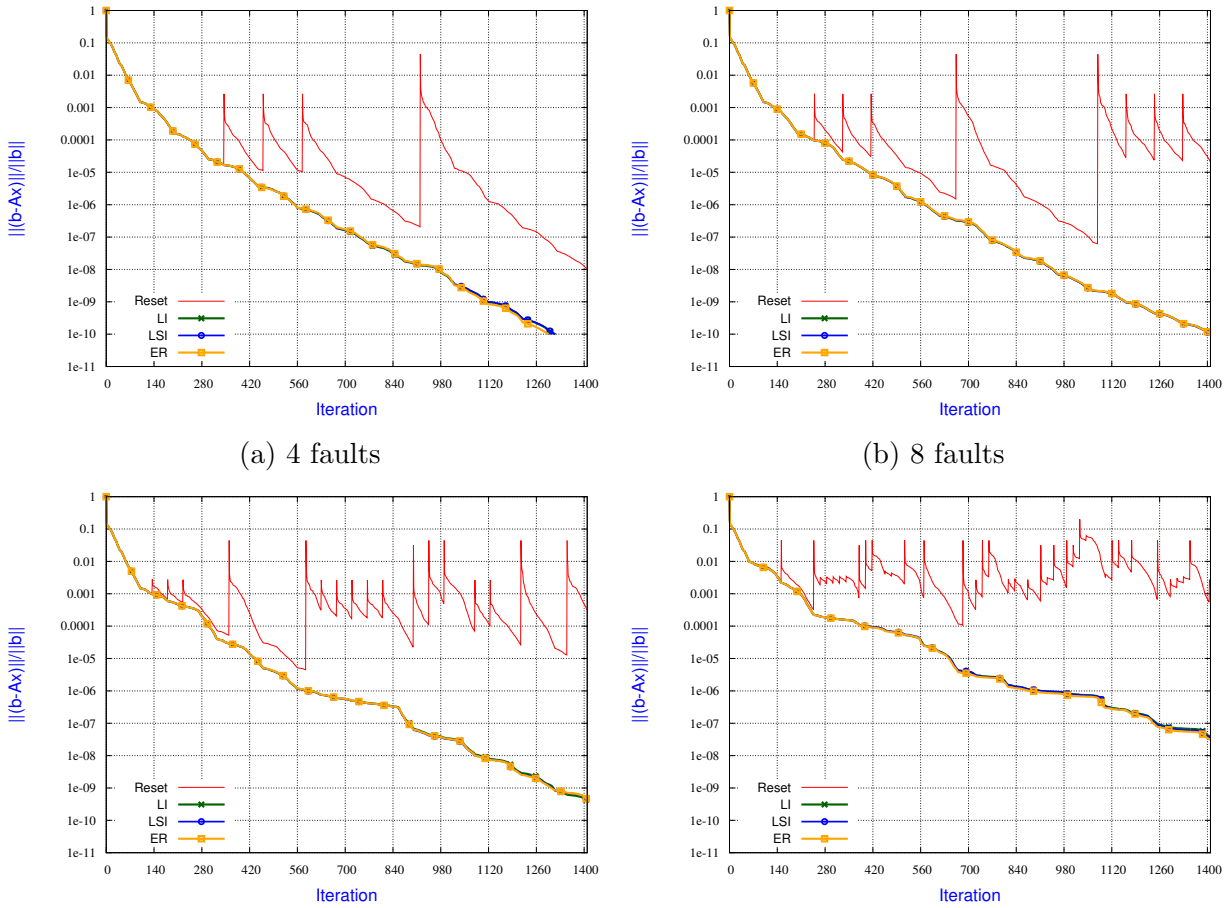
(c) 0.2 % data lost

(d) 0.001 % data lost

Figure 2.1 – Numerical behavior of the IR strategies when the amount of lost data is varied (matrix Averous/epb3 with 10 faults).

where the entries of x_{I_p} and x_{I_q} are lost at the same iteration and either the block A_{I_p,I_q} or the block A_{I_q,I_p} is non zero (i.e., nodes p and q are neighbor). 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 very large number of faults. This situation has a very low probability to occur on real systems but deserves some observations on the numerical behavior of the interpolation schemes in such extreme situations.

In Figure 2.3, 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 IR strategies LI-U and LSI-U, which are almost as poor as the straightforward Reset approach. The underlying idea to design these heuristics was to interpolate lost entries by fully ignoring other simultaneous faults (enabling a natural parallelism in the interpolation). 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 for SPD or the LSI-G for general matrices, where all the lost entries are regenerated at once as if a "large" single



(a) 4 faults

(b) 8 faults

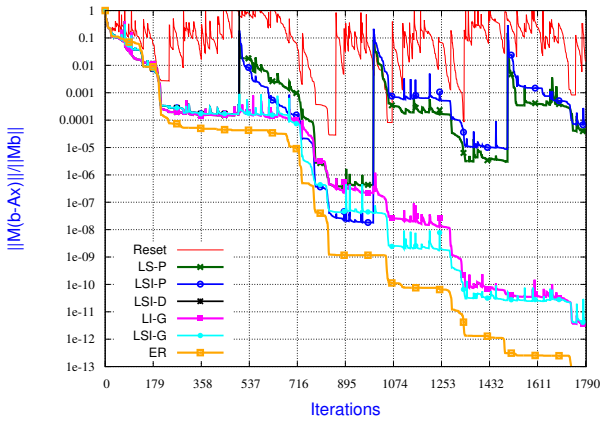
(c) 17 faults

(d) 40 faults

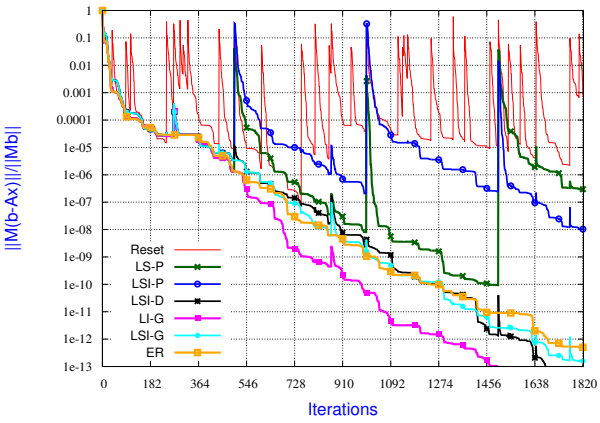
Figure 2.2 – Numerical behavior of the IR strategies when the rate of faults varies (matrix Kim/kim1 with 0.2 % lost data at each fault)

fault occurred. It can be seen in these figures that the numerical behavior is consequently very similar to the ones we observed in the previous section where only single fault was considered. More interesting is the behavior of the LSI-D strategy whose behavior seems to vary a lot from one example to the another. In Figure 2.3c and 2.3b, this policy enables a convergence similar to the robust strategies LI-G and LSI-G, while in Figure 2.3a and 2.3d a jump is observed with this IR strategy (the convergence curve disappears from the plot area at iteration 530 and never shows up again because the preconditioned scaled residual remains larger than the one in Figure 2.3a). Actually, this latter bad behavior occurs when the least squares problem, which is solved once the correlated rows have been discarded, becomes rank deficient. In that case, the regenerated initial guess is extremely poor. In order to overcome 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 IR approach and would thus certainly represent a relevant strategy for such extremely unstable computing environments.

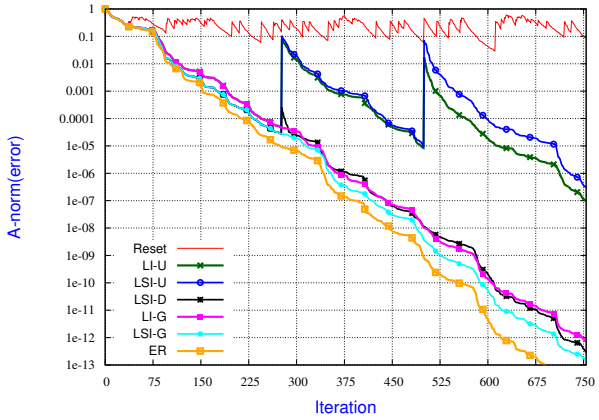
Remark 2. *In Figure 2.3a-2.3b, some residual norm increases can be observed for the LSI variants with GMRES that are due to the left preconditioner location.*



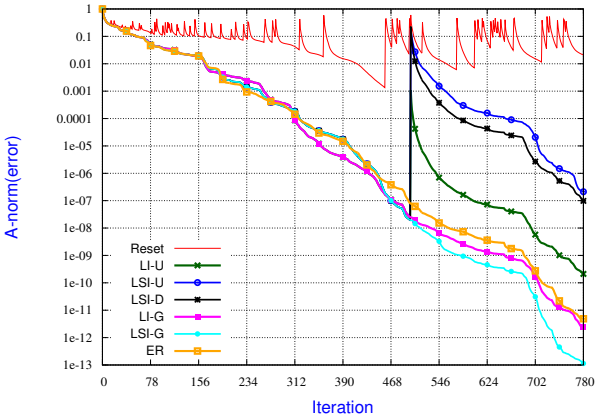
(a) Left preconditioned GMRES (UF Averous/epb0 - 103 single and 3 multiple faults)



(b) Left preconditioned GMRES (UF Boeing/nasa1824 - 32 single faults and 3 multiple faults)



(c) PCG on a 7-point stencil (3D Poisson equation - 67 single and 2 multiple faults)

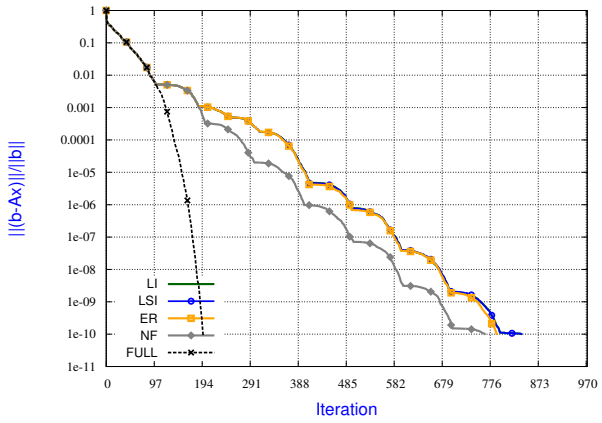


(d) PCG (UF MathWorks/Kuu - 70 single faults and 1 multiple fault)

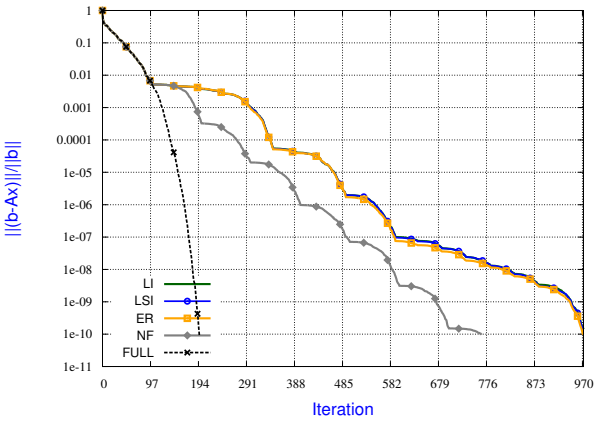
Figure 2.3 – Numerical behavior of the IR strategies with multiple faults.

2.3.4 Penalty of the Interpolation-Restart strategy on convergence

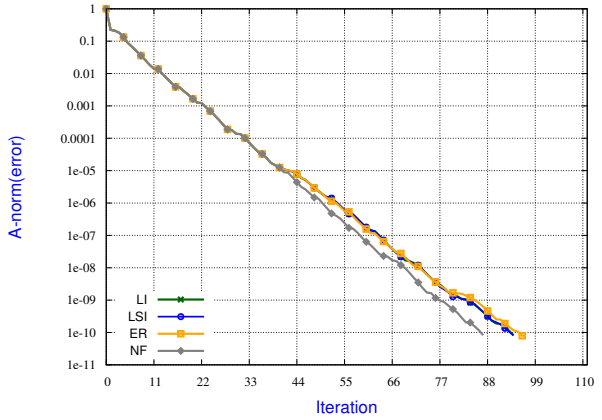
One of the main feature of the resilient numerical algorithms described in this chapter is to restart once meaningful entries have been interpolated to replace the lost ones. When restarting, the Krylov subspace built before the fault is lost and a new sequence of Krylov subspaces is computed. To reduce the computational resource consumption, such a restarting mechanism is implemented in GMRES that it is known to be likely to delay the convergence compared to full-GMRES. This delay can be observed in Figure 2.4a-2.4b, where the convergence history of full-GMRES is also depicted. Although the convergence history of the faulty executions are much slower than the one of full-GMRES, they are not that far and remain close to the non faulty restarted GMRES(100). On the contrary, CG does not need to be restarted. In order to evaluate how the restarting affects the convergence, we display in Figure 2.4c-2.4d the convergence history of CG with and without fault. As already mentionned in Section 2.3.2 for experiments with GMRES, the larger the number



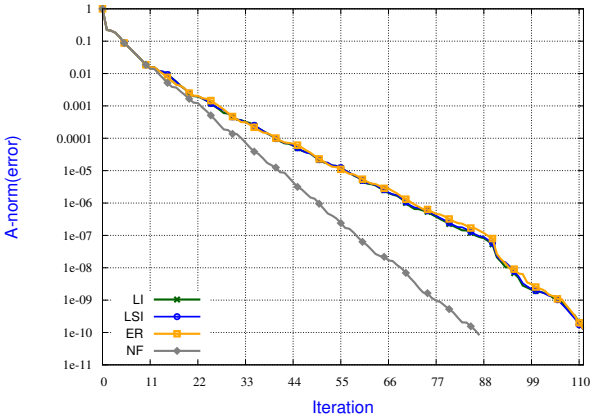
(a) GMRES(100) with matrix Chevron/Chevron2 with 5 faults



(b) GMRES(100) with matrix Chevron/Chevron2 with 10 faults



(c) PCG with matrix Cunningham/qa8fm 2 faults



(d) PCG with with matrix Cunningham/qa8fm 9 faults

Figure 2.4 – Convergence penalty induced by the restart after the interpolation.

of faults, the larger the convergence penalty.

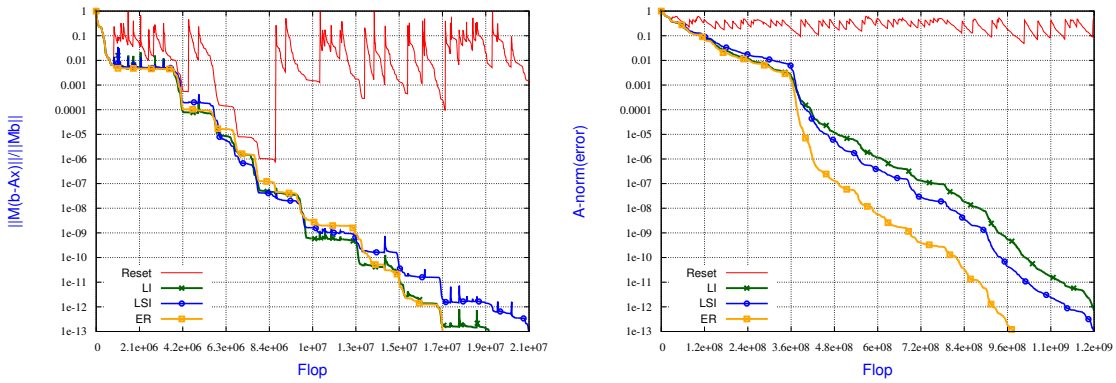
The objective of this chapter is to give some qualitative information on the numerical behavior of IR procedures to enable the Krylov solvers surviving to faults. Nevertheless to be of practical interest, the interpolation cost should be affordable. From a computational view point, the interpolation cost depends on the amount of lost data and the circumstance of the fault. For instance, in a parallel distributed framework, if a block Jacobi preconditioner is considered, the interpolation for LI reduces to the application of the preconditioner [101].

2.3.5 Cost of interpolation strategies

The main objective of this work is to give some qualitative insights on the numerical behavior of IR strategies to enable the Krylov solvers surviving to faults. Nevertheless we also roughly assess the computational cost associated with each of the interpolation alternatives that should remain affordable to be applicable. In that respect we measure the computational complexity in terms of Flops 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 [51] and Umfpack [50] to get their computational cost. We did not account for the communication in the Krylov solver, but accounted 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 node and make the assumption that all the nodes are involved in the interpolation calculation. We furthermore arbitrarily assume that the parallel sparse LU or sparse QR is ran with a parallel efficiency of 50 %.

We report in Figure 2.5a-2.5b the convergence history of the Krylov solvers as a function of the Flop count performed. In can be seen that the qualitative behaviors 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) interpolation 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 ER 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 Flops so that the convergence curves in Figure 2.5a-2.5b should have to be shifted slightly to the left to account for this penalty.



(a) Left preconditioned GMRES on Averous/epb0 with 44 single faults

(b) PCG on EDP-SPD with 70 single faults

Figure 2.5 – Interpolation cost of preconditioned GMRES and PCG while a proportion of 6% of data is lost at each fault.

2.4 Concluding remarks

In this chapter we have investigated some IR techniques to design resilient parallel Krylov subspace methods. The resilience techniques are based on simple interpolation approaches that compute meaningful entries of the lost components of the iterate. Using basic linear algebra arguments, we have shown that for SPD matrices the LI strategy does preserve the A-norm error monotony of the iterates generated by CG and PCG. We have also shown that the LSI strategy does guarantee the residual norm monotony decrease generated by GMRES, Flexible GMRES and MINRES as well as for preconditioned GMRES for some class of preconditioners. For non SPD matrices, the LI strategy lacks robustness as it might not be defined when the diagonal block involved in its definition is singular.

Because we have considered a restarting procedure after the interpolation phase, we have illustrated the numerical penalty induced by the restarting on short terms recurrence Krylov approaches. For CG the convergence delay remains acceptable for a moderate number of faults. 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 IR techniques is usually low. Extensive numerical experiments have shown that these strategies do ensure convergence of the solvers with moderate convergence and computational penalties. In addition, the proposed schemes have no overhead when no fault occurs.

Finally, we have experimented a general procedure applicable to any linear solver. 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. For Krylov solvers, the larger the number of faults, the slower the convergence mainly due to the restart. It will be the focus of future research to tune the recovery method for a specific Krylov solver in order to attempt to recover more information, in particular on the global Krylov space to alleviate the penalty induced by the restart after each fault.

Chapter 3

Interpolation-restart strategies for resilient eigensolvers

Contents

3.1 Introduction

The computation of eigenpairs (eigenvalues and eigenvectors) of large sparse matrices is involved in many scientific and engineering applications for instance when stability analysis

is a concern. It appears for instance in structural dynamics, thermodynamics, thermo-acoustics, quantum chemistry, etc. In this chapter, we extend the IR strategies introduced for linear systems in Chapter 2 to a few state-of-the-art eigensolvers. More precisely, the Arnoldi [11], Implicitly restarted Arnoldi [105] and subspace iteration [141] algorithms have been revisited to make them resilient in the light of faults. Contrarily to the linear system solution, for eigensolvers we attempt to regenerate more dynamic information than just the current eigenpair approximation. For instance, for the Jacobi-Davidson solver, the interpolations are applied to the converged Schur vectors as well as to the best direction candidates in the current search space. After a fault, this new set of vectors are used as initial guess to restart the Jacobi-Davidson iterations.

The remainder of the chapter is structured as follows: in Section 3.2 we describe how the interpolation techniques can be extended to regenerate meaningful spectral information. We briefly present the eigensolvers that we have considered in Section 3.3 as well as how the recovery ideas can be tuned for each of them. Section 3.4 is devoted to the numerical experiments. We discuss the robustness of the various resilient numerical schemes and conclude with some perspectives in Section 3.5.

3.2 Interpolation-restart principles

In this section, we describe how IR strategies can be extended to regenerate meaningful spectral information. Contrarily to what we have proposed for the Krylov linear solvers where only meaningful iterate are computed to serve as a new initial guess for restarting the iterations, more flexibilities exist in the framework of eigensolution where similar ideas can be adapted to best exploit the numerical features of the individual eigensolvers. The main reasons are that some of the considered eigensolvers do not rely on a central equality (as Arnoldi's equality for GMRES) or a sophisticated short term recurrence (as for CG); furthermore we consider also situations where a few *nev* eigenpairs are sought, which also provides additional freedom. We present in details the variants for selecting and computing the relevant subspaces to perform the restart for each particular considered eigensolver in Section 3.3.

3.2.1 Interpolation methods

Following similar ideas as those presented in Chapter 2 for linear systems, we adapt the IR strategies to eigensolvers. The IR strategies consist in interpolating lost data by using noncorrupted data. Let $u^{(k)}$ be the approximate eigenvector when a fault occurs. After the fault, the entries of $u^{(k)}$ are correct, except those in the corrupted memories. Assuming that in parallel distributed environment, the current eigenvalue λ_k is naturally replicated in the memory of the different computing nodes, we present two strategies to compute a new approximate solution. The first strategy, referred to as linear interpolation and denoted LI consists in solving a local linear system associated with the failed node; the second one relies on the solution of a least squares interpolation and is denoted LSI. Those two alternatives result from considering $(\lambda_k, u^{(k)})$ as an exact eigenpair and the equations the lost entries

should comply with. We may alternatively have a row-block view point, which defines the LI variant; or a column-block point of view, which leads to the LSI variant.

If node p fails, LI computes a new approximation of the eigenvector $u^{(LI)}$ as follows

$$\begin{cases} u_{I_q}^{(LI)} = u_{I_q}^{(k)} & \text{for } q \neq p, \\ u_{I_p}^{(LI)} = (\mathcal{A}_{I_p,I_p} - \lambda \mathcal{I}_{I_p,I_p})^{-1}(-\sum_{q \neq q} \mathcal{A}_{I_p,I_q} u_{I_q}^{(k)}). \end{cases}$$

On the other hand, LSI computes $u^{(LSI)}$ via

$$\begin{cases} u_{I_q}^{(LSI)} = u_{I_q}^{(k)} & \text{for } q \neq p, \\ u_{I_p}^{(LSI)} = \arg\min_{u_{I_p}} \|(\mathcal{A}_{:,I_p} - \lambda \mathcal{I}_{:,I_p}) u_{I_p} + \sum_{q \neq p} (\mathcal{A}_{:,I_q} - \lambda \mathcal{I}_{:,I_q}) u_{I_q}^{(k)})\|. \end{cases}$$

Here, $\mathcal{I} \in \mathbb{C}^{n \times n}$ is the identity matrix and we furthermore assume that $(\mathcal{A}_{I_p,I_p} - \lambda I_{I_p,I_p})$ is non singular and that $(\mathcal{A}_{:,I_p} - \lambda \mathcal{I}_{:,I_p})$ has full rank. The matrix involved in the least squares problem, $(\mathcal{A}_{:,I_p} - \lambda \mathcal{I}_{:,I_p})$, is sparse of dimensions $|J_p| \times |I_p|$ where its number of rows $|J_p|$ depends on the sparsity structure of $\mathcal{A}_{:,I_p}$. Consequently the LSI strategy may have a higher computational cost. In the rest of this chapter, we again use IR to denote LI and/or LSI.

3.2.2 Reference policies

To assess the numerical robustness of the IR policies we not only compare them with non faulty (NF) executions, we also rely on Reset and ER strategies. In this context, the Reset strategy consists of substituting lost data ($u_{I_p}^{(k)}$) with random values. The benefits of the IR strategies over Reset will thus highlight both the importance and the quality of the interpolation. On the other hand, the ER strategy consists only in enforcing the solver to restart using current solution as an initial guess. The comparison between NF and ER allows one to measure the numerical impact of the restart while the convergence delay between ER and IR illustrates the quality of the interpolated directions with respect to the eigensolver under study.

When ambiguous, we mention explicitly which directions are affected by the Reset and ER for particular eigensolvers in Section 3.3. In all cases, for the sake of comparison, faults are always injected on the same nodes at the same iteration for all the IR, Reset and ER cases.

3.3 Interpolation-Restart strategies for well-known eigensolvers

In this section we briefly describe a few of the most popular eigensolvers for general sparse matrices that we have considered in this thesis. For each solver we describe which data are

regenerated after a fault to make them resilient. For each eigensolver, we briefly present the numerical approach and associated algorithm and then we describe how the IR strategies (as well as Reset and ER when not straightforward) can be applied to compute key ingredients of the solver when some data are lost.

In the sequel, we introduce eigensolvers to compute a number nev of eigenpairs which ranges from one to five.

3.3.1 Some background on basic methods for computing eigenvectors

In this work, we only consider iterative schemes for the solution of eigenproblems. We define

$$\tilde{\eta}(u^{(k)}, \lambda_k) = \frac{\|Au^{(k)} - \lambda_k u^{(k)}\|}{|\lambda_k|} \tag{3.1}$$

the scaled residual associated with the approximate eigenpair $(\lambda_k, u^{(k)})$ for nonzero eigenvalue approximation. Given a threshold ε, the widely used stopping criterion to detect convergence is defined by

$$\tilde{\eta}(u^{(k)}, \lambda_k) \leq \varepsilon.$$

We refer to Section 1.2.4.3 in Chapter 1 for the detailed motivations for this stopping criterion.

The power method [54, Chapter 5] [82, Chapter 2] depicted in Algorithm 5 is the most basic iterative method to compute the eigenvector associated with the eigenvalue of largest modulus of $\mathcal{A}$.

Algorithm 5 Power method

1: Start: Choose a nonzero initial vector $u^{(0)}$.
2: **for** $k = 0 \ldots$ until convergence **do**
3: $\quad u^{(k+1)} = \dfrac{\mathcal{A}u^{(k)}}{\|\mathcal{A}u^{(k)}\|}$
4: **end for**

The power method consists of computing the sequence of vectors $u^{(k)}$ with the initial guess $u^{(0)}$, a nonzero vector. The power method uses only the vector of the current iteration, throwing away the information contained in the previously computed vectors [142, Chapter 3]. In [82, Chapter 4], it is shown that the spaces spanned by the sequence $u^{(k)}$ of the iterate generated by the power method contain accurate spectral information about the eigenvectors associated with the next largest eigenvalues. The following paragraph describes a procedure for extracting spectral information from a given subspace.

The Rayleigh-Ritz method is commonly used to compute an approximation to an eigenspace $\mathcal{X}$ of a given matrix $\mathcal{A}$ from a subspace $\mathcal{U}$ containing an approximation of $\mathcal{X}$. The subspace $\mathcal{U}$ may be chosen as the Krylov subspace associated with the matrix $\mathcal{A}$, or it may be generated otherwise. The simplest form of the Rayleigh-Ritz procedure to compute

an approximate eigenpair (λ, u) of $\mathcal{A}$ presented in [92] is depicted in Algorithm 6. For the sake of simplicity of notation, we mix u the solution and the approximate solution.

Algorithm 6 Rayleigh-Ritz procedure

1: Start: Compute an orthonormal basis U for the k-dimensional space $\mathcal{U}$.
2: Compute : $C = U^H \mathcal{A} U$
3: Let (λ, g) be an eigenpair of C
4: Set (λ, Ug) as candidate eigenpair of $\mathcal{A}$

The matrix C is known as the Rayleigh quotient matrix associated with U. Let $\sigma(C)$ denotes the spectrum of C, that is the set of all its eigenvalues. The couple (λ, Ug) is called a Ritz pair, where λ is the Ritz value and (Ug) the Ritz vector.

3.3.2 Subspace iterations to compute *nev* eigenpairs

Brief description of the numerical algorithm: The subspace iteration method is a block variant of the power method [82, Chapter 2]. It starts with an initial block of m (with $m \geq nev$) linearly independent vectors $U^{(0)} = [u_1^{(0)}, \ldots, u_m^{(0)}] \in \mathbb{C}^{n \times m}$. Under certain assumptions [141, Chapter 5], the sequence of $U^{(k)} = \mathcal{A}^k U^{(0)}$ generated by the algorithm, converges to the m eigenpairs of $\mathcal{A}$ associated with the eigenvalues of largest magnitude. To guarantee the full column rank in $U^{(k)}$ for large values of k, the Q factor of its QR factorization may be used at each iteration.

Algorithm 7 Basic subspace iteration

1: Choose $U^{(0)} = [u_1^{(0)}, ..., u_m^{(0)}] \in \mathbb{C}^{n \times m}$
2: **for** $k = 0 \ldots$ until convergence **do**
3: Orthonormalize $U^{(k)}$
4: Compute $W^{(k)} = \mathcal{A} U^{(k)}$
5: Form the Rayleigh quotient $C^{(k)} = {W^{(k)}}^H \mathcal{A} W^{(k)}$
6: Compute the eigenvectors $G^{(k)} = [g_1, \ldots, g_m]$ of $C^{(k)}$ and eigenvalues $\sigma(C^{(k)}) = (\lambda_1, \ldots, \lambda_m)$
7: Update Ritz vectors : $U^{(k+1)} = W^{(k)} G^{(k)}$
8: **end for**

To compute the eigenpairs associated with the smallest eigenvalues in magnitude, or eigenpairs associated with a given set of eigenvalues in a given region of the complex plane, the basic subspace iteration depicted in Algorithm 7 is no longer appropriate. For example, to compute eigenpairs associated with the eigenvalues nearest to $\tau \in \mathbb{C}$, it is possible to combine the subspace iterations with the shift-invert technique [70]. With shift-invert spectral transformation, the subspace iteration method will be applied to the matrix $(\mathcal{A} - \tau I)^{-1}$. Furthermore, a polynomial acceleration [136], can be used as a preconditioning technique to approximate eigenvalues near τ. This polynomial acceleration consists in applying a

given polynomial $\mathcal{P}$ to the matrix $\mathcal{A}$. In Algorithm 7 line 4 would change and become $W^{(k)} = \mathcal{P}(\mathcal{A})U^{(k)}$. The polynomial should act as a filter to damp eigencomponents in some undesired part of the spectrum. In our numerical example, to compute the eigenpairs associated with smallest magnitude eigenvalues, we will consider a Chebyshev polynomial of the first kind of degree $\tilde{n}$.

3.3.2.1 Chebyshev polynomial filter

Chebyshev polynomials are very useful to, at the same time, damp unwanted eigenvalues and magnify the wanted ones, corresponding to the filtering criterion. They are basically used to accelerate single vector iterations or projection processes. In **real arithmetic**, the Chebyshev polynomial of the first kind of degree $\tilde{n}$ is defined as:

$$C_{\tilde{n}}(t) = \begin{cases} cos(\tilde{n}\ cos^{-1}t), & |t| < 1, \\ cosh(\tilde{n}\ cosh^{-1}t), & |t| > 1. \end{cases} \tag{3.2}$$

Given an initial vector v, the Chebyshev polynomial sequence $w_j = C_j(\mathcal{A})v$ can be easily computed using the three-term recurrence:

$$\begin{aligned} &C_0(t) = 1, \\ &C_1(t) = t, \\ &C_{j+1}(t) = 2tC_j(t) - C_{j-1}(t),\ j = 1, 2, \ldots \end{aligned}$$

which leads to, $w_{j+1} = 2\mathcal{A}w_j - w_{j-1}$, with $w_0 = v$, and $w_1 = \mathcal{A}v$.

Theorem 1. *Let $[\alpha, \beta]$ be a non-empty interval in $\mathbb{R}$ and let γ be any real scalar such that $\gamma \geq \beta$. Then the minimum*

$$\min_{p \in \mathbb{P}_{\tilde{n}}, p(\gamma)=1} \max_{t \in [\alpha,\beta]} |p(t)|$$

is attained by the polynomial

$$\hat{C}_{\tilde{n}} \equiv \frac{C_{\tilde{n}}\left(1 + 2\frac{t-\beta}{\beta-\alpha}\right)}{C_{\tilde{n}}\left(1 + 2\frac{\gamma-\beta}{\beta-\alpha}\right)}.$$

Theorem 1 from [136] (see also [43]) shows that among all the possible polynomials of degree $\tilde{n}$, $\hat{C}_{\tilde{n}}$ reaches the smallest possible absolute values in the interval $[\alpha, \beta]$, such that $\hat{C}_{\tilde{n}}(\gamma) = 1$.

The definition of Chebyshev polynomials can be extended to **complex arithmetic** [136]. The Chebyshev polynomial of degree $\tilde{n}$ can still be computed using the three-term recurrence:

$$\begin{aligned} &C_0(z) = 1, \\ &C_1(z) = z, \\ &C_{j+1}(z) = 2zC_j(z) - C_{j-1}(z),\ j = 1, 2, \ldots, \tilde{n} - 1 \end{aligned}$$

The segment $[\alpha, \beta]$ in real arithmetic becomes an ellipse E in the complex plane which is symmetric with respect to the real axis, i.e., its center is on the real axis and its foci are either pure real or pure imaginary numbers (Figure 3.1).

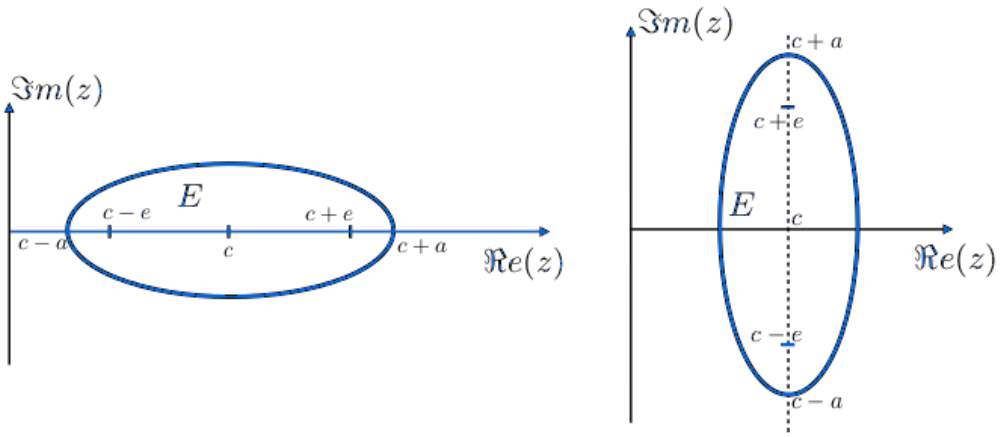


Figure 3.1 – Ellipses in the complex plane defined by their center c, foci $c-e$ and $c+e$ and major semi-axis a.

3.3.2.2 Subspace iteration with Chebyshev acceleration

This algorithm consists in using Chebyshev polynomials during the subspace iterations to accelerate the convergence of the method. Yousef Saad [136] shows that, if the ellipse $E(c,a,e)$ (see Figure 3.1) encloses the unwanted part of the spectrum of $\mathcal{A}$, then the asymptotically best min-max polynomial is the polynomial

$$p_{\tilde{n}}(\lambda) = \frac{C_{\tilde{n}}[(\lambda - c)/e]}{C_{\tilde{n}}[(\tau - c)/e]}, \tag{3.3}$$

where $C_{\tilde{n}}$ is the Chebyshev polynomial of degree $\tilde{n}$ of the first kind and τ is an approximation of the first wanted eigenvalue that is not enclosed within the ellipse E. Therefore, the successive applications of the polynomial defined by Equation (3.3) to a set of vectors U during the subspace iterations will make the space spanned by the columns of U converge to an invariant subspace corresponding to the eigenvalues that lay out of the ellipse E.

The computation of $w_{\tilde{n}} = \mathcal{P}_{\tilde{n}}(\mathcal{A})w_0$ is performed iteratively thanks to the three-term recurrence for Chebyshev polynomials [136] as:

1. Given the initial vector w_0, compute

$$\sigma_1 = \frac{e}{\tau - c},$$

$$w_1 = \frac{\sigma_1}{e}(\mathcal{A} - cI)w_0.$$

2. Iterate for $j = 1, \ldots, \tilde{n} - 1$:

$$\sigma_{j+1} = \frac{1}{2/\sigma_1 - \sigma_j},$$

$$w_{j+1} = 2\frac{\sigma_{j+1}}{e}(\mathcal{A} - cI)w_j - \sigma_j\sigma_{j+1}w_{j-1}.$$

Algorithm 8 implements the subspace iteration method with Chebyshev acceleration. The parameters c, a and e are defined by the ellipse E that encloses the unwanted part of the spectrum of $\mathcal{A}$. It requires some primary knowledge about the spectrum of $\mathcal{A}$.

Algorithm 8 Subspace Iteration with Chebyshev acceleration

1: Choose $U^{(0)} = [u_1^{(0)}, ..., u_m^{(0)}] \in \mathbb{C}^{n \times m}$
2: Orthonormalize $U^{(0)}$
3: **for** $k = 0, \ldots$ until convergence **do**
4: $\quad W_0^{(k)} = U^{(k)}$
5: $\quad \sigma_1 = \frac{e}{\tau - c}$
6: $\quad W_1^{(k)} = \frac{\sigma_1}{e}(\mathcal{A} - cI)U^{(k)}$
7: $\quad$ **for** j=2,. . . ,$\tilde{n}$ **do**
8: $\quad\quad \sigma_j = \frac{1}{2/\sigma_1 - \sigma_{j-1}}$
9: $\quad\quad W_j^{(k)} = 2\frac{\sigma_j}{e}(\mathcal{A} - cI)W_{j-1}^{(k)} - \sigma_{j-1}\sigma_j W_{j-2}^{(k)}$
10: $\quad$ **end for**
11: $\quad$ Orthonormalize $W_{\tilde{n}}^{(k)}$
12: $\quad$ Form the Rayleigh quotient $C^{(k)} = {W_{\tilde{n}}^{(k)}}^H \mathcal{A} W_{\tilde{n}}^{(k)}$
13: $\quad$ Compute the eigenvectors $G^{(k)} = [g_1, \ldots, g_m]$ of $C^{(k)}$ and eigenvalues $(\lambda_1, \ldots, \lambda_m)$
14: $\quad$ Update Ritz vectors: $U^{(k+1)} = W_{\tilde{n}}^{(k)} G^{(k)}$.
15: **end for**

Interpolation-restart policy: In the subspace iteration method depicted in Algorithms 7 and 8, according to Assumption 1 (see p. 37), the Ritz vectors $U^{(k)}$ are distributed, whereas the Rayleigh quotient $C^{(k)}$ and Ritz values are replicated. When a fault occurs, we distinguish two cases. During an iteration, a fault may occur before or after the computation of the Rayleigh quotient $C^{(k)}$.

1. When a fault occurs before the computation of the Rayleigh quotient $C^{(k)}$ (Algorithm 7, lines 2 to 5 and Algorithm 8, lines 3 to 12), surviving nodes cannot compute the Rayleigh quotient $C^{(k)}$ because entries of $W^{(k)}$ are missing. In this case, we consider the available entries of the Ritz vectors $U^{(k)}$ and its corresponding eigenvalues $\sigma(C^{(k-1)})$. We interpolate the m Ritz vectors individually $(u_\ell^{(m)}, 1 \leq \ell \leq m)$ using LI or LSI. In the particular case of Algorithm 8, all computation in the filtering step (line 5 to 10) are lost.

2. When a fault occurs after the computation of the Rayleigh quotient $C^{(k)}$ (Algorithm 7, line 6 to 7 and Algorithm 8, line 13 to 14), all surviving nodes can compute the entries of $U^{(k+1)}$ relying on a local replicate of $C^{(k)}$ and the local entries of $W^{(k)}$. The missing entries of each Ritz vector $(u_\ell^{(m)}, 1 \leq \ell \leq m)$, can be individually interpolated using LI or LSI relying on the corresponding eigenvalues $\sigma(C^{(k)})$.

After the interpolation, the subspace iteration algorithm is restarted with the matrix $U^{(IR)} = [u_1^{(IR)}, \ldots, u_m^{(IR)}] \in \mathbb{C}^{n \times m}$ until convergence.

Reset and ER policies: For the sake of comparison with the IR strategies, ER restarts with $U^{(k)}$ when the fault occurs before the computation of the Rayleigh quotient $C^{(k)}$, while it restarts with $U^{(k+1)}$ when the fault occurs after the computation of the Rayleigh $C^{(k)}$. It is important to notice that at every iteration where the fault occurs before the computation of $C^{(k)}$, ER delays by one iteration compared to NF. However at each iteration where the fault occurs after the computation of $C^{(k)}$, ER behaves exactly as NF. On the other hand, Reset restarts with $U^{(k)}$ or with $U^{(k+1)}$ when the fault occurs before the computation of $C^{(k)}$, respectively, with missing entries replaced by random values.

3.3.3 Arnoldi method to compute one eigenpair

Brief description of the numerical algorithm: The Arnoldi method is an efficient procedure for approximating eigenvalues lying at the periphery of the spectrum of $\mathcal{A}$. Its origin can be found in the work of Arnoldi, back in the 50's [11]. It is a generalization of the Lanczos algorithm designed for symmetric matrices [100]. After m steps, the method produces an upper Hessenberg matrix $H_m \in \mathbb{C}^{m\times m}$ that satisfies the Arnoldi relation:

$$\mathcal{A}V_m = V_m H_m + \beta_m v_{m+1} e_m^T, \tag{3.4}$$

where e_m denotes the last column of the $m \times m$ identity matrix. The Arnoldi method is attractive because it approximates the eigenpairs of a large matrix $\mathcal{A}$ from the eigenpairs of the small matrix H_m thanks to the Rayleigh-Ritz procedure. Due to memory limitation, the Arnoldi factorization is not expanded until convergence, but it is restarted after a search space of dimension m is built. When restarting, the Ritz vector u, associated with the targeted Ritz value, computed in V_m is used as initial guess.

Algorithm 9 Arnoldi method with restart m

1: Set the initial guess u.
2: **for** $k = 0, 1, \ldots,$ until convergence, **do**
3: $v_1 = u/\|u\|$
4: **for** $j = 1, \ldots, m$ **do**
5: $w_j = \mathcal{A}v_j$.
6: **for** $i = 1$ `to` j **do**
7: $h_{i,j} = v_i^T w_j$; $w_j = w_j - h_{i,j} v_i$.
8: **end for**
9: $h_{j+1,j} = \|w_j\|$.
10: If $|h_{j+1,j}| = 0$; $m = j$; goto 13. // happy breakdown.
11: $v_{j+1} = w_j / h_{j+1,j}$.
12: **end for**
13: Define the $m \times m$ upper Hessenberg matrix H_m.
14: Solve the eigenproblem $H_m g = \lambda g$.
15: Set Ritz vector $u = V_m g$.
16: **end for**

Interpolation-restart policy: According to Assumption 1 (see p. 37), we assume that the low dimensional Hessenberg matrix H_k is replicated on each node. Consequently, regardless of the step where the fault occurs during the iteration, each surviving node q can solve the eigenproblem $H_k g = \lambda g$ redundantly, then compute its entries of the Ritz vector $u_{I_q} = V_k(I_q, :)g$ (line 15 of Algorithm 9). The next step is the computation of the Ritz vector entries allocated on the failed node using LI or LSI. The resulting vector becomes a candidate to restart the Arnoldi iterations.

Reset and ER policies: In the case of the Arnoldi algorithm ER consists in enforcing the restart at the corresponding faulty iteration. In this particular case, the restart is performed from a Krylov basis of size k smaller than m. Reset proceeds as IR policies except missing entries of the current iterate are replaced by random values.

3.3.4 Implicitly restarted Arnoldi method to compute nev eigenpairs

Brief description of the numerical algorithm: Developed by Lehoucq and Sorensen in [105], the implicitly restarted Arnoldi method (IRAM) depicted in Algorithm 10 is commonly used with success for the solution of large eigenvalue problems. IRAM is an extension of the Arnoldi method. It starts with an Arnoldi equality expanded up to size m. After the expansion of the Arnoldi equality, IRAM performs a contraction of the Arnoldi equality from size m down to size $\tilde{m}$ ($nev \leq \tilde{m} < m$). This is achieved by applying a polynomial filter of degree $\ell = m - \tilde{m}$ that reduces the size of the Arnoldi equality down to a size $\tilde{m}$ (see Algorithm 10, line 12). The expansion and contraction steps are repeated until convergence. The contraction step has a key feature as it provides an efficient scheme to extract a set of eigenvalues in a target region of the spectrum from the Krylov subspace while maintaining the Arnoldi equality.

Interpolation-restart policy: When a fault occurs during an iteration, it may be during the expansion of the Krylov subspace (Algorithm 10, line 13) or during the contraction step (Algorithm 10, line 3 to 12). The contraction step is performed locally relying on the Hessenberg matrix H_m which is replicated. When a fault occurs during the contraction step each surviving node q can compute

$$V_{\tilde{m}}(I_q, :) = V_m(I_q, :)Q(:, 1 : \tilde{m}),$$

as well as the corresponding Hessenberg matrix

$$H_{\tilde{m}} = H_m(1 : \tilde{m}, 1 : \tilde{m}).$$

From $V_{\tilde{m}}(I_q, :)$ and $H_{\tilde{m}}$, the surviving nodes may then compute eigenvectors $G = [g_1, \ldots, g_{nev}]$ and eigenvalues $(\lambda_1, \ldots, \lambda_{nev})$ of $H_{\tilde{m}}$. Consequently, the entries of the Ritz vectors are computed by

$$U^{(k)}(I_q, :) = V_{\tilde{m}}(I_q, :)G. \tag{3.5}$$

Algorithm 10 Implicitly restarted Arnoldi method with restart m

1: Compute Arnoldi equality $\mathcal{A}V_m = V_m H_m + f_m e_m^T$.
2: **for** $k = 0, 1, \ldots,$ until convergence, **do**
3: Compute $\sigma(H_m)$ and select ℓ shifts $\mu_1, \ldots, \mu_\ell$ $(\ell = m - \tilde{m})$.
4: $Q = \mathcal{I}_m$
5: **for** $i = 1, \ldots, \ell$ **do**
6: QR Factorize $Q_i R_i = H_m - \mu_i I$
7: $H_m = {Q_i}^H H_m Q_i$
8: $Q = QQ_i$
9: **end for**
10: $\beta_{\tilde{m}} = H_m(\tilde{m}+1, \tilde{m})$
11: $f_{\tilde{m}} = v_{\tilde{m}+1}\beta_{\tilde{m}} + f_m Q(m, \tilde{m})$
12: $V_{\tilde{m}} = V_m Q(:, 1:\tilde{m})$; $H_{\tilde{m}} = H_m(1:\tilde{m}, 1:\tilde{m})$
13: Starting with $\mathcal{A}V_{\tilde{m}} = V_{\tilde{m}} H_{\tilde{m}} + f_{\tilde{m}} e_{\tilde{m}}^T$,
perform ℓ steps of Arnoldi algorithm to get $\mathcal{A}V_m = V_m H_m + f_m e_m^T$
14: **end for**

The missing entries I_p of Ritz vectors may be interpolated using either LI or LSI of the interpolation algorithms.

Let us consider the case when the fault occurs during the expansion step. Assuming that the fault occurs after the computation of an Arnoldi equality of size m_f with $\tilde{m} < m_f < m$. On the surviving nodes the contraction step is applied using the available entries of V_{m_f}. Using the replicated Hessenberg matrix H_{m_f}, each surviving node q may perform the QR iterations (see Algorithm 10, lines 3 to 9). The low dimensional matrix Q from the QR iteration allows one to perform the contraction step

$$V_{\tilde{m}}(I_q, :) = V_{m_f}(I_q, :)Q(:, 1:\tilde{m}),$$

and compute the corresponding Hessenberg matrix

$$H_{\tilde{m}} = H_{m_f}(1:\tilde{m}, 1:\tilde{m}).$$

The surviving nodes compute eigenvectors $G = [g_1, \ldots, g_{nev}]$ and eigenvalues $(\lambda_1, \ldots, \lambda_{nev})$ of $H_{\tilde{m}}$. The entries of the Ritz vectors are computed by Equation (3.5). The missing entries I_p of Ritz vectors may be interpolated using either LI or LSI.

For both cases, the available entries of $V_{\tilde{m}}$ do not longer satisfy the Arnoldi equality. To take into account all the available spectral information, we compute the linear combination of the interpolated eigenvectors, $u = \sum_{j=1}^{nev} u_j^{(IR)}$, and restart with the normalized linear combination $v_1 = \frac{u}{\|u\|}$ as initial vector. In exact arithmetic, it is well known that, starting from a vector v that is a linear combination of k eigenvectors, the Krylov sequence based on u terminates within k steps [82]. In terms of Arnoldi iteration, it means that the k eigenvectors would converge to the solution within the first iteration.

Reset and ER policies: The IRAM given in Algorithm 10 restarts with an Arnoldi equality of smaller size whereas the IR policies restart with the normalized linear combination of the interpolated eigenvectors. When using the Reset strategy, surviving nodes

use the schemes described for IR strategies and compute $U^{(k)}(I_q, :)$. The missing entries of the *nev* Ritz eigenvectors are replaced by random values and then the normalized linear combination of the resulting vectors is used as an initial guess. With ER, we compute all the entries of $U^{(k)}$ and we restart using the normalized linear combination of the *nev* approximate eigenvectors from $U^{(k)}$.

3.3.5 The Jacobi-Davidson method to compute *nev* eigenpairs

Brief description of the numerical algorithm: The Jacobi–Davidson method is a widely used eigensolver, especially for eigenpairs in the interior of the spectrum. It was proposed by Sleijpen and van der Vorst in [151]. It is a combination of Davidson's algorithm for the computation of a few smallest eigenpairs of symmetric matrices [49] with the Jacobi orthogonal component correction [91]. The basic ingredients of Jacobi-Davidson are presented in Algorithm 11 for the computation of one eigenpair whose eigenvalue is close to a given target τ. It starts with a given normalized vector v and constructs a basis V extended using the Jacobi orthogonal correction method. At each iteration the algorithm computes the Ritz pairs associated with V and selects the eigenpair whose eigenvalue is the closest to the target τ.

Algorithm 11 $[\lambda, u]$ = Basic-Jacobi-Davidson(v, τ)
Jacobi–Davidson algorithm to compute the eigenvalue of A closest to a target value τ

1: Set $V_1 = [v]$
2: **for** $k = 1, 2, \ldots$,until convergence **do**
3: Compute Rayleigh quotient: $C_k = V_k^H \mathcal{A} V_k$, and eigenpairs of C_k.
4: Select Ritz pair (λ_k, u_k) such that λ_k is the closest to τ
5: $r_k = \mathcal{A}u_k - \lambda_k u_k$
6: Perform a few steps of GMRES to solve the linear system

$$(I - u_k u_k^H)(\mathcal{A} - \tau I)(I - u_k u_k^H)v = -r_k, \text{ so that } v \perp u_k$$

7: Compute w by orthonormalizing v against V_k: $w = v - V_k(V_k^H v)$
8: Set $V_{k+1} = [V_k, w]$
9: **end for**

Algorithm 11 converges to one eigenpair. If more than one eigenpair needs to be computed, Algorithm 11 can be accommodated to compute a partial Schur decomposition of $\mathcal{A}$. In that respect, the next iterations are enforced to generate a search space orthogonal to the space spanned by the *nconv* already converged eigenvectors. This is achieved by representing this space using the corresponding Schur vectors. Let $\mathcal{A}Z_{nconv} = Z_{nconv}T_{nconv}$ denote the partial Schur form where the columns of the orthonormal matrix Z_{nconv} span the converged eigenspace and the diagonal of the upper triangular matrix T_{nconv} are the associated converged eigenvalues.

Algorithm 12 corresponds to the Jacobi-Davidson style QR algorithm presented in [67]. It is conceived to be used by a higher level routine that decides on the number of wanted

eigenpairs nev, the target point τ, the maximum and the minimum size of the basis V, etc.

The inputs for the algorithm are the existing converged Schur vectors Z_{nconv} of $\mathcal{A}$, the current size k of the basis V_k, $W_k = \mathcal{A}V_k$, and the Rayleigh quotient C_k. The focal point τ_0, the maximum size m affordable for V, the size of the restarted basis $\tilde{m}$ ($1 \leq \tilde{m} < m$) and the maximum number of restart allowed are also provided. Outputs are μ, z and t, such that

$$\mathcal{A}(Z_{nconv}\ z) = (Z_{nconv}\ z)\begin{pmatrix} T_{nconv} & t \\ 0 & \mu \end{pmatrix}$$

is a partial Schur decomposition of one higher dimension.

The higher level routine must furnish the necessary inputs to Algorithm 12. If the process starts from the beginning, there are then two situations. The first one corresponds to the case when the computation starts from a single random vector. Then the higher level routine computes an Arnoldi decomposition of size $\tilde{m}$

$$\mathcal{A}V_{\tilde{m}} = V_{\tilde{m}}H_{\tilde{m}} + \beta v_{\tilde{m}+1}e_{\tilde{m}}^T,$$

and Jacobi-Davidson starts with $U = [\ ]$, $V = V_{\tilde{m}}$, $W = \mathcal{A}V$ and $C = H_{\tilde{m}}$. The second case is when the process starts from a given number k of initial vectors. The initial block of vectors is then orthonormalized to obtain V_k and the process can start as indicated previously, with $Z = [\ ]$, $V = V_k$, $W = \mathcal{A}V_k$ and $C = V_k^H\mathcal{A}V_k = V_k^HW$.

Once a partial Schur form of size nev is available, the eigenpairs (λ_ℓ, u_ℓ) (with $\ell = 1, \ldots, nev$) of $\mathcal{A}$ can be extracted as follows. The eigenvalue λ_ℓ is the Ritz value of T_{nev} associated with the Ritz eigenvector g_ℓ so that $u_\ell = Z_{nev}g_\ell$.

Interpolation-restart policy: According to Assumption 1 (see p. 37), the Schur vectors $Z_{nconv} = [z_1, \ldots, z_{nconv}]$, and the basis $V_k = [v_1, \ldots, v_k]$ are distributed among the computing units as the matrix $T_{nconv} \in \mathbb{C}^{nconv \times nconv}$ and the Rayleigh quotient matrix $C_k \in \mathbb{C}^{k \times k}$ are replicated.

The Jacobi-Davidson algorithm enables more possibilities to regenerate a meaningful context for the restart after a fault. They are mainly two reasons. First, the algorithm does not rely on an equality that is incremented at each iteration such as Arnoldi; preserving such an incremental equality after a fault is unfeasible. Second, the algorithm can start from a set of vectors and its convergence will be fast if these vectors are rich in the sought spectral information.

When the fault occurs on node p while $nconv$ ($nconv > 0$) Schur vectors were converged, good approximations of the associated converged eigenvectors can easily be computed as follows. Each non-faulty node q performs

1. the spectral decomposition of the partial Schur matrix T_{nconv}

$$T_{nconv}G_{nconv} = G_{nconv}D \text{ with } G_{nconv} = [g_1, \ldots, g_{nconv}],$$

2. the computation of its entries of the converged eigenvectors

$$u_\ell(I_q) = Z_{nconv}(I_p, :)g_\ell \text{ for } \ell = 1, \ldots, nconv.$$

Algorithm 12 $[\mu, z, t]$=JDQR(Z(:,1:$nconv$),V(:,1:k),W(:,1:k),C(1:k,1:k), τ,$\tilde{m}$,m,$maxiter$)
Jacobi-Davidson style QR algorithm for expansion of partial Schur decomposition

Set $iter = 0$; $k_{init} = k$; $\tau = \tau_0$; $tr = 0$
while iter<maxiter **do**
  $iter = iter + 1$
  for $k = k_{init}, \ldots, m$ **do**
    % Computation of the Schur decomposition $CQ = QT$
    % so that the eigenvalues on the diagonal of T
    % are sorted by increasing distance to τ
    $[Q, T] = SortSchur(C(1:k, 1:k), \tau, k)$,
    Choose $\mu = T(1,1)$ and $g = Q(:,1)$, the Ritz pair closest to τ
    Approximate eigenvector of $\mathcal{A}$: $z = V(:, 1:k)g$, and $\mathcal{A}z$: $y = W(:, 1:k)g$
    Compute the residual $r = y - \mu z$, orthogonalize it against Z(:,1:$nconv$) and compute its norm: $rnorm$=norm(r)
    % Convergence test:
    if $rnorm$ is small enough **then**
      $nconv = nconv + 1$
      % Prepare outputs and deflate:
      $t = Z^H y$; $V = V(:, 1:k)Q(:, 2:k)$;
      $W = W(:, 1:k)Q(:, 2:k)$; $C = T(2:k, 2:k)$.
      return
    else if $k = m$ **then**
      % Restart:
      $V(:, 1:\tilde{m}) = V(:, 1:m)Q(:, 1:\tilde{m})$;
      $W(:, 1:\tilde{m}) = W(:, 1:m)Q(:, 1:\tilde{m})$;
      $C(1:\tilde{m}, 1:\tilde{m}) = T(1:\tilde{m}, 1:\tilde{m})$;
      $k_{init} = \tilde{m}$
    end if
    %No convergence reached and $k < m$.
    Solve the correction equation:

$$(I - zz^H)(\mathcal{A}_\perp - \tau I)(I - zz^H)v = -r$$

    Orthogonalize v against $V(:, 1:k)$ and $Z(:, 1:nconv)$
    % Extend the Rayleigh basis and the Rayleigh quotient:
    $V(:, k+1) = v$, $W(:, k+1) = \mathcal{A}v$, $C(k+1,1{:}k) = v^H W(:,1{:}k)$,
    $C(1{:}k, k+1) = V(:,1{:}k)^H W(:, k+1)$, $C(k+1, k+1) = v^H W(:, k+1)$
  end for
end while

The missing entries of the eigenvectors can be computed using IR to build $U_{nconv}^{(IR)} = [u_1^{(IR)}, \ldots, u_{nconv}^{(IR)}]$.

Let us assume that the partial Schur decomposition has converged in exact arithmetic

($\mathcal{A}Z_{nconv} = Z_{nconv}T_{nconv}$), and that the *nconv* eigenpairs also are exact solutions ($\mathcal{A}u_\ell = \lambda_\ell u_\ell$) in exact arithmetic. Under this assumption, the eigenvectors ($u_\ell^{(IR)}$) computed by IR are the same exact eigenvectors as long as $\left(\mathcal{A}(I_p, I_p) - \lambda_\ell I_{I_p,I_p}\right)$ is nonsingular or $\left(\mathcal{A}_{:,I_p} - \lambda_\ell \mathcal{I}_{:,I_p}\right)$ is full column rank for LI and LSI, respectively. As a consequence, if Jacobi-Davidson is restarted with the initial basis V_{nconv} obtained from the orthonormalization of the vectors of $U_{nconv}^{(IR)}$ then, the *nconv* already converged Schur vectors will be retrieved in the initial basis V_{nconv}.

Remark 3. *In floating point arithmetic, there is no guarantee to retrieve the already converged nconv Schur vectors by restarting with V_{nconv}, although this is likely to happen in practice.*

In addition to $U_{nconv}^{(IR)}$, further information can be extracted from the search space V_k and the Rayleigh quotient matrix C_k available when the fault occurs. Following the same methodology, spectral information built from C_k and V_k can be computed to generate additional directions to expand the initial search space ($U_{nconv}^{(IR)}$) used to restart the Jacobi-Davidson algorithm. Each non-faulty node q computes

1. the Schur decomposition $C_k\tilde{G}_k = \tilde{G}_k D_k$ so that the eigenvalues on the diagonal of D_k are sorted by increasing distance to τ, (Algorithm 12, line 5),
2. the entries of the Ritz vectors $\tilde{u}_\ell(I_q) = V_k(I_q, :)\tilde{g}_\ell$ for $\ell = 1, \ldots, s$, where s is the number of Ritz vectors we want to interpolate. Because $\tilde{G}_k$ has been sorted, these vectors may be considered as the s best candidates to expand $U_{nconv}^{(IR)}$. That is, $\tilde{u_1}$ is the Ritz vector associated with $D(1,1)$ which is the Ritz value the closest to the target τ, that is improved by Jacobi-Davidson iterations.

In addition, the missing entries I_p of the Ritz vectors $\tilde{u}_\ell$ can be computed using LI or LSI, $U^{(IR)} = [u_1^{(IR)}, \ldots, u_{nconv}^{(IR)}, \tilde{u}_1^{(IR)}, \ldots, \tilde{u}_s^{(IR)}]$. Once $U^{(IR)}$ has been computed, the vectors in $U^{(IR)}$ is then orthonormalized to obtain $V_{restart}$. The Jacobi-Davidson algorithm can be restarted with $Z = [\ ]$, $V = V_{restart}$, $W_{restart} = \mathcal{A}V_{restart}$, $C = V_{restart}^H W$. Although in principle s has only to satisfy $0 \leq s \leq k$, natural choices for s may be such that $nconv + s = nev$ (we interpolate nev vectors) or $s = nev$ (we interpolate $nconv + nev$ vectors). In general, s can be chosen depending on the computational cost one wants to invest.

Reset and ER policies: Reset and ER policies restart with the same amount of directions as IR policies. The key difference is that the lost entries are replaced with random entries in Reset case whereas ER restarts with the vectors computed from all the information available at iteration k.

3.4 Numerical experiments

In this section we investigate the numerical behavior of the eigensolvers in the presence of faults when the IR policies are applied. In Section 3.4.2, we present results for subspace

iteration methods with interpolation of all vectors of the subspace in presence of faults. We study the robustness of IR strategies when converging eigenvectors associated with both smallest and largest eigenvalues. We assess the robustness of our resilient Arnoldi procedure in Section 3.4.3, whereas Section 3.4.4 analyzes the robustness of the resilient algorithm designed for IRAM. In Section 3.4.5, we discuss results obtained for Jacobi-Davidson and the impact of different variants on the convergence behavior.

3.4.1 Experimental framework

We have performed extensive numerical experiments and only report here on qualitative numerical behavior observed on a few examples that are representative of our observations with the OP matrix described in Section 1.2.

Although many test matrices have been considered to evaluate the qualitative behavior of the resilient schemes, we only kept one example in this section that comes from thermo-acoustic instabilities calculation in combustion chambers. Indeed this test case exhibits many illustrative features. The matrix is unsymmetric, its spectrum lies in the right plane and it has small eigenvalues close to zero that can be computed using the different eigensolvers we have considered without shift invert techniques. Although its size is rather small, it exhibits numerical difficulties that are encountered on real life large scale problems [142, 143].

For a given figure (e.g, Figure 3.2a), faults are injected at the same iterations (e.g, 485 and 968) and during the same instructions for all cases (except NF, of course).

3.4.2 Resilient subspace iteration methods to compute *nev* eigenpairs

In this section, we analyze the robustness of the proposed resilient IR subspace iteration methods in the presence of faults. To analyze the robustness of our strategies, we simulate stressful conditions by increasing the fault rate and the volume of lost data. We present results for two variants of subspace iteration methods:

1. The subspace iteration with Chebyshev polynomial acceleration is used for the computation of the five eigenpairs corresponding to the smallest eigenvalues (Figure 3.2 and 3.3). In practice, a certain amount of information about the spectrum is needed in order to build the ellipse associated with the Chebyshev filter polynomial and the ellipse must be chosen as small as possible as long as it encloses the unwanted part of the spectrum. We mention that for this thermo-acoustic framework [142, 143] this prerequisite information is available.

2. The classical method for the computation of the five eigenpairs corresponding to the largest magnitude eigenvalues (Figure 3.4 and 3.5).

For both calculations (five largest and five smallest eigenvalues), we report the maximum of the individual scaled residual norms (defined by Equation (3.1)) of the five Ritz pairs at

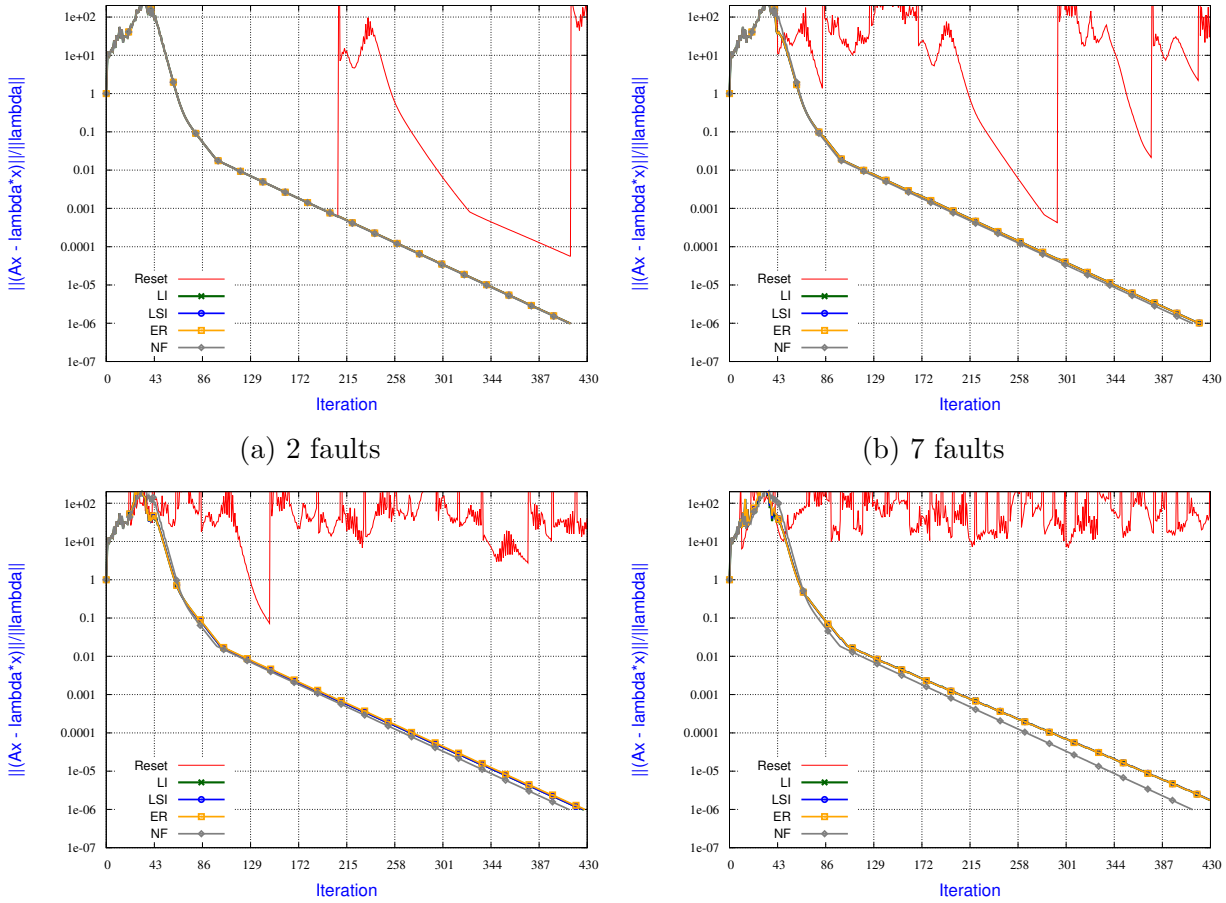
each iteration. The execution ends when the five Ritz pairs satisfy the stopping criterion, i.e, when the maximum of the scaled residual norms is lower than or equal to the selected threshold $\epsilon = 10^{-6}$. When converging the Ritz pair associated with the smallest Ritz value, the scaled residual norm increases during the first iterations before starting to decrease slowly and reaching the target threshold in 420 iterations in the NF case (Figure 3.2 and 3.3). The Reset strategy strongly penalizes the convergence at each fault, even with a very small fault rate (Figure 3.2a). When the fault rate is relatively large (Figure 3.2d), this strategy is likely to stagnate completely, and does exhibit large convergence peak even when a little amount of data is lost as it can be observed in Figure 3.3 when the amount of lost data ranges from 0.2% to 6%.

Contrary to Reset, both LI and LSI are extremely robust and resilient. Indeed, regardless of the number of faults and the volume of lost data, LI and LSI almost consistently overlap with ER and NF, except in the presence of a very large fault rate (Figure 3.2d). However, the resilience capability of LI and LSI is preserved because they overlap with ER when varying either the fault rate (Figure 3.2) or the volume of lost data (Figure 3.3).

When converging the Ritz pair associated with the largest eigenvalues, NF converges in 485 iterations as depicted in Figure 3.4 and 3.5. The Reset strategy again exhibits large peaks in the residual norm after each fault, but it can this time converge when only a few faults occur (Figure 3.4a) or only a little amount of data is lost (Figure 3.5). Regarding the robustness of both IR strategies, the convergence histories of LI and LSI again almost consistently overlap the NF curve regardless of the fault rate (Figure 3.4) or the volume of lost data (Figure 3.5).

3.4.3 Arnoldi method to compute one eigenpair

In this section we assess the robustness of our resilient Arnoldi for computing the eigenpair associated with the largest eigenvalue in magnitude. We choose a restart parameter $m = 7$ (see Algorithm 9). One iteration consists of building a Krylov basis of size m followed by the approximation of the desired eigenpair. When a fault occurs during the building of the Krylov subspace, we apply the interpolation strategies to the faulty subspace (an Arnoldi equality of size smaller than m) to regenerate the initial guess for the next iteration. Because this computation requires only a few (outer) iterations we consider one single fault rate and we focus on the impact of proportion of lost data. Because the fault rate is constant the number of faults displayed in the different plots might differ depending on the convergence penalty they induce for the different resilient strategies. We report the convergence histories in Figure 3.6. NF converges smoothly in 9 (outer) iterations (index k in Algorithm 9) whereas Reset strategy exhibits a large peak in the scaled residual norm after each fault. When the amount of lost data is low (0.2% in Figure 3.6a), the Reset penalty remains reasonable (2 extra iterations), but it becomes more significant (7 extra iterations) if that amount increases (6% in Figure 3.6d). Because ER has to restart more often than NF, its convergence history exibits some delay compared to the one of NF. On the other hand, both IR strategies are again extremely robust. Indeed, LI and LSI convergence coincide with ER, regardless the proportion of lost data (Figure 3.6). Note that if the proportion of lost data is very large (Figure 3.6d), LI and LSI may slightly differ from ER. The fact that LI and



(a) 2 faults

(b) 7 faults

(c) 15 faults

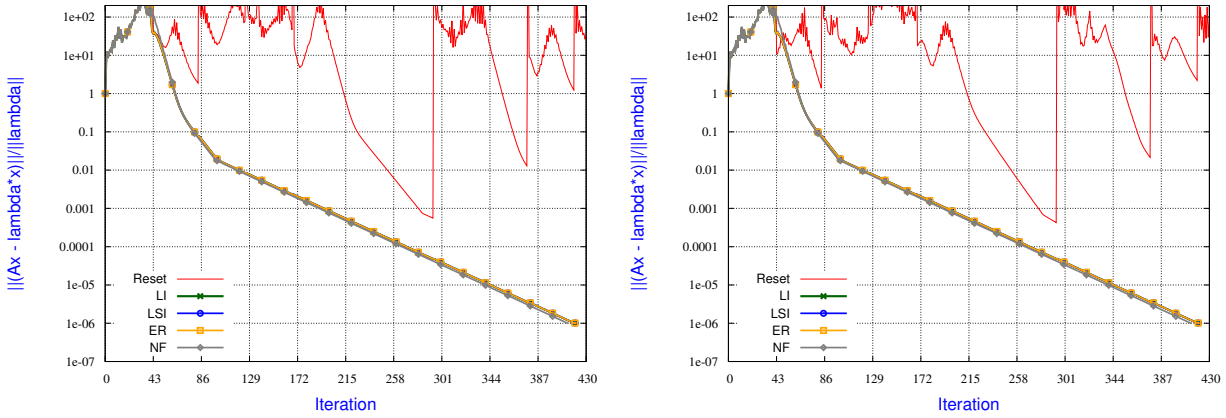
(d) 39 faults

Figure 3.2 – **Impact of the fault rate** on the resilience of IR strategies when converging the **five eigenpairs** associated with the **smallest eigenvalues** using subspace iteration method accelerated with **Chebyshev**. A proportion of 0.8 % of data is lost at each fault. Note that LI, LSI, ER and NF almost coincide.

LSI convergence coincide with ER, indicates that the spectral information regenerated by the LI and LSI is as good as the one computed by the regular solver.

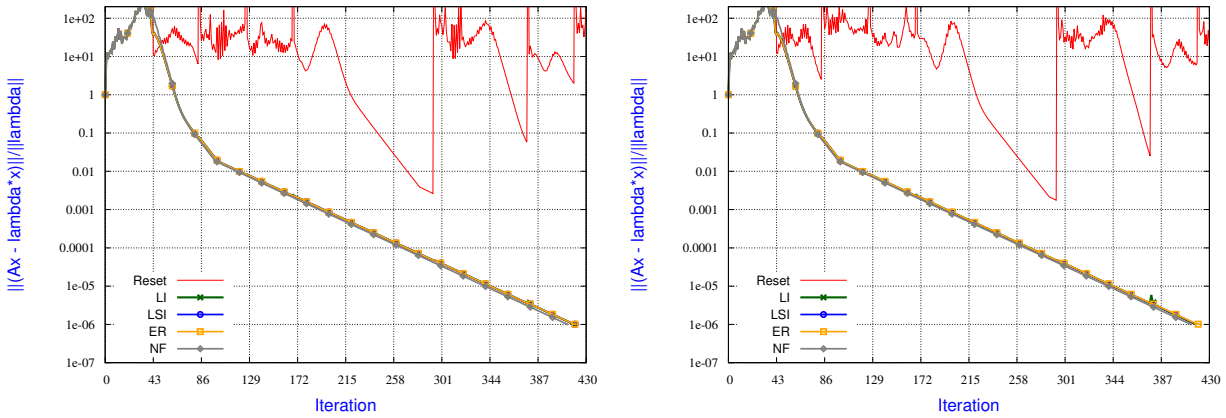
3.4.4 Implicitly restarted Arnoldi method to compute *nev* eigenpairs

To investigate the robustness of IR strategies designed for IRAM, we compute the five eigenpairs ($nev = 5$) that correspond to the largest magnitude eigenvalues. At each iteration, we report the maximum of the scaled residual norms of those five sought eigenpairs. We consider a restart parameter m=10 (see Algorithm 10). One iteration thus consists of building a Krylov subspace of size 10, followed by the computation of the approximate eigenpairs. If the eigenpairs do not satisfy the stopping criterion, the next iteration starts with a contracted Arnoldi equality of size $\tilde{m} = 5$.



(a) 0.2% of lost data

(b) 0.8% of lost data



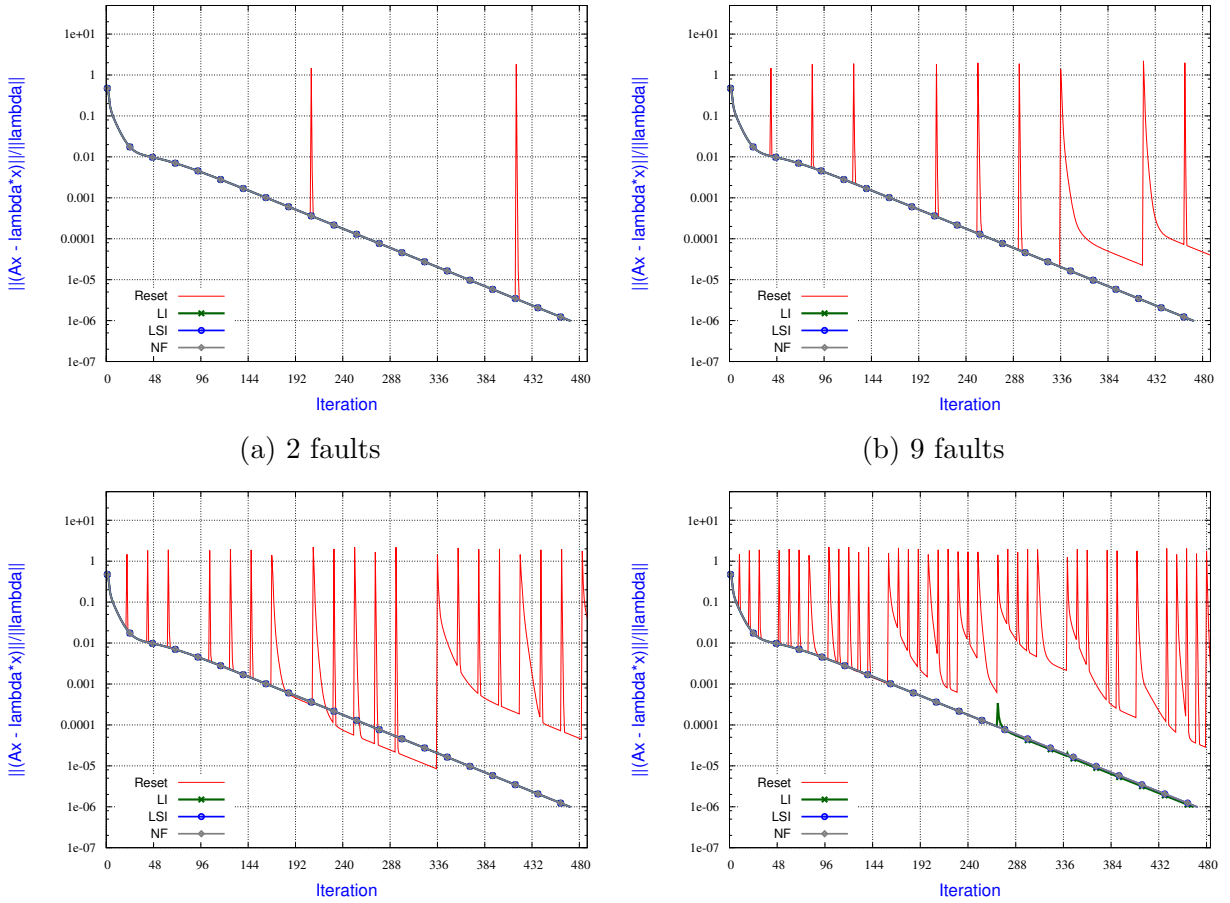
(c) 3% of lost data

(d) 6% of lost data

Figure 3.3 – **Impact of the amount of lost data** on the resilience of IR strategies when converging the **five eigenpairs** associated with the **smallest eigenvalues** using subspace iteration method accelerated with **Chebyshev**. The volume of lost data varies from 0.2% to 6% whereas the fault rate is constant (7 faults). LI, LSI, ER and NF coincide.

The NF calculation computes the five sought eigenvectors in 11 (outer) iterations (index k in Algorithm 10). The Reset strategy exhibits large peak in the scaled residual norm after each fault, its scaled residual norm increases further than the initial one. As a consequence, convergence is very much delayed. Furthermore Reset is also sensitive to the amount of lost data. The larger the volume of lost data, the more its convergence is delayed (Figure 3.7).

On the other hand, both IR strategies are much more robust than Reset. However, they still require a few more iterations than NF. Because they almost consistently coincide with ER, it can be concluded that this slight penalty is not due to the quality of interpolation but to the necessity of restarting with the information of the five dimension space compressed in one single direction.



(a) 2 faults

(b) 9 faults

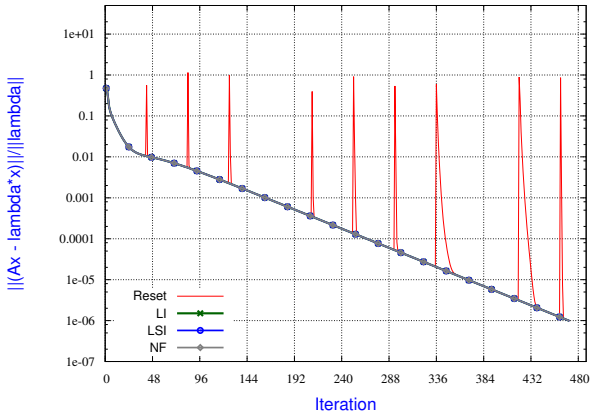
(c) 19 faults

(d) 36 faults

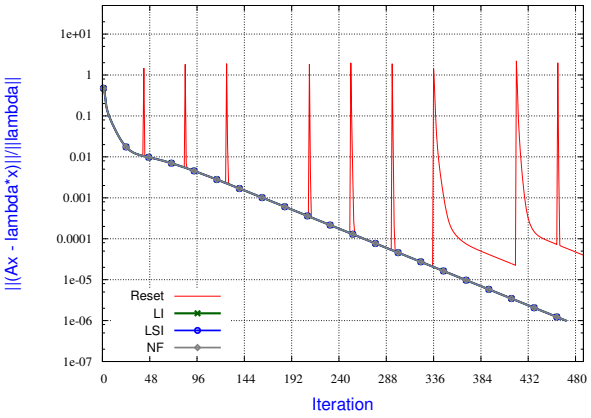
Figure 3.4 – **Impact of the fault rate** on the resilience of IR strategies when converging the **five eigenpairs** associated with the **largest eigenvalues** using the **basic subspace iteration method**. A proportion of 0.8 % of data is lost at each fault. LI, LSI and NF coincide in (a), (b) and (c).

3.4.5 Jacobi-Davidson method to compute *nev* eigenpairs

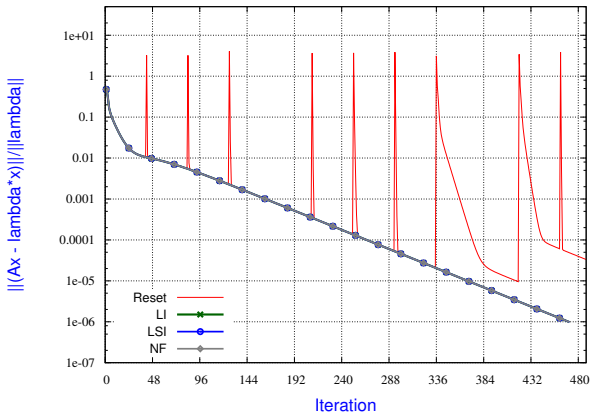
In this section, we investigate the resilience of the IR strategies designed for Jacobi-Davidson. In all the experiments, we seek for the five ($nev = 5$) eigenpairs whose eigenvalues are the closest to zero ($\tau = 0$). To facilitate the readability and the analysis of the convergence histories ploted in this section, we use vertical green lines to indicate the convergence of new eigenpair (such as iterations 95, 130, 165, 200 and 242 in Figure 3.8a), and vertical red lines to indicate faulty iterations (such as iterations 148 and 228 for the sixth and ninth fault, respectively, in Figure 3.8a). According to Remark 3, although it is very likely to happen, there is no guarantee to retrieve all the already converged Schur vectors in the basis used to restart. As a consequence, we indicate the number of Schur vectors retrieved in the basis used to restart in red color under the vertical red line corresponding to the associated fault. For instance, 2 already converged Schur vectors are immediately retrieved at restart, after the fault at iteration 148 in Figure 3.8a.



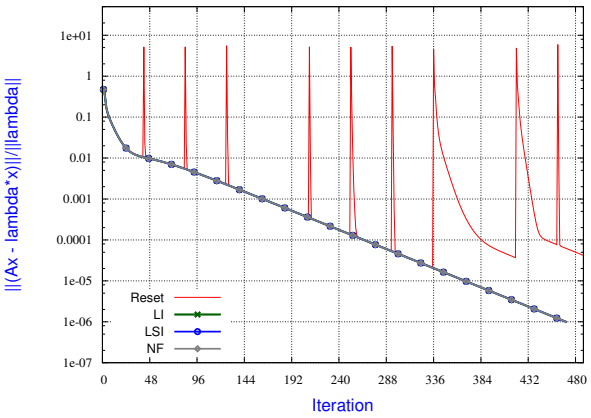
(a) 0.2% lost data



(b) 0.8% lost data



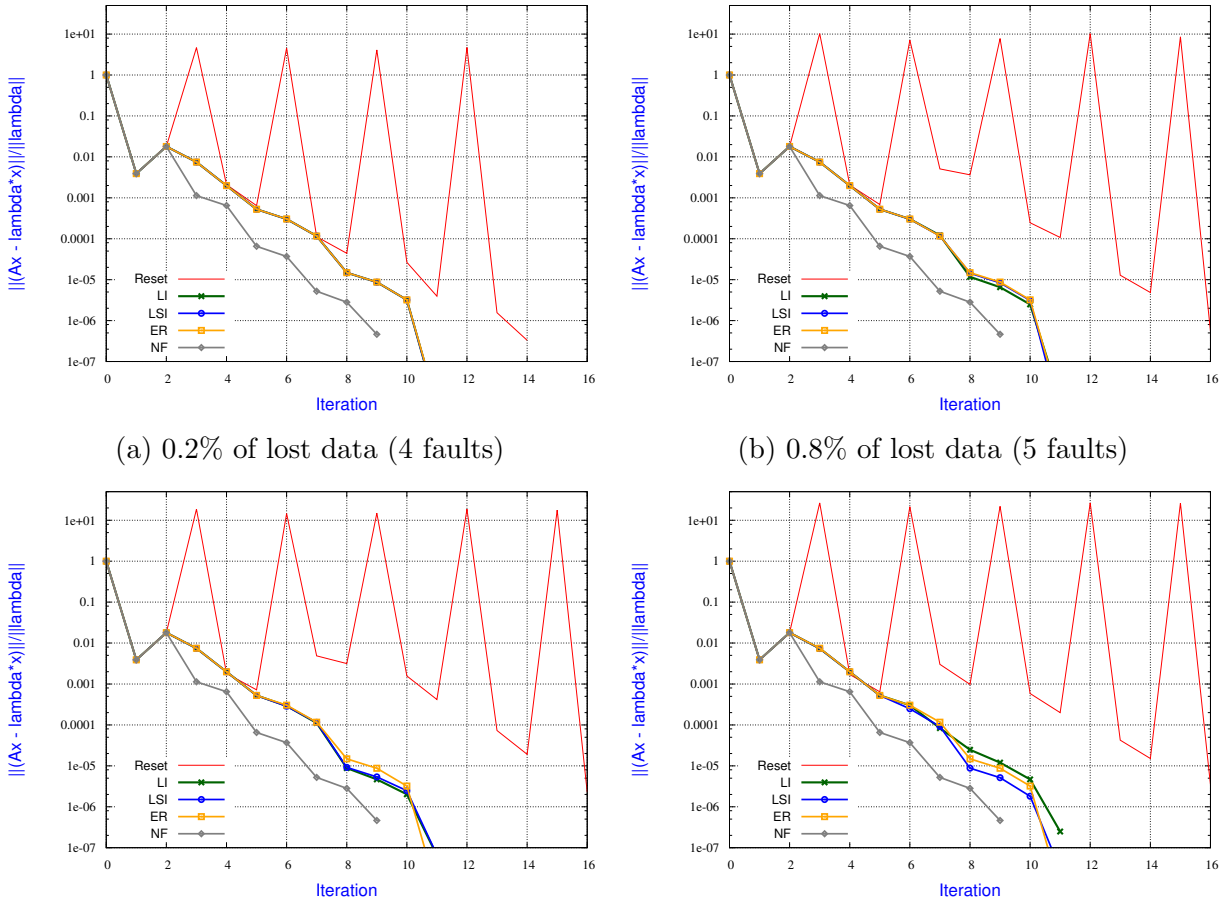
(c) 3% lost data



(d) 6% lost data

Figure 3.5 – **Impact of the amount of lost data** on the resilience of IR strategies when converging the **five eigenpairs** associated with the **largest eigenvalues** using **basic subspace iteration method**. The volume of lost data varies from 0.2% to 6% whereas the fault rate is constant (9 faults). LI, LSI and NF coincide.

In the Jacobi-Davidson method there is some flexibility to select the number of vectors (i.e., the dimension of the space generated for restarting) that can be interpolated after a fault; that are the converged Schur vectors as well as a few of the best candidates for Schur vectors extracted from the search space V_k. Because many possibilities exist, the results of the investigations are structured as follows. In Section 3.4.5.1, we first study the robustness of the IR approaches using a fixed dimension equal to nev for the space for restarting the Jacobi-Davidson method after a fault. As the number of converged eigenpairs $nconv$ becomes close to nev we observed that the restart behaves poorly for reasons that will be explained. Consequently in Section 3.4.5.2, we consider restarting after a fault with a space of dimension $nconv+nev$ vectors to account for the partial convergence. Finally, we consider an hybrid approach in Section 3.4.5.3 where the IR approach is apply to regenerate $nev - 1$ (or $nev + nconv - 1$) vectors while considering the best candidate for the Schur vector that was computed in V_k before the fault occurs (in a real parallel implementation this latter vector could be for instance checkpointed).



(a) 0.2% of lost data (4 faults)

(b) 0.8% of lost data (5 faults)

(c) 3% of lost data (5 faults)

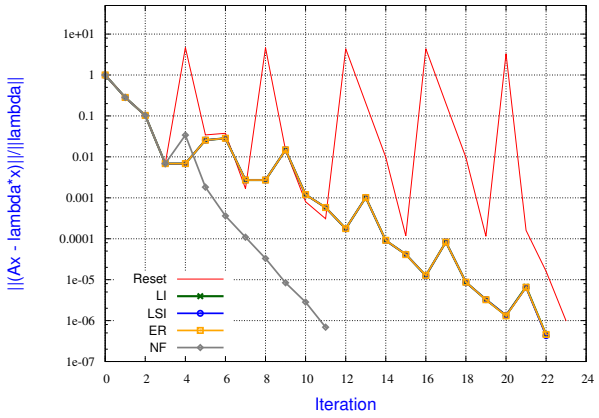
(d) 6% of lost data (5 faults)

Figure 3.6 – **Impact of the amount of lost data** on the resilience of IR strategies when converging the eigenpair associated with the **largest eigenvalue** using **Arnoldi method**. LI, LSI and NF coincide in (a) (b) and (c).

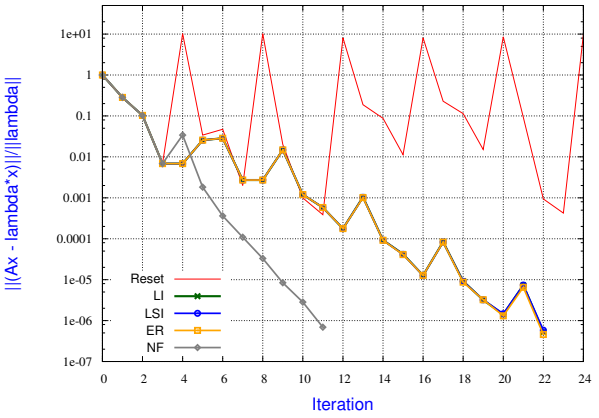
For the calculation of the five smallest eigenvalues, the NF algorithm converges in 210 iterations while faulty executions have extra iterations. For the sake of comparison, we consider only the first 300 iterations of all the runs so that the graphs will have exactly the same scales and range of iteration count.

3.4.5.1 Interpolation-restart strategy using *nev* regenerated vectors

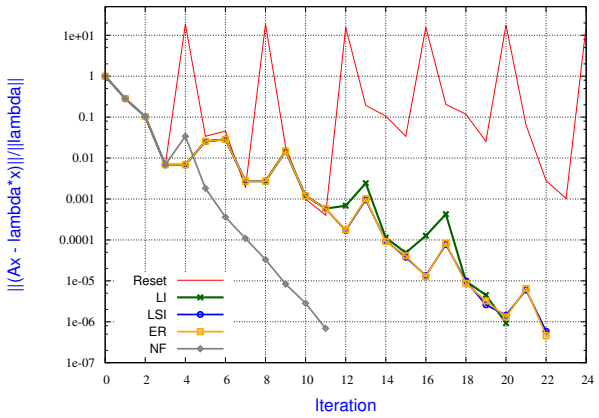
We first consider all the different restarting strategies. We report in Figure 3.8 their convergence histories in different subplots for a fixed fault rate to evaluate the quality of the basis used for the restarts. The curves in Figure 3.8a show the impact of the enforced restarts (35 additional iterations for ER compared to NF) and will serve as reference to evaluate the quality and relevance of the interpolated *nev* directions considered for LI, LSI and Reset. The first comment goes to the Reset approach that completely fails and is unable to compute any eigenpair. The LI interpolation behaves much better but only succeeds to compute four out of the five eigenpairs in 300 iterations. For this example, LSI is slightly



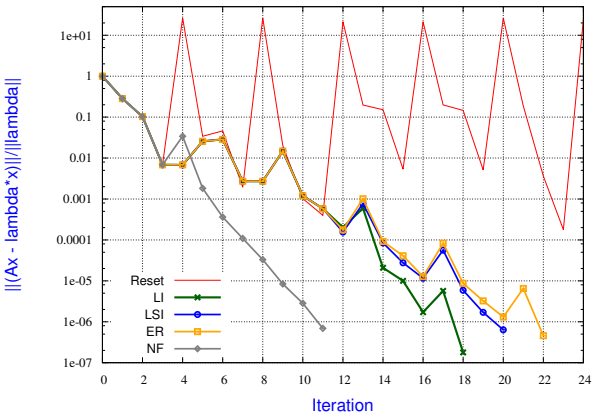
(a) 0.2% of lost data (5 faults)



(b) 0.8% of lost data (6 faults)



(c) 3% of lost data (6 faults)

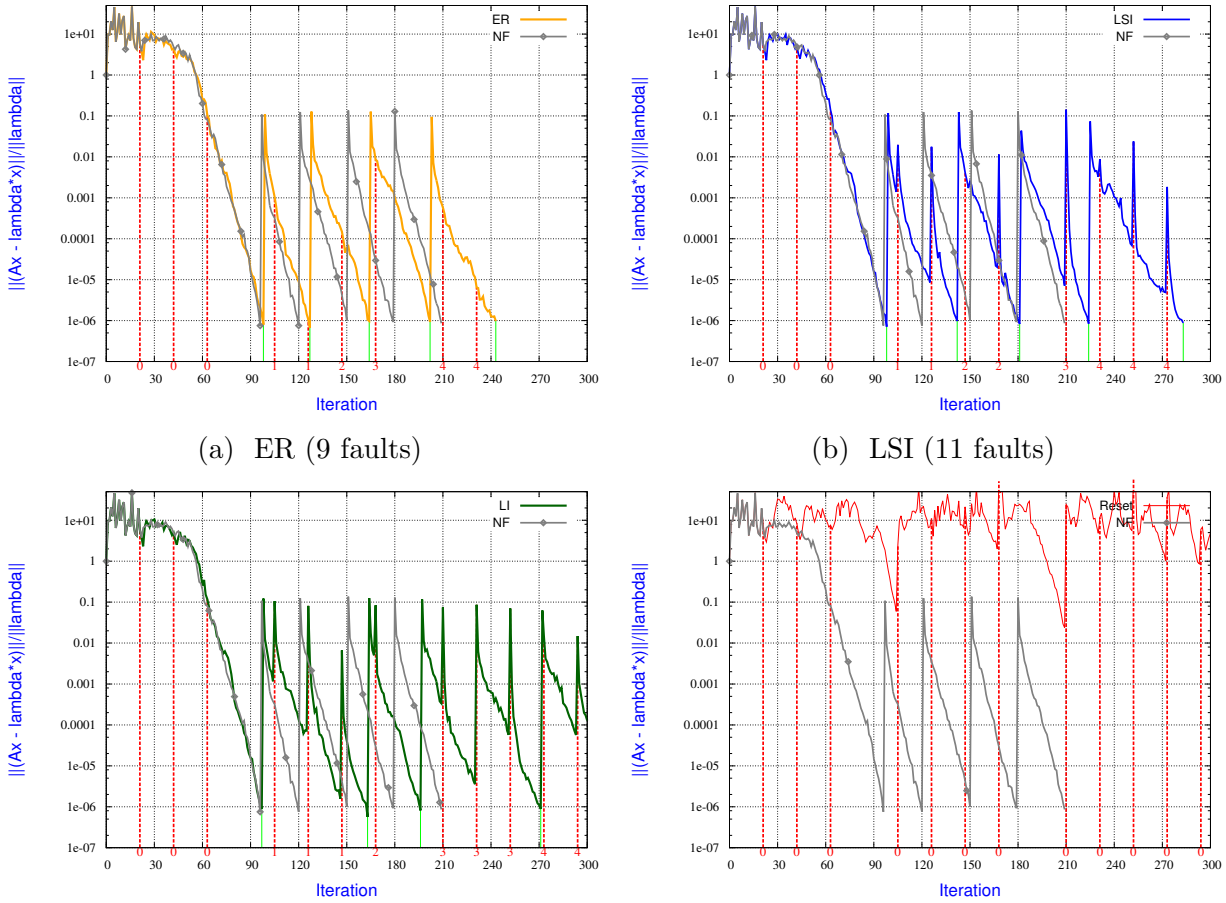


(d) 6% of lost data (6 fautls)

Figure 3.7 – **Impact of the amount of lost data** on the resilience of IR strategies when converging the **five eigenpairs** with **largest eigenvalues** using **IRAM**. The volume of lost data varies from 0.2% to 6% whereas the fault rate is constant. LI, LSI and ER coincide in (a) and (b).

more robust and computes the five eigenpairs with a few extra iterations compared to ER. For both IR approaches it can be observed that the converged Schur vectors are recomputed by the first step of the Jacobi-Davidson method that is the Raleigh quotient procedure. As it is for instance illustrated in Figure 3.8c and 3.8b, when a fault occurs, the *nconv* converged Schur vectors before a fault are found immediately from the interpolated vectors at restart (e.g, in Fig 3.8b, the value 4 under the vertical red line at the faulty iteration 275 means that all four Schur vectors converged before the fault are immediately rediscovered). Finally we notice that even if, in Figure 3.8, experiments are performed with the same fault rate, the number of faults varies depending on the interpolation strategy. Indeed, the less robust the strategy, the more the convergence is delayed, which increases the probability for additional faults to occur.

In order to reduce the number of graphs to illustrate the impact of the amount of lost data (see Figure 3.9) then the influence of the fault rate (see Figure 3.10), we only consider the LSI approach in the rest of this section. The influence of the volume of lost data is



(a) ER (9 faults)

(b) LSI (11 faults)

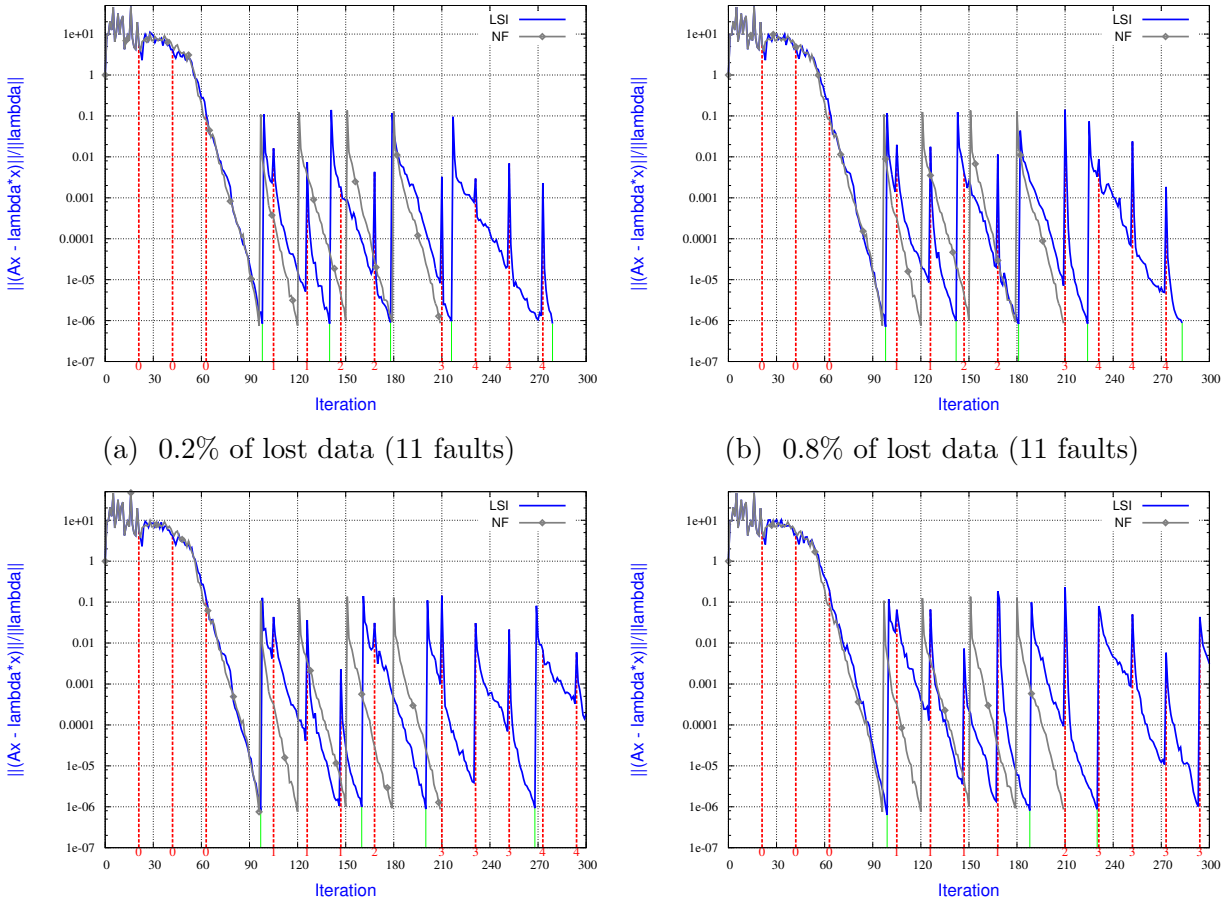
(c) LI (12 faults)

(d) Reset (12 faults)

Figure 3.8 – **Comparison of IR strategies** using **nev** regenerated vectors when converging the **five eigenpairs** associated with the **smallest eigenvalues** using **Jacobi-Davidson**. The fault rate is the same for all strategies and a proportion of 0.8 % data is lost at each fault.

displayed in Figure 3.9 where its proportion varies from 0.2 % to 6 % while the fault rate is constant. As it could have been expected the general trend is: the larger the amount of lost data, the larger the convergence penalty. A large amount of lost data at each fault prevents LSI from converging the five eigenpairs; only four (three) are computed for 3 % lost (6 % repectively). Allowing for more than the 300 iterations would enable the solver to succeed the calculation of the five eigenpairs.

Finally, in Figure 3.10 we depict the convergence histories of the various restarting strategies when the fault rate varies leading to a number of faults that varies from 3 to 26; as expected the larger the number of faults, the larger the convergence delay. However, the IR policy is rather robust and succeds to converge the five eigenpairs in less than 300 iterations except for the 26 faults case. On that latter example, one can observe that the converged Schur vectors are always recovered at restart after a fault; futhermore the number of faulty steps to converge a new eigenpair tends to increase when the number of converged eigenpairs increases. This phenomenum is due to the fact that we always interpolate *nev*



(a) 0.2% of lost data (11 faults)

(b) 0.8% of lost data (11 faults)

(c) 3% of lost data (12 faults)

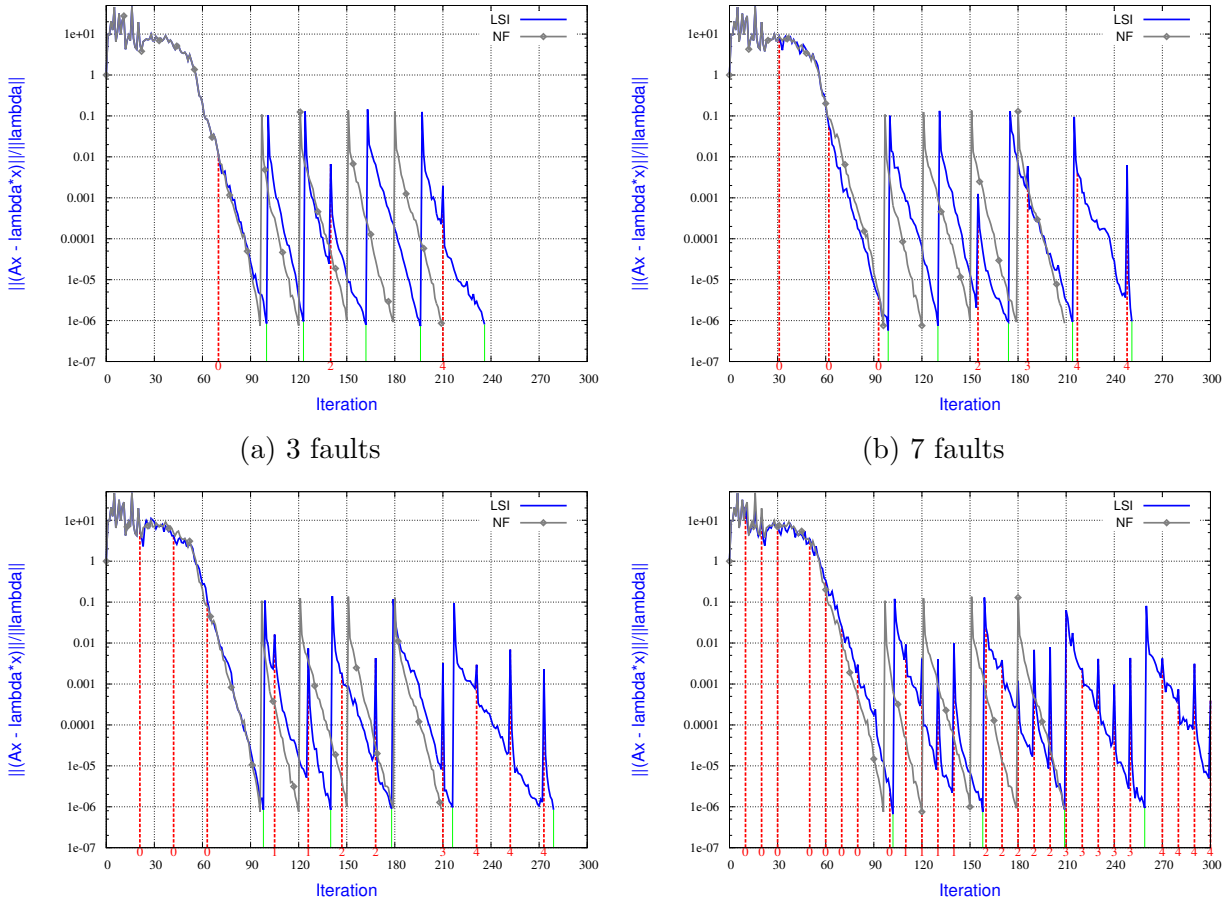
(d) 6% of lost data (12 faults)

Figure 3.9 – **Impact of the amount of lost data** on the resilience of **LSI** using **nev** regenerated vectors when converging the **five eigenpairs** associated with the **smallest eigenvalues** using **Jacobi-Davidson**. The volume of lost data varies from 0.2% to 6% whereas the fault rate is constant. All converged Schur vectors are found immediately after interpolation followed by restart.

directions independently of the number of Schur vectors that have converged. Because the *nconv* eigenpairs are immediately recovered in the space of dimension *nev* used to restart after the fault, the dimension of the actual starting space used for the subsequent Jacobi-Davidson method schrinks. Consequently this space V is less rich in spectral information which delays the convergence of the next eigenpair.

3.4.5.2 Interpolation-restart strategy using $(nev + nconv)$ regenerated vectors

The lack of complete convergence observed in Figure 3.9c, 3.9d and 3.10d might be due to the smaller space dimension actually used after a fault because *nconv* Schur vectors are extracted from the *nev* interpolated directions. We consider in this section larger interpolated space to account for the already converged Schur vectors when the fault occurs. More precisely, instead of using a fixed dimension space to restart after a fault, we make



(a) 3 faults

(b) 7 faults

(c) 11 faults

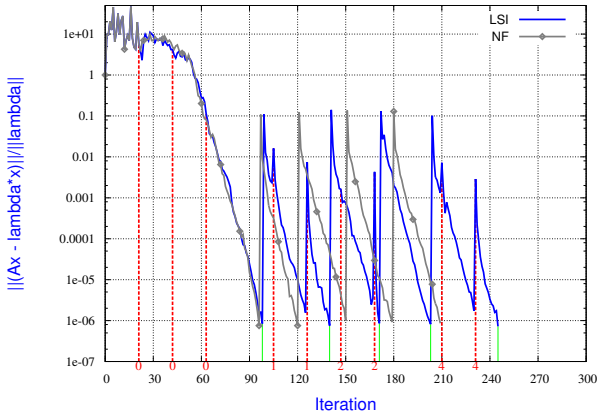
(d) 26 faults

Figure 3.10 – **Impact of the fault rate** on the resilience of **LSI** using **nev** regenerated vectors when converging the **five eigenpairs** associated with the **smallest eigenvalues** using **Jacobi-Davidson**. The fault rate varies whereas a proportion of 0.2 % of data is lost at each fault. At each fault, all the already converged Schur vectors are retrieved in the basis of restart.

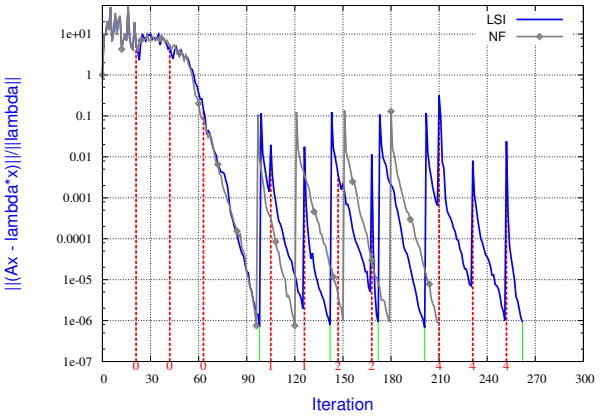
it of size $nev + nconv$. Similarly to the previous section, we only consider the LSI restart strategy.

Figure 3.11 shows the convergence history when the amount of lost data varies. Compared with Figure 3.9, the benefit of using larger space to restart the current Jacobi-Davidson search is clear as the convergence is improved.

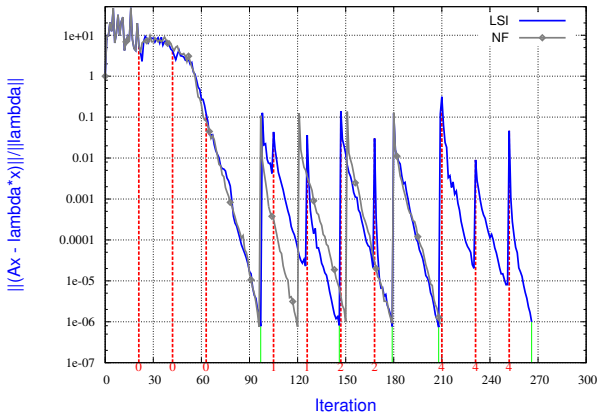
Similar observations can be made, when the fault rate is increased. Comparing results displayed in Figure 3.12 and Figure 3.10, it can be seen that the increase of the dimension of the space used to restart makes Jacobi-Davidson more robust.



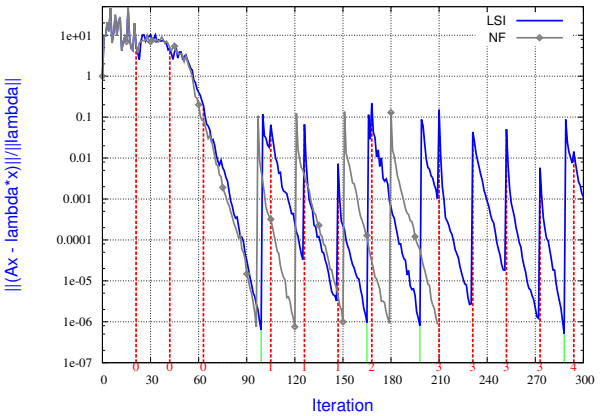
(a) 0.2% of lost data (9 faults)



(b) 0.8% of lost data (10 faults)



(c) 3% of lost data (10 faults)

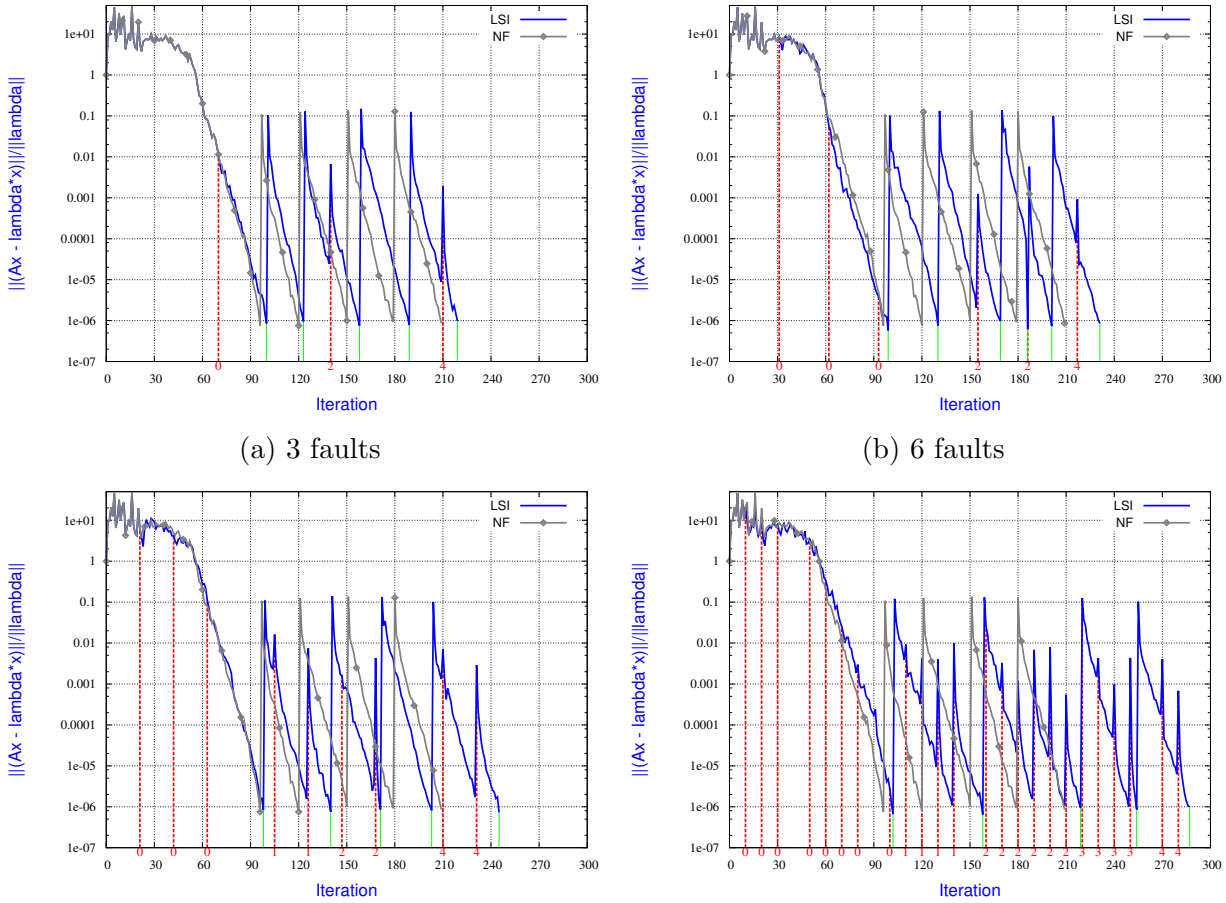


(d) 6% of lost data (12 faults)

Figure 3.11 – **Impact of the amount of lost data** on the resilience of **LSI** using $(nev + nconv)$ regenerated vectors when converging the **five eigenpairs** associated with the **smallest eigenvalues** using **Jacobi-Davidson**. The volume of lost data varies from 0.2% to 6% whereas the fault rate is constant. All converged Schur vectors are found immediately after interpolation followed by restart.

3.4.5.3 Hybrid check-pointing/interpolating approach

Despite the increase of the amount of recovered data, the peak of the residual norm associated with the current Schur vector persists after each fault. A possible remedy consists in using an hybrid approach where we interpolate the $nconv$ Schur vectors while furthermore reusing the best candidate Schur vector available in V_k (as if we had checkpointed this single direction) when the fault occurs and interpolate $nev - 1$ directions but the first to recover additional meaingingfull spectral information from V_k. This procedure is beneficial to improve the convergence independently of the size of the space used for the restart. It is best highlighted in the most difficult situations that are large amount of lost data (see Figure 3.13) and large number of faults (see Figure 3.14). In these figures, the convergence of the scaled residual norm does no longer exhibit peak after the faults when the best Schur candidate vector is used in the set of directions for the restart. The most relevant information on the next Schur vector to converge seems to be concentrated in the current best



(a) 3 faults

(b) 6 faults

(c) 9 faults

(d) 24 faults

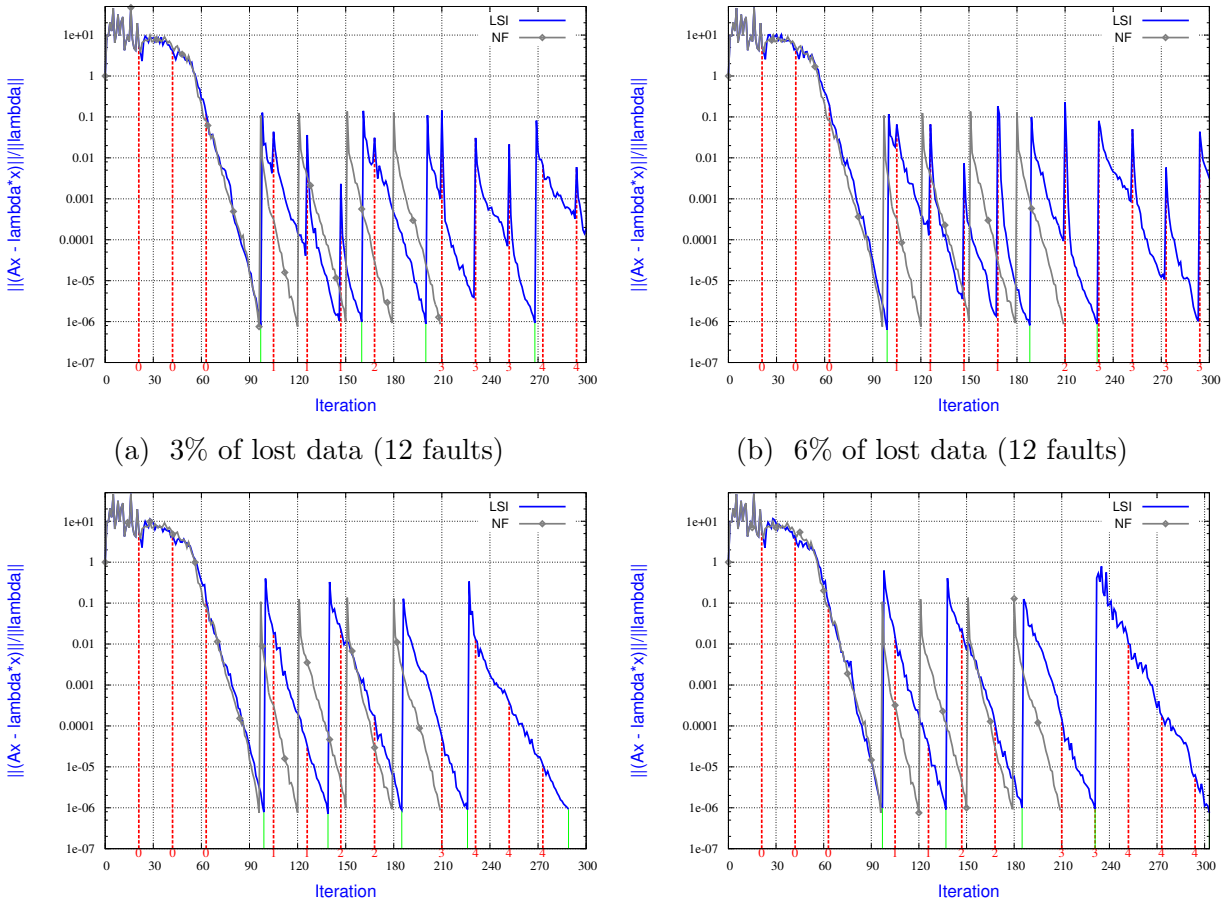
Figure 3.12 – **Impact of the fault rate** on the resilience of **LSI** using $(nev + nconv)$ regenerated vectors when converging the **five eigenpairs** associated with the **smallest eigenvalues** using **Jacobi-Davidson**. The fault rate varies whereas a proportion of 0.2 % of data is lost at each fault. At each fault, all the already converged Schur vectors are retrieved in the basis of restart.

candidate Schur vector. Intensive experiments show that the interpolation strategies do not succeed to regenerate accurately the lost entries of this special direction of V_k.

A reasonable trade-off may consist in checkpointing the current Schur vector to increase the robustness of the resilient Jacobi-Davidson method while keeping a low overhead when no fault occurs.

3.5 Concluding remarks

Many scientific and engineering applications require the computation of eigenpairs of large sparse matrices. The objective of the chapter was to study numerical schemes to design resilient parallel eigensolvers. For that purpose, we have investigated IR approaches to interpolate lost data relying on non corrupted data and tuned restart mechanisms to cope



(a) 3% of lost data (12 faults)

(b) 6% of lost data (12 faults)

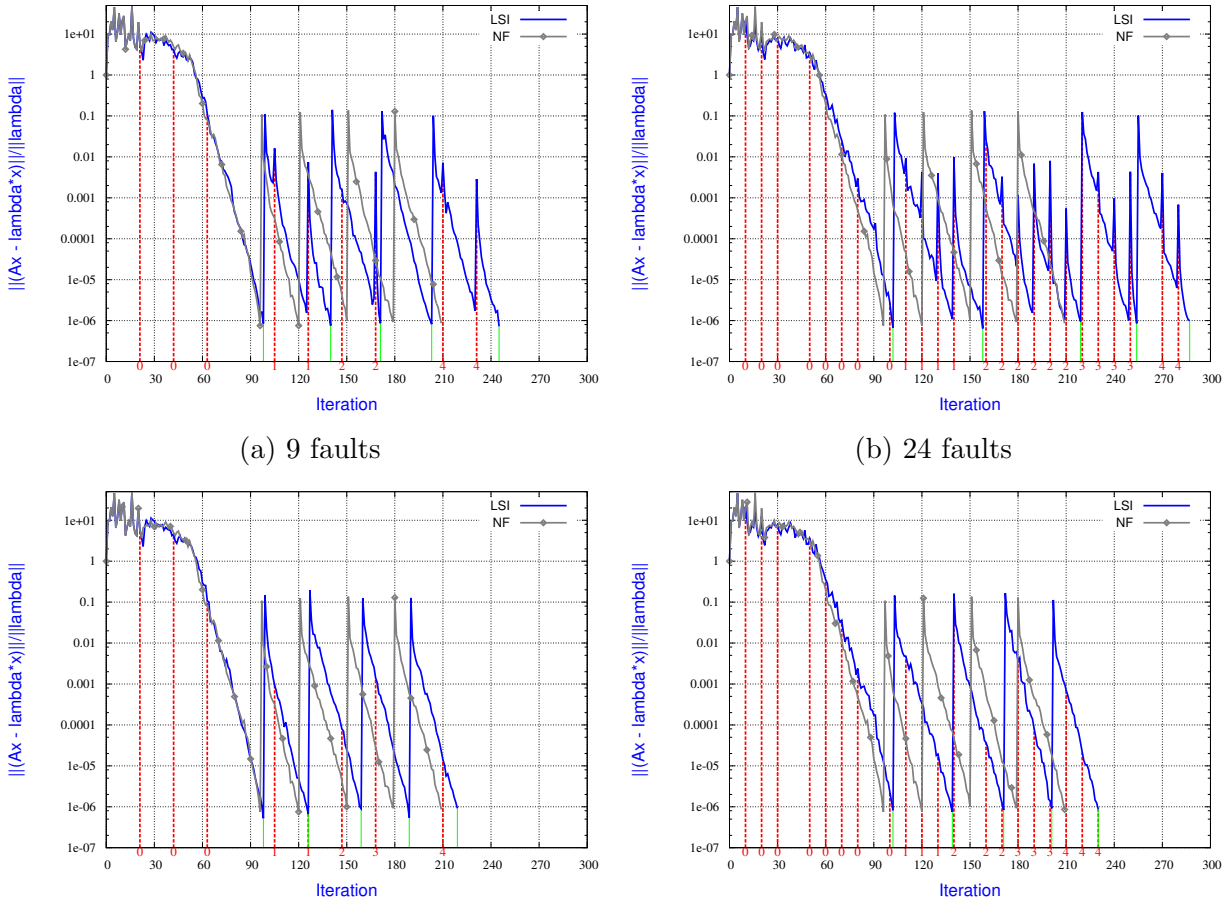
(c) 3% of lost data

(d) 6% of lost data

Figure 3.13 – Impact of **keeping the best Schur vector candidate** in the search space after a fault combined with $nev + nconv - 1$ interpolated directions. Top two plots nev interpolated directions, bottom two plots $nev - 1$ interpolated directions plus the best Schur candidate. Calculation of the **five eigenpairs** associated with the **smallest eigenvalues** using **Jacobi-Davidson**. The fault rate is constant over all subfigures. The proportion of lost data is either 3% on Figure (a) and (c) or 6% on Figure (b) and (d). All converged Schur vectors are found immediately after interpolation followed by restart.

with numerical features of the individual eigensolvers. To evaluate the qualitative behavior of the resilient schemes, we have simulated stressful conditions by increasing the fault rate and the volume of lost data.

We have considered two variants of subspace iteration methods. On the one hand, we have considered the subspace iteration with Chebyshev polynomial acceleration for the computation of the five eigenpairs corresponding to the smallest eigenvalues. On the other hand, we have considered the classical method for the computation of the five eigenpairs corresponding to the largest magnitude eigenvalues. For both methods, the Reset strategy strongly penalizes the convergence at each fault, while both LI and LSI are extremely robust and resilient, regardless of the number of faults and the volume of lost data. The same numerical behavior is observed for our resilient Arnoldi for computing the eigenpair



(a) 9 faults

(b) 24 faults

(c) 8 faults

(d) 20 faults

Figure 3.14 – Impact of **keeping the best Schur vector candidate** in the search space after a fault combined with $nev + nconv - 1$ interpolated directions. Top two plots $nev + nconv$ interpolated directions, bottom two plots $nev + nconv - 1$ interpolated directions plus the best Schur candidate. Calculation of the **five eigenpairs** associated with the **smallest eigenvalues** using **Jacobi-Davidson** with a proportion of 0.2 % of data is lost at each fault.

associated with the largest eigenvalue in magnitude. Our resilient IRAM for the computation of the five eigenpairs that correspond to the largest magnitude eigenvalues are much more robust than Reset. However, they exhibit a slight penalty not due to the quality of interpolation but to the necessity of restarting with the information of the five dimension space compressed in one single direction.

In this chapter, we have had a higher emphasis on the Jacobi-Davidson method for the computation of the five eigenpairs corresponding to the smallest eigenvalues. The motivation is twofold: the Jacobi-Davidson method is widely used in many real-life applications, and in addition, it offers some flexibility to select the number of vectors that can be interpolated after a fault. Relying on that flexibility, we have designed IR approaches that interpolate the *nconv* converged Schur vectors and a various number of the best candidates Schur vectors extracted from the search space V_k. We have first studied the robustness of

the IR approaches using a fixed dimension equal to *nev* for the space for restarting the Jacobi-Davidson method after a fault. With a restart based on *nev* vectors, the Reset approach completely fails and is unable to compute any eigenpair, while both IR approaches are robust. In addition, when a fault occurs, the *nconv* converged Schur vectors before a fault are found immediately from the interpolated vectors at restart for both IR approaches. However when the number of converged eigenpairs *nconv* becomes close to *nev* we have observed that the restart behaves poorly. We have noticed that this phenomenon is due to the fact that we always interpolate *nev* directions independently of the number of Schur vectors that have converged. Because the *nconv* eigenpairs are immediately recovered in the space of dimension *nev* used to restart after the fault, the dimension of the actual starting space used for the subsequent Jacobi-Davidson method shrinks. Consequently the space used to restart is less rich in spectral information which delays the convergence of the next eigenpair. To overcome this penalty, we have increased the number of best Schur candidate vectors extracted from the search space V_k interpolated. The resulting numerical experiments demonstrated that the convergence behavior significantly improves while increasing the number of interpolated vectors used to restart after a fault.

Finally we have observed that despite the increase of the amount of recovered data, the peak of the residual norm associated with the current Schur vector persists after a fault. For a possible remedy of these effects, we have designed a hybrid approach that consists in interpolating the *nconv* Schur vectors as well as reusing (as if we had checkpointed this single direction) the best candidate Schur vector available in V_k when the fault occurs and interpolate $nev - 1$ directions but the first to recover additional meaningful spectral information from V_k. This last procedure has improved the convergence whatever the size of the space used for the restart. This hybrid strategy based on a combination of light checkpointing and a numerical resilience has a very low overhead when no fault occurs.

We have also investigated the potential of IR strategies to design resilient eigensolvers. This study shows that even adapted to eigenvalue problems, IR strategies exhibit a level of numerical robustness as high as observed in the context of linear systems. Moreover, the interesting numerical behavior obtained for the hybrid approach suggests that a reasonable trade-off may consist in a combination of IR strategies and light checkpointing.

Part II

Application of Interpolation-restart Strategies to a Parallel Linear Solver

Chapter 4

Resilient MaPHyS

Contents

4.1 Introduction

In Part I, we have investigated some IR strategies to design resilient parallel sparse Krylov subspace methods. This part aims at extending these IR approaches in the context of a parallel sparse linear solver designed for processing large problems at scale. For that purpose, we use the Massively Parallel Hybrid Solver (MaPHyS[1]) [4, 75, 83], a fully-featured sparse hybrid (direct/iterative) solver. MaPHyS is based on non-overlapping domain decomposition techniques applied in a fully algebraic framework that leads to the iterative solution of a so-called condensed linear system defined on the interface between the algebraic subdomains (subset of equations). As discussed in Section 1.2 of the general introduction, direct

[1] https://project.inria.fr/maphys/

methods present an excellent numerical robustness but their large memory footprint may prevent them to process very large sparse linear systems. On the other hand, iterative methods, which require a much lower memory usage may fail to converge on certain types of linear systems. Relying on domain decomposition ideas, more precisely a Schur complement approach, hybrid methods aim at combining the robustness of direct methods and the lower memory footprint of iterative schemes. After performing a domain partitioning, unknown associated with the interior and with the interface of the domains are processed with a direct and an iterative method, respectively.

MaPHyS mainly consists of four steps: *(1)* graph partitioning, *(2)* local factorization of interiors and computation of the local Schur complement, *(3)* computation of the preconditioner and *(4)* the solve step which itself consists of two sub-steps: *(4a)* iterative solve of the linear system associated with the interface and *(4b)* back substitution on the interiors. Assuming that there exist fault tolerant strategies for the first three steps, we focus on designing a resilient Krylov solver for the solution of the reduced system on the interface (step (4a)).

We consider the fault model introduced in Section 2.2 of Chapter 2 which distinguishes three categories of data: *computational environment*, *static data* and *dynamic data*. In this chapter, we assume that static data are all data available before the iterative solution step. Furthermore, we assume that the Schur complement, the preconditioner and the right-hand side are static, while the current iterate and any other data generated during the step *(4a)* are dynamic data. We recall that a fully resilient strategy must provide mechanisms to change any failed process, processor, core or node as well as strategies to retrieve the computational environment and lost data. However in this thesis we focus on numerical strategies to retrieve meaningful dynamic data. For this purpose, we assume that there is a mechanism to replace lost computational resources, restore the computational environment and load static data (for instance from disk). Having said that, the problem we focus on is: *How IR strategies can be adapted to make the iterative solve step of* MaPHyS *resilient?* In this context, we simulate fault by overwriting dynamic data because according to our assumption, computational resources and static data are assumed to be recovered by a separate fault tolerant mechanism.

When a single fault occurs, we exploit data redundancy intrinsic to MaPHyS to retrieve all lost dynamic data. In the rest of this chapter, this case is termed *single process fault*. When faults are simultaneously injected into two neighbor processes, the strategy based on exploiting data redundancy cannot be applied anymore. We call this *neighbor processes fault case*. We use an IR strategy to survive neighbor processes fault.

To assess the effectiveness of our schemes, we use four different matrices described in Table 4.1. `Matrix211` is a non-symmetric matrix of dimension $801,378$ from the fusion energy study code M3D-C.[2]. It important to notice that with matrix `Matrix211`, iterative methods may suffer from slow convergence [109]. `Haltere` is a symmetric complex non-Hermitian matrix of dimension $1,288,825$ from a 3D electromagnetism problem. On the other hand, `Tdr455k` is a non-symmetric matrix of dimension $2,738,556$ from numerical simulation of an accelerator cavity design [102]. `Tdr455k` is a highly-indefinite linear system which exhibits an effective numerical difficulty. Matrix `Nachos4M` is of order 4M and comes

[2]Center for Extended MHD Modeling (CEMM) URL: http://w3.pppl.gov/cemm/

from a numerical simulation of the exposure of a full body to electromagnetic waves with discontinuous Galerkin method [61].

Matrix	`Matrix211`	`Haltere`	`Tdr455k`	`Nachos4M`
N	801K	1,288K	2,738K	4M
Nnz	129,4M	10,47M	112,7M	256
Symmetry	non-symmetric	symmetric	non-symmetric	non-symmetric

Table 4.1 – Description of the matrices considered for experimentation.

The remainder of this chapter is organized as follows: Section 4.2 explains how Schur complement methods are used to designed hybrid (direct/iterative) solutions for large sparse linear systems. In Section 4.3 we explain the recovering techniques used to survive faults. In Section 4.4 we present the numerical experiments, discuss the robustness and overhead of the resilient Krylov solver designed for MaPHyS followed with some conclusions and perspectives in Section 4.5.

4.2 Sparse hybrid linear solvers

To achieve a high scalability algebraic domain decomposition methods are commonly employed to split a large size linear system into smaller size linear systems. To achieve this goal, the Schur complement method is often used to design sparse hybrid linear solvers [76, 131, 166].

4.2.1 Domain decomposition Schur complement method

This section describes how to rely on the Schur complement method to solve linear systems. Let us assume that the problem is subdivided in subdomains. We distinguish two types of unknowns: the interior unknowns $x_\mathcal{I}$ and the interface unknowns x_Γ. With respect to such a decomposition, the linear system $\mathcal{A}x = b$ in the corresponding block reordered form reads:

$$\begin{pmatrix} \mathcal{A}_{\mathcal{I}\mathcal{I}} & \mathcal{A}_{\mathcal{I}\Gamma} \\ \mathcal{A}_{\Gamma\mathcal{I}} & \mathcal{A}_{\Gamma\Gamma} \end{pmatrix} \begin{pmatrix} x_\mathcal{I} \\ x_\Gamma \end{pmatrix} = \begin{pmatrix} b_\mathcal{I} \\ b_\Gamma \end{pmatrix}. \tag{4.1}$$

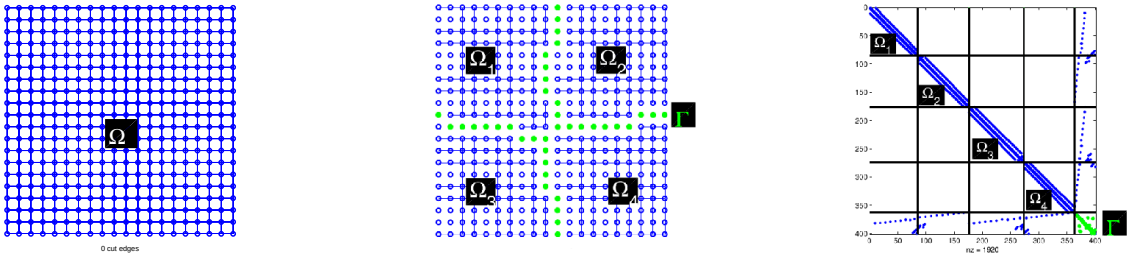
Eliminating $x_\mathcal{I}$ from the second block row of Equation (4.1) leads to the reduced system

$$\mathcal{S}x_\Gamma = f, \tag{4.2}$$

where

$$\mathcal{S} = \mathcal{A}_{\Gamma\Gamma} - \mathcal{A}_{\Gamma\mathcal{I}}\mathcal{A}_{\mathcal{I}\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I}\Gamma} \quad \text{and } f = b_\Gamma - \mathcal{A}_{\Gamma\mathcal{I}}\mathcal{A}_{\mathcal{I}\mathcal{I}}^{-1}b_\mathcal{I}. \tag{4.3}$$

The matrix $\mathcal{S}$ is referred to as the *Schur complement* matrix. This reformulation leads to a general strategy for solving (4.1). A sparse direct method is used to apply $\mathcal{A}_{\mathcal{I}\mathcal{I}}^{-1}$ and form (4.2). This latter system associated with the Schur complement is solved with an



(a) Graph representation. (b) Domain decomposition. (c) Block reordered matrix.

Figure 4.1 – Domain decomposition into four subdomains $\Omega_1, \ldots, \Omega_4$. The initial domain Ω may be algebraically represented with the graph G associated with the sparsity pattern of matrix $\mathcal{A}$ (a). The local interiors $\mathcal{I}_1, \ldots, \mathcal{I}_N$ form a partition of the interior $\mathcal{I} = \sqcup \mathcal{I}_p$ (blue vertices in (b)). They interact with each others through the interface Γ (red vertices in (b)). The block reordered matrix (c) has a block diagonal structure for the variables associated with the interior $\mathcal{A}_{\mathcal{I}\mathcal{I}}$.

iterative method on which we focus. Once x_Γ is known, $x_\mathcal{I}$ can be computed with one additional direct back-solve step.

Domain decomposition methods are often referred to as substructuring schemes. This terminology comes from structural mechanics where non-overlapping domain decomposition were first developed. The structural analysis finite element community has been and still is heavily involved in the design and development of these techniques. Not only is their definition fairly natural in a finite element setting but their implementation can preserve data structures and concepts already present in large engineering software packages. For the sake of simplicity, we assume that $\mathcal{A}$ is symmetric in pattern and we denote $G = \{V, E\}$ the adjacency graph associated with $\mathcal{A}$. In this graph, each vertex is associated with a row or column of the matrix $\mathcal{A}$. There exists an edge between the vertices p and q if and only if the entry $a_{p,q}$ is non zero. Figure 4.1a shows such an adjacency graph.

From the finite element viewpoint, a non-overlapping decomposition of a domain Ω into subdomains $\Omega_1, \ldots, \Omega_N$ corresponds to a *vertex split* of the graph G. V is decomposed into N subsets $V_1, \ldots, V_N$ of interiors $\mathcal{I}_1, \ldots, \mathcal{I}_N$ and boundaries $\Gamma_1, ..., \Gamma_N$ (algebraic view). Figure 4.1b depicts the algebraic view of the domain decomposition into four subdomains.

Local interiors are disjoint and form a partition of the interior $\mathcal{I} = \sqcup \mathcal{I}_p$ (blue vertices in Figure 4.1b). Two subdomains Ω_p and Ω_q may share part of their interface ($\Gamma_p \bigcap \Gamma_q \neq \emptyset$), such as Ω_1 and Ω_2 in Figure 4.1b which share eleven vertices. Altogether the local boundaries form the overall interface $\Gamma = \cup \Gamma_p$ (red vertices in Figure 4.1b), which is thus not nessarily a disjoint union. One may note that the local interiors and the (global) interface form a partition of the original graph: $V = \Gamma \bigsqcup \sqcup \mathcal{I}_p$ (the original graph in Figure 4.1a is exactly covered with blue and red points in Figure 4.1b).

Because interior vertices are only connected to vertices of their subset (either on the interior or on the boundary), matrix $\mathcal{A}_{\mathcal{I}\mathcal{I}}$ associated with the interior has a block diagonal structure, as shown in Figure 4.1c. Each diagonal block $\mathcal{A}_{\mathcal{I}_p\mathcal{I}_p}$ corresponds to a local interior. On the other hand, to handle shared interfaces with a local approach, the coefficients on the interface may be weigthed so that the sum of the coefficients on the local interface

submatrices are equal to one. To that end, we introduce the *weighted local interface* matrix $\mathcal{A}^w_{\Gamma_p\Gamma_p}$ which satisfies $\mathcal{A}_{\Gamma\Gamma} = \sum_{p=1}^{N} \mathcal{R}^T_{\Gamma_p} \mathcal{A}^w_{\Gamma_p\Gamma_p} \mathcal{R}_{\Gamma_p}$, where $\mathcal{R}_{\Gamma_p} : \Gamma \to \Gamma_p$ is the canonical point-wise restriction which maps full vectors defined on Γ into vectors defined on Γ_p. For instance, the ten points on the red interface shared by subdomains Ω_1 and Ω_2 in Figure 4.1b may get a weight $\frac{1}{2}$ as they are shared by two subdomains. In matrix terms, a subdomain Ω_p may then be represented by the *local matrix* $\mathcal{A}_p$ defined by

$$\mathcal{A}_p = \begin{pmatrix} \mathcal{A}_{\mathcal{I}_p\mathcal{I}_p} & \mathcal{A}_{\mathcal{I}_p\Gamma_p} \\ \mathcal{A}_{\Gamma_p\mathcal{I}_p} & \mathcal{A}^w_{\Gamma_p\Gamma_p} \end{pmatrix}. \tag{4.4}$$

The global Schur complement matrix S from (4.3) can then be written as the sum of elementary matrices

$$\mathcal{S} = \sum_{p=1}^{N} \mathcal{R}^T_{\Gamma_p} \mathcal{S}_p \mathcal{R}_{\Gamma_p}, \tag{4.5}$$

where

$$\mathcal{S}_p = \mathcal{A}^w_{\Gamma_p\Gamma_p} - \mathcal{A}_{\Gamma_p\mathcal{I}_p} \mathcal{A}^{-1}_{\mathcal{I}_p\mathcal{I}_p} \mathcal{A}_{\mathcal{I}_p\Gamma_p} \tag{4.6}$$

is a *local Schur complement* associated with the subdomain Ω_p. This local expression allows for computing local Schur complements independently from each other.

While the Schur complement system is significantly better conditioned than the original matrix $\mathcal{A}$ [115], it is important to consider further preconditioning when employing a Krylov method as discussed in Section 1.2.3.3 of Part I. It is well-known, for example, that $\kappa(A) = \mathcal{O}(h^{-2})$ when $\mathcal{A}$ corresponds to a standard discretization (e.g. piecewise linear finite elements) of the Laplace operator on a mesh with spacing h between the grid points. Using two non-overlapping subdomains effectively reduces the condition number of the Schur complement matrix to $\kappa(S) = \mathcal{O}(h^{-1})$. While improved, preconditioning can significantly lower this condition number further. In the literature, multiple variants for computing a preconditioner for the Schur complement of such hybrid solvers have been proposed. For example PDSLin [108], ShyLU [131] and Hips [71] first perform an exact [3] factorization of the interior of each domain concurrently. PDSLin and ShyLU then compute the preconditioner with a two-fold approach. First, an approximation $\widetilde{\mathcal{S}}$ of the (global) Schur complement $\mathcal{S}$. Second, this approximate Schur complement $\widetilde{\mathcal{S}}$ is factorized to form the preconditioner for the Schur Complement system, which does not need to be formed explicitly. While PDSLin has multiple options for discarding values lower than some user-defined threshold at different steps of the computation of $\widetilde{\mathcal{S}}$, ShyLU also implements a structure-based approach for discarding values named *probing* that was first proposed to approximate interfaces in DDM [36] inspired from coloring techniques to approximate Hessian in optimization [45]. Instead of following such a two-fold approach, Hips [71] forms the preconditioner by computing a global Incomplete LU (ILU) factorization based on the multi-level scheme formulation from [86]. Finally, in this study, MaPHyS [76] computes an additive Schwarz preconditioner for the Schur complement as further described in Section 4.2.2.

[3] There are also options for computing Incomplete LU factorizations of the interiors but the related descriptions are out the scope of this work.

4.2.2 Additive Schwarz preconditioning of the Schur Complement

For illustration, we consider the preconditioner originally proposed in [35] which aims at being highly parallel. The most straightforward method for building a preconditioner from the information provided by the local Schur complements would consist of performing their respective inversion. Such a preconditioner would write $M_{NN} = \sum_{p=1}^{N} \mathcal{R}_{\Gamma_p}^T \mathcal{S}_p^{-1} \mathcal{R}_{\Gamma_p}$ and corresponds to a Neumann-Neumann [24, 53] preconditioner applied to the Schur complement. However, even in the SPD case, the local Schur complement can be singular and such a preconditioner cannot be formed. Therefore, we consider the *local assembled Schur complement* $\bar{\mathcal{S}}_p = \mathcal{R}_{\Gamma_p} S \mathcal{R}_{\Gamma_p}^T$, which corresponds to the restriction of the global Schur complement to the interface Γ_p and which cannot be singular in the SPD case (as $\mathcal{S}$ is SPD as well [35]). This preconditioner reads:

$$M_{AS} = \sum_{p=1}^{N} \mathcal{R}_{\Gamma_p}^T \bar{\mathcal{S}}_p^{-1} \mathcal{R}_{\Gamma_p}. \tag{4.7}$$

This local assembled Schur complement can be built from the local Schur complements $\mathcal{S}_p$ by assembling their diagonal blocks. If we consider a planar graph partitioned into horizontal strips (1D decomposition), the resulting Schur complement matrix has a block tridiagonal structure as depicted in (4.8)

$$\mathcal{S} = \begin{pmatrix} \ddots & & & & \\ & S_{i-1,i-1} & S_{i-1,i} & & \\ & S_{i,i-1} & S_{i,i} & S_{i,i+1} & \\ & & S_{i+1,i} & S_{i+1,i+1} & \\ & & & & \ddots \end{pmatrix}. \tag{4.8}$$

For that particular structure of $\mathcal{S}$, the submatrices in boxes correspond to the $\bar{\mathcal{S}}_p$. Such diagonal blocks, which overlap, are similar to the classical block overlap of the Schwarz method when written in matrix form for a 1D decomposition. Similar ideas have been developed in a pure algebraic context in earlier papers [32, 130] for the solution of general sparse linear systems. Because of this link, the preconditioner defined by (4.7) is referred to as algebraic additive Schwarz preconditioner for the Schur complement. This is the preconditioner we deal with in the rest of this Chapter.

4.2.3 Parallel implementation

Given a linear system $\mathcal{A}x = b$ in a parallel distributed environment, MaPHyS proceeds as follows. It relies on graph partitioning tools such Scotch [44] or METIS [97] to partition the related adjacency matrix, which leads to subgraphs. These subgraphs correspond to subdomains while shared edges correspond to interface unknowns as early depicted in Figure 4.1a. Each subgraph interior is mapped to only one process whereas each local interface is replicated on each process connected to it.

With this data distribution, each process p concurrently eliminates the internal unknowns using a direct method. The factorizations of the local interiors are performed by each process

independently from each other and require no communication. The global linear system to solve in parallel is thus reduced to linear system associated with the interface, which is solved with an iterative method. For example to solve the linear system involving the matrix `Haltere` depicted in Table 4.1 with 128 processes, we use the graph partitioner METIS [97] to compute 128 local matrices defined by Equation (4.4). As depicted in Table 4.2, the mean size of the subsystems of equations mapped to each process is 10700, which is reduced to 1178 at the interface.

Local matrix	n	max	11075
		avg	10700
		min	10541
	nnz	max	84141
		avg	81850
		min	79429

Local Schur	n	max	1612
		avg	1178
		min	1028
	nnz	max	2598544
		avg	1387684
		min	1056784

Table 4.2 – Statistics on local matrices and associated Schur complements. The matrix `Haltere` is partitioned into 128 subdomains thanks to `METIS`.

For the computation of the Schur complement, each process computes $\mathcal{S}_p$ defined in Equation (4.9) in parallel using PastiX [65, 99] (or Mumps [9]) which is a direct sparse solver:

$$\mathcal{S}_p = \mathcal{A}_{\Gamma_p\Gamma_p} - \mathcal{A}_{\Gamma_p\mathcal{I}_p}\mathcal{A}^{-1}_{\mathcal{I}_p\mathcal{I}_p}\mathcal{A}_{\mathcal{I}_p\Gamma_p}. \tag{4.9}$$

Once the local Schur complements have been computed, each process communicates with its neighbors (in the graph) to assemble its local Schur complement $\bar{\mathcal{S}}_p$ and perform its factorization using the Intel Mkl library. This step only requires a few point-to-point communications. Finally, the last step is the iterative solution of the interface problem (4.2). For that purpose, we use Krylov method subroutines developed in [69].

4.3 Resilient sparse hybrid linear solver

As discussed in Section 2.2.4, the location of the preconditioner has an impact on the residual norm that is available for free during the iteration. For example GMRES as well as right preconditioned GMRES have the property of minimizing the residual norm at each iteration whereas this property is not guaranteed for left preconditioned GMRES.

We recall that the first step of the IR strategies is the computation of meaningful values for the lost entries using those of the current iterate available on the surviving processes. If right preconditioned GMRES is used, it will compute $x^{(k)}$ as follows $u^{(k)} = V_k y_k$, and $x^{(k)} = \mathcal{M}u^{(k)}$. When a single fault occurs, all the entries of $u^{(k)}$ can be computed except on the failed process. If some entries of $u^{(k)}$ are missing, even surviving processes cannot compute the entries of $x^{(k)}$ they are mapped on, except for a block diagonal preconditioner. To overcome this issue, it is important to notice that $x^{(k)}$ reads $x^{(k)} = \mathcal{M}V_k y_k$. If we

Algorithm 13 FGMRES, given a matrix $\mathcal{A}$, a preconditioner $\mathcal{M}$, a right hand side b, and an initial guess $x^{(0)}$

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

consider FGMRES presented in Algorithm 13, the entries of $\mathcal{M}V_k$ are explicitly stored (line 6), what is the matrix $Z_k = \mathcal{M}V_k$. This allows each surviving process q to compute the entries $x_{I_q}^{(k)} = Z_k(I_q, :)y_k$ (Algorithm 13, line 17). In the rest of this chapter, we use FGMRES at the iterative solve.

For the sake of exposition, we consider the 1D domain decomposition depicted in Figure 4.2 to describe how data is allocated over processes. Without loss of generality, we will also use this example for illustrating all the recovery mechanisms throughout this section. In this example, the domain is decomposed in four subdomains Ω_1, Ω_2, Ω_3 and Ω_4 with the associated interface $\Gamma = \Gamma_A \sqcup \Gamma_B \sqcup \Gamma_C$. Interface Γ_A is shared by subdomains Ω_1 and Ω_2, Γ_B by Ω_2 and Ω_3, Γ_C by Ω_3 and Ω_4.

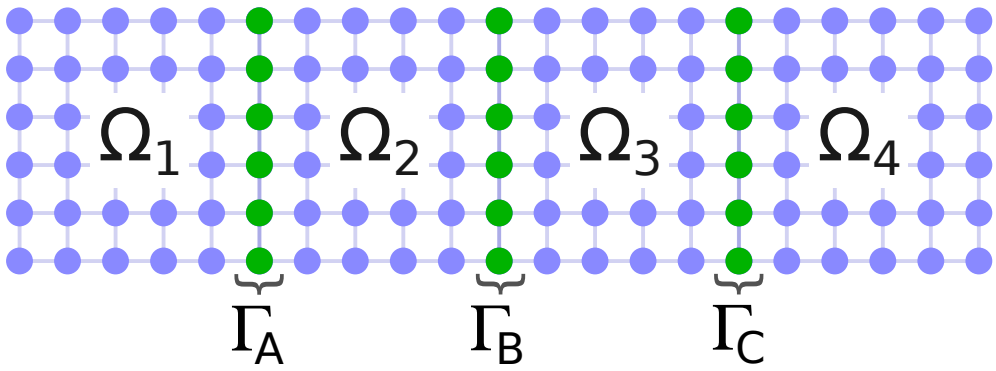


Figure 4.2 – 1D domain decomposition. The originally rectangular domain is partitioned into four subdomains with three interfaces.

With such a decomposition, the linear system associated with the Schur complement is described by Equation (4.10), where x_{Γ_A}, x_{Γ_B} and x_{Γ_C} are the unknowns associated with the interfaces Γ_A, Γ_B and Γ_C, respectively.

$$\begin{pmatrix} \mathcal{S}_{A,A} & \mathcal{S}_{A,B} & \\ \mathcal{S}_{B,A} & \mathcal{S}_{B,B} & \mathcal{S}_{B,C} \\ & \mathcal{S}_{C,B} & \mathcal{S}_{C,C} \end{pmatrix} \begin{pmatrix} x_{\Gamma_A} \\ x_{\Gamma_B} \\ x_{\Gamma_C} \end{pmatrix} = \begin{pmatrix} f_A \\ f_B \\ f_C \end{pmatrix}. \tag{4.10}$$

Following a classical parallel implementation of finite element substructuring approaches each submatrix described by Equation (4.4) associated with a given subdomain is allocated to a process. A direct consequence is that each process can compute its local Schur complement and the unknowns associated with a given interface are naturally replicated on the processes sharing this interface. This is the choice made in MaPHyS and with respect to this choice, processes p_1, p_2, p_3, p_4, are mapped on Γ_A, $\Gamma_A \sqcup \Gamma_B$, $\Gamma_B \sqcup \Gamma_C$ and Γ_C, respectively. Consequently, x_{Γ_A} is replicated on processes p_1 and p_2, x_{Γ_B} is replicated on processes p_2 and p_3, and so on. During the parallel solution of the Schur complement system, the Krylov solver computes redundantly and consistently dynamic data associated with these replicated unknowns. In the present case, dynamic data are the basis V_k, the search space Z_k and the Hessenberg matrix H_k. According to Assumption 1 (see p. 37), the basis V_k and Z_k are distributed, whereas the Hessenberg matrix is replicated on each process. For our 1D decomposition example, the block-row $V_{\Gamma_A,:}$ is replicated on processes p_1 and p_2, the block-row $V_{\Gamma_B,:}$ is replicated on processes p_2 and p_3, and the block-row $V_{\Gamma_C,:}$ is replicated on processes p_3 and p_4. The matrix Z_k is distributed in the same way.

4.3.1 Single fault cases

In this section, we explain the strategy to survive single process faults in the iterative solve step of MaPHyS (step (4a)). One advantage of having redundant local interfaces is that dynamic data on each process is also computed on neighbor processes. So, when a single process fails, we retrieve all its dynamic data from its neighbors. Once all dynamic data are recovered, FGMRES iterations can continue with exactly the same data as before the fault. Consequently, the numerical behavior and the solution from the faulty execution is the same compared to the corresponding fault free execution. Indeed, the unique penalty is the communication time to reconstitute lost data.

Let us come back to the 1D decomposition from Figure 4.2 to illustrate how to retrieve lost data. We present two examples. First we illustrate how to retrieve data when a process with only one neighbor fails. Second we illustrate the case of a process with two neighbors. For the first case, we assume that process p_1 fails. The Hessenberg matrix is retrieved from any process because it is fully replicated. The block-rows $V_{\Gamma_A,:}$ and $Z_{\Gamma_A,:}$ are retrieved from process p_2. For the second case, we assume that a fault occurs on process p_2. The Hessenberg matrix is retrieved from any surviving process. The block-rows $V_{\Gamma_A,:}$ and $Z_{\Gamma_A,:}$ are retrieved from process p_1 while the block-rows $V_{\Gamma_B,:}$ and $Z_{\Gamma_B,:}$ are retrieved from process p_3. Once all lost data are retrieved, FGMRES iterations continue in the same state as before the fault, exhibiting the same numerical behavior as in the non faulty case.

4.3.2 Interpolation-restart strategy for the neighbor processes fault cases

When a fault occurs on neighboring processes, some data remain lost despite data redundancy. We describe how IR strategies from Section 2.2 of Chapter 2 can be modified to take advantage of the features of MaPHyS's preconditioner in order to efficiently survive neighbor processes fault.

To illustrate a fault on neighbor processes, we reconsider the example of a 1D decomposition illustrated in Figure 4.2 and we assume that processes p_2 and p_3 have both failed. This example, there is no way to retrieve the dynamic data associated with the interface Γ_B shared by the failed processes; an interpolation procedure must be implemented to generate meaningful entries of the iterate on Γ_B.

A first possibility is to use the LI strategy; processes p_2 and/or p_3 solve the local linear system

$$\mathcal{S}_{B,B} x_{\Gamma_B} = f_B - \mathcal{S}_{B,A} x^{(k)}_{\Gamma_A} - \mathcal{S}_{B,C} x^{(k)}_{\Gamma_C}$$

to interpolate $x^{(k)}_{\Gamma_B}$. This direct application of the LI strategy requires the factorization of $\mathcal{S}_{B,B}$. Similarly, a direct application of the LSI strategy requires a QR factorization. However it is possible to design an IR strategy that exploits the features of the MaPHyS preconditioner and consequently avoids the additional factorization. As discussed in Section 4.2.3, the factorization of the local assembled Schur complement is the main building block of the preconditioner. In our 1D example, the local preconditioner of p_2 is $\bar{\mathcal{S}}^{-1}_{p_2}$ with

$$\bar{\mathcal{S}}_{p_2} = \begin{pmatrix} \mathcal{S}_{A,A} & \mathcal{S}_{A,B} \\ \mathcal{S}_{B,A} & \mathcal{S}_{B,B} \end{pmatrix}$$

and the local preconditioner of p_3 is $\bar{\mathcal{S}}^{-1}_{p_3}$ with

$$\bar{\mathcal{S}}_{p_3} = \begin{pmatrix} \mathcal{S}_{B,B} & \mathcal{S}_{B,C} \\ \mathcal{S}_{C,B} & \mathcal{S}_{C,C} \end{pmatrix}$$

These factorizations of local assembled Schur complements are computed before the iterative solve step and are considered as static in our model. Consequently they are available after a fault. Based on these matrix factorizations we design an interpolation variant referred to as LI^{AS}. With the LI^{AS} approach, process p_2 solves the local linear system

$$\begin{pmatrix} \mathcal{S}_{A,A} & \mathcal{S}_{A,B} \\ \mathcal{S}_{B,A} & \mathcal{S}_{B,B} \end{pmatrix} \begin{pmatrix} x_{\Gamma_A} \\ x_{\Gamma_B} \end{pmatrix} = \begin{pmatrix} f_A \\ f_B - \mathcal{S}_{B,C} x^{(k)}_{\Gamma_C} \end{pmatrix} \tag{4.11}$$

while p_3 solves

$$\begin{pmatrix} \mathcal{S}_{B,B} & \mathcal{S}_{B,C} \\ \mathcal{S}_{C,B} & \mathcal{S}_{C,C} \end{pmatrix} \begin{pmatrix} x_{\Gamma_B} \\ x_{\Gamma_C} \end{pmatrix} = \begin{pmatrix} f_B - \mathcal{S}_{B,A} x^{(k)}_{\Gamma_A} \\ f_C \end{pmatrix}. \tag{4.12}$$

Consequently, a different values of x_{Γ_B} are available on p_2 and p_3 and new entries of x_{Γ_A} and x_{Γ_C} are computed. The entries of x_{Γ_A} and x_{Γ_C} computed are not used since they are

available on p_1 and p_4, respectively. Finally we make the value of x_{Γ_B} consistent on p_2 and p_3 by simply averaging these values,

$$x_{\Gamma_B}^{(LI^{AS})} = \frac{1}{2}(x_{\Gamma_B 2}^{(LI^{AS})} + x_{\Gamma_B 3}^{(LI^{AS})}).$$

Finally the vector

$$\begin{pmatrix} x_{\Gamma_A}^{(k)} \\ x_{\Gamma_B}^{(LI^{AS})} \\ x_{\Gamma_C}^{(k)} \end{pmatrix}$$

is used as an initial guess to restart FGMRES.

The presented LI^{AS} strategy naturally extends to general decompositions based on the same idea and can be summarized as follows into four main steps:

1. *Computation of non faulty entry*: All living processes compute the entries of the current iterate that they are mapped on.

2. *Computation of right-hand side contribution*: The neighbors of the failed processes compute the contributions required to update the right-hand sides of the interpolation linear systems. The computation of the right-hand sides associated with the linear interpolation may require significant communication time depending on the number of neighbors of the failed processes. Indeed, to update the right-hand side, a failed process needs contributions from all its neighbors. On the other hand, neighbors participate to the interpolation by computing locally matrix vector multiplications required for right-hand side update.

3. *Communication*: Each failed process retrieves lost entries of the current iterate except the entries definitively lost, which are share by failed processes. At the same time, failed processes receive the contributions from neighbors to update the right-hand side associated with the local interpolation.

4. *Interpolation*: Each failed process solves the interpolation linear system, and failed processes communicate to maintain the same value of the interpolated entries.

At the end of the these four steps, consistent state and FGMRES can be restarted with the interpolated iterate as a new initial guess. In contrast to the single process fault case, the numerical behavior is not the same as the non faulty case anymore.

4.4 Experimental results

In this section, we present experimental results for the resilient sparse hybrid linear solver described above. Instead of actually crashing a process, we simulate its crash by removing its dynamic data. Indeed, the goal of this thesis is not to assess systems mechanisms for

supporting resilience. We consider this a complementary problem. Therefore the cost for resetting the system in a coherent state (such as creating a new process and adapting communicators) and retrieving static data are not taken into account. In the single process fault case, we assess only the communication time required to retrieve lost dynamic data. In the neighbor processes fault case, we present the numerical behavior of LI^{AS} as well as a performance analysis. The experiments are performed two platforms. On one hand the `Plafrim`[4] platform where each node of has two Quad-core Nehalem Intel Xeon X5550. On the other hand, the `Hopper` platform[5]. Each node on `Hopper` has two twelve-core AMD 'MagnyCours' 2.1-GHz processors. MaPHyS as well as the proposed resilient extension have been written in Fortran 90 and support two levels of parallelism (MPI + Thread). As discussed in Section 4.2.3, MaPHyS is modular and relies on state-of-the-art packages for performing domain decomposition and direct factorization. For the experiments, we have used the METIS package, the PastiX package and the Intel Mkl libraries.

We present two categories of experiments. On the one hand, we present the numerical behavior of LI^{AS} as well as numerical details using the resilient version of MaPHyS with one level of parallelism (MPI only) on the `Plafrim` platform using 128 cores. On the other hand, the performance analysis was achieved with multithreaded version of the designed resilient MaPHyS on the `Hopper` platform up to 12,288 cores.

4.4.1 Results on the `Plafrim` platform

4.4.1.1 Single fault cases

In this section we present results for the single process fault case presented in Section 4.3.1. We recall that the numerical behavior is the same as the non faulty execution and the overhead is only due to communication. In this case, the overhead depends mainly on the volume of data received from surviving neighbors. To illustrate this behavior, we have considered two cases of fault injection. We inject a fault when the dimension of the search basis is very small, that is, the fault occurs a few iteration steps after a restart. In the second situation, we inject a fault a few iterations before the restart. In the last case, the dimension of the search space is close to the maximum affordable.

In the results depicted in Table 4.3 for matrix `Matrix211`, FGMRES restarts every other 300 iterations and converges within 499 iterations. We have injected a fault at iteration 305, which means that when the fault occurs, the basis V_k and Z_k are of dimension 5 and the Hessenberg matrix H_k is small. The failed process receives the Hessenberg matrix of size 6×5 from one of its neighbors and two block-rows (one from V_k and the other from Z_k) of five vectors from each of its neighbors. Without fault, FGMRES converges in 7.72 seconds while the faulty execution converges in 7.86 seconds, which leads to an overhead of 1.78% in term of extra computational cost.

If we consider the results of `Haltere` and `Tdr455k` displayed in Table 4.3, faults are injected at iteration 3 and 5, respectively, after the restart. The overheads are very low (0.06% and 0.16% for `Haltere` and `Tdr455k`, respectively).

[4]https://plafrim.bordeaux.inria.fr/

[5]https://www.nersc.gov/users/computational-systems/hopper/configuration/compute-nodes/

Matrix	Matrix211	Haltere	Tdr455k
Nb iterations	499	32	4941
Restart (m)	300	16	500
Faulty iteration	305	19	505
Time NF	7.72	0.77	292.50
Time faulty	7.86	0.77	292.90
Overhead	1.78%	0.06%	0.16%

Table 4.3 – Overhead to retrieve lost data exploiting data redundancy with 128 cores. For each matrix, the fault is injected a few iterations after a restart.

To assess how the overhead is impacted when the dimension of the search space Z_k increases, we keep the same configuration but faults are injected a few iteration before a restart. The results are depicted in Table 4.4. The faulty iteration of matrix `Matrix211` is set to 470 while the restart is every other 300 iterations. In this configuration, the fault occurs when matrices V_k, Z_k and H_k have 170 columns, as a consequence, the overhead significantly increases from 1.78% to 8.37%. A slight increase of the overhead is observed for matrices `Haltere` and `Tdr455k` also.

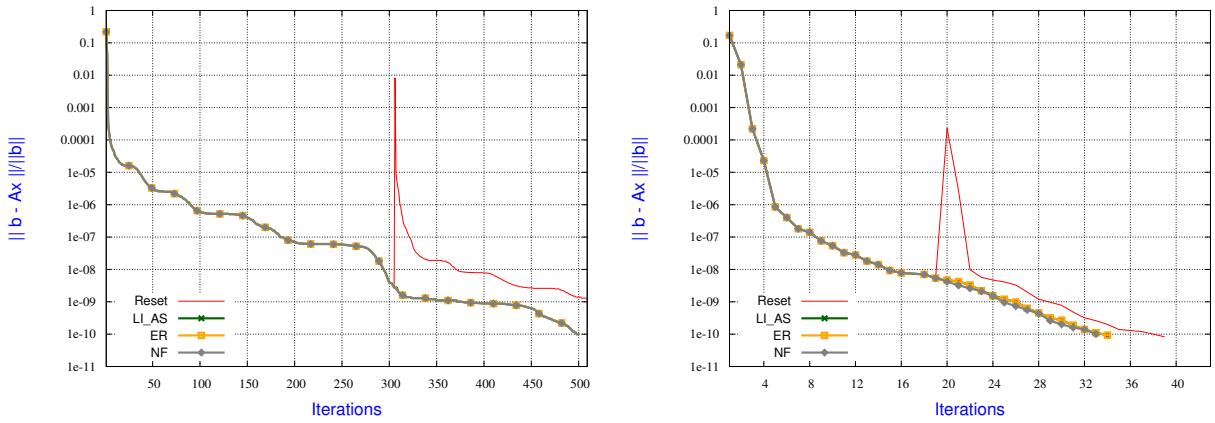
Matrix	Matrix211	Haltere	Tdr455k
Nb iterations	499	32	4941
Restart parameter	300	16	500
Faulty iteration	470	27	4940
Time NF	7.72	0.77	292.50
Time faulty	8.37	0.77	293.20
Overhead	8.37%	0.54%	0.24%

Table 4.4 – Overhead to retrieve lost data exploiting data redundancy with 128 cores. For each matrix, the fault is injected a few iterations before a restart.

It is important to notice that `Tdr455k` has recovered block-rows of 440 vectors, however its overhead remains very low. Indeed, `Tdr455k` required many iterations to converge (4941 iterations in our example), so the time spent in the recovery step is not significant compared to the overall time required to converge.

4.4.1.2 Neighbor processes fault cases

In this section, we present results for the LI^{AS} strategy designed to handle neighbor processes fault (see Section 4.3.2). For the sake of exposure, we keep the same configuration as in the single process fault but, at the faulty iterations, we inject a fault in two neighbor processes. In this section, we assess the numerical robustness of LI^{AS}. For the sake of comparison, we also display result for NF, ER and Reset. We recall that NF refers to the non faulty execution. ER refers to Enforced Restart, more precisely with the ER strategy, no dynamic data is lost, we only use the current iterate to restart FGMRES. The Reset strategy consists in restarting with the current iterate in which corrupted entries are set to zero.



(a) Matrix: `Matrix211`, Restart: 300, Faulty iteration: 305.

(b) Matrix: `Haltere`, restart m=16, faulty iteration: 19.

Figure 4.3 – Numerical robustness of Interpolation-restart strategy designed for neighbor processes fault during the iterative solve. For both matrices, the original system is partitioned into 128 subdomains using FGMRES as iterative solver. For both matrices, LI^{AS}, ER and NF almost coincide.

In Figure 4.3, we present the robustness of LI^{AS}. The first observation is that when a fault occurs, the Reset strategy exhibits a large peak in the residual norm, while LI^{AS} remains as robust as ER and NF. These results highlight the robustness of LI^{AS}. The overlap of LI^{AS}, ER and NF illustrates the numerical quality of the interpolated entries. For both `Haltere` and `Matrix211`, the fault occurs when the size of the search space was small. Consequently, the effect of the restart was low (which is illustrated by the overlap of ER and NF). These results show the robustness of LI^{AS} for MaPHyS.

In this section we analyze the overhead induced by the LI^{AS} strategy. For that, we measure the time T_r for performing the whole recovery consisting of the four steps described in Section 4.3.2 (see p. 103). In the cases of ER and Reset, the recovery is reduced to the communication step (step 3, see again p. 103). The time for performing the recovery is thus reduced to this single communication step for ER and Reset. Besides the time for performing the recovery, there is a potential numerical penalty due to the quality of the interpolated data (in the LI$^{(AS)}$ case) and of the necessity of restarting (in ER, LI$^{(AS)}$, and Reset cases). For each execution (NF, ER, LI$^{(AS)}$ and Reset), we denote T_c the computation

time required for FGMRES convergence, and $Overhead_{total}$ the total overhead. The total overhead includes the time spent in the recovery step and the time induced by a numerical penalty. We also display separately the overhead related to the recovery, which is denoted $Overhead_r$.

In Table 4.5, we depict the overheads associated with the result presented in Figure 4.3a. In term of number of iterations, ER and LI^{AS} do not exhibit a convergence delay while Reset converges after 95 additional iterations. However in term of computational time, there is an overhead rounded to 0.96%, while these additional iterations required by Reset increase its overhead to 21.79%. The first remark in this experiment is that LI^{AS} is as good as ER and both induce a similar overhead.

	NF	ER	LI^{AS}	Reset
Nb Iterations	499	499	499	594
T_c (second)	7.72	7.80	7.80	9.41
T_r (second)	-	8.27E-04	8.19E-03	8.34E-04
$Overhead_{total}$	-	0.96%	0.96%	21.79%
$Overhead_r$	-	0.01%	0.1%	0.01%

Table 4.5 – `Matrix211` on 128 cores (restart m=300). Neighbor processes fault at iteration 305.

Compared to Reset and ER, LI$^{(AS)}$ requires more time to regenerate an initial guess when a fault occurs. Indeed Reset and ER need only communication whereas LI$^{(AS)}$ has additional computation cost due to interpolation. However the overhead induced by LI^{AS} to regenerate the initial guess represents only 0.1%, what is consistent with our expectation.

	NF	ER	LI^{AS}	Reset
Nb Iterations	499	608	608	758
T_c (second)	7.72	9.96	10	12.80
T_r (second)	-	8.49E-04	8.11E-03	8.29E-04
$Overhead_{total}$	-	28.88%	29.40%	65.63%
$Overhead_r$	-	0.01%	0.1%	0.01%

Table 4.6 – `Matrix211` on 128 cores (restart m=300). Neighbor processes fault at iteration 470.

When the fault occurs while the Krylov basis contains many vectors, the overheads significantly increase as depicted in Table 4.6. Reset has required much more iterations to converge (758). Because of the large restart, LI^{AS} and ER are similarly impacted as the number of iterations increases from 499 to 608. This translates in high overheads 29.40% for LI^{AS} and 28.88% for ER. However the fact that the overheads of LI^{AS} and ER are of the same order magnitude indicates that these increases of the overhead are particularly due to the restart. This is also confirmed by T_r which has barely increased.

For problems where the convergence can be observed with a small restart, the trends are different. As depicted in Table 4.7, with `Haltere`, Reset converges in 37 iterations, LI^{AS} and ER in 33 iterations, while NF converges in 32 iterations. ER and LI^{AS} have only two iterations of delay, which indicates that the restart due to a fault has not significantly perturbed the convergence. On the other hand, the fact that LI^{AS} and ER finish after 34 iterations indicates that data interpolated by LI^{AS} has a good numerical quality. The time spent by LI^{AS} to regenerate lost data remains very low with a total overhead of 10.40% while ER has a total overhead of 7.41%. Reset is not penalized by restart but by the numerical quality of data regenerated, what leads to a total overhead of 21.70%.

	NF	ER	LI^{AS}	Reset
Nb Iterations	32	33	33	37
T_c	0.77	0.83	0.85	0.94
T_r	-	1.37E-04	0.01	1.45E-04
$Overhead_{total}$	-	7.41%	10.40%	21.70%
$Overhead_r$	-	0.02%	1.71%	0.02%

Table 4.7 – `Haltere` on 128 cores (restart m=16). Neighbor processes fault at iteration 19.

	NF	ER	LI^{AS}	Reset
Nb Iterations	32	34	34	39
T_c	0.77	0.86	0.87	0.96
T_r	-	1.39E-04	0.01	1.40E-04
$Overhead_{total}$	-	11.44%	12.18%	24.56%
$Overhead_r$	-	0.02%	1.73%	0.02%

Table 4.8 – `Haltere` on 128 cores (restart m=16). Neighbor processes fault at iteration 27.

4.4.2 Performance analysis on the `Hopper` platform

Here, we assess the overhead induces by our strategies on a large-scale platform. Once the problem is partitioned, each subdomain is mapped to one process. We use three threads per process and eight processes per node, which leads to a total of 24 threads per node in order to exploit the 24 cores on each node. For each matrix and a given number of processes, we performed many experiments by varying the iteration when the fault is injected with only one fault by experiment, and we report the average overhead. It is important to note that here we do not detail the average, but we report on the whole average.

4.4.2.1 Single fault cases

In this section, we present results for single process fault cases. We recall that in this case, the numerical behavior is the same as the non faulty execution and the overhead

is only due to communication. To solve the linear systems associated with `Matrix211`, we vary the number of cores from 384 to 3,072 (Table 4.9). Regardless of the number of cores, the overhead induced by the fault recovery strategy remains low. One can also observe the decrease of the overhead when the number of cores varies between 348 and 1,536. Indeed, when the number of cores increases, the volume of data associated with each process decreases. This leads to the decrease of the volume of data loss when a fault occurs. However with 3,072 cores, the overhead increases. This is due to the limitation of the gain associated with the increase of the number of processes. On the other hand, according to the size of the matrix, beyond a given number of processes, the fault recovery involves many processes. This may be penalizing because of MPI communication synchronization.

Nb of cores	384	768	1,536	3,072
Overhead	2.10%	1.18%	0.05%	0.38%

Table 4.9 – Variation of the overhead in the case of a single process fault while increasing the number of cores using `Matrix211`.

If we consider the result of `Nachos4M` presented in Table 4.10, one can observe that even with 12,288 cores, the overhead keeps decreasing because `Nachos4M` has a larger size. Furthermore, since the size of `Nachos4M` allows us to exploit larger numbers of processes, the induced overheads are very low, which demonstrate the potential of such strategies for large-scale problems.

Nb of cores	1,536	3,072	6,144	12,288
Overhead	0.84%	0.82%	0.76%	0.02%

Table 4.10 – Variation of the overhead in the case of a single process fault while increasing the number of cores using `Nachos4M`.

4.4.2.2 Neighbor processes fault cases

In this section, we present results for the LI^{AS} strategy designed to handle neighbor processes faults. We recall that LI^{AS} exploits data redundancy to retrieve available entries from surviving neighbors, before interpolating missing entries taking advantage of the additive Schwarz preconditioner. The overhead of the LI^{AS} strategy includes the communication time to retrieve available entries from surviving neighbors, the computational time to interpolate missing entries and the overhead induced by a possible numerical penalty. The numerical penalty may be induced by the quality of interpolated entries and the necessity to restart after a neighbor processes fault. The numerical penalty often leads to additional iterations, which may increase the computational time. The results for `Matrix211` is reported in Table 4.11. With 384 cores, we have an overhead of 3.65%, but with the increase of the number of cores, the overhead decreases significantly down to 0.12%.

Nb of cores	384	768	1,536	3,072
Overhead	3.65%	1.31%	0.12%	0.45%

Table 4.11 – Variation of the overhead in the case of neighbor processes fault while increasing the number of cores using `Matrix211`.

Even in the case of neighbor processes fault, the overhead associated with `Nachos4M` remains very low, from 1.70% down to 0.06%. This demonstrates again the attractive potential of our strategies for large-scale problems.

Nb of cores	1,536	3,072	6,144	12,288
Overhead	1.70%	1.26%	0.67%	0.06%

Table 4.12 – Variation of the overhead in the case of neighbor processes fault while increasing the number of cores using `Nachos4M`.

4.5 Concluding remarks

The main objective of this part was to design a numerically resilient solution for large sparse linear systems on large massively parallel platforms. For that purpose, we have considered the fully-featured sparse hybrid solver MaPHyS. We have exploited the solver properties to design two different resilient solutions for the iterative solve step: one to recover from a single fault and another one to survive a fault on neighbor processes.

In the case of a single fault, we have exploited data redundancy to retrieve all dynamic data from neighbors. Once all dynamic data are recovered, the iterations continue with exactly the same data as before the fault. This solution is simple and requires only communications to reconstitute lost data. This solution has no numerical penalty so it exhibits the same convergence behavior as a faultfree execution. In the case of a faults on neighbor processes, we have designed the $\text{LI}^{(AS)}$ variant which takes advantage of the features of MaPHyS's preconditioner so that it does not require any additional factorization. We have analyzed the numerical behavior of $\text{LI}^{(AS)}$ and evaluated its overhead to regenerate an initial guess to restart the iterative solver. The results show that $\text{LI}^{(AS)}$ is very robust and it computes an initial guess with a residual norm of the same order of magnitude as the one computed before a fault. All the experiments show that the computation time required to regenerate an initial guess after a fault is reasonable because it has a very low overhead. However $\text{LI}^{(AS)}$ also suffers from the numerical penalty due to a restart, as commonly observed for IR strategies. Our studies show that the numerical penalty varies according to the size of the Krylov basis when a fault occurs. The larger the size of the Krylov basis, the higher the numerical penalty. Consequently, problems that can converge with a small restart parameter are less penalized.

The numerical approach used in this chapter to an algebraic domain decomposition technique does apply to substructuring classical non-overlapping domain decomposition approach where the redundancy is naturally implemented in a finite element framework. Furthermore, the IR strategy can also be extended and applied to many classical domain decomposition methods for PDE solution. As an example, we can report on an ongoing work in collaboration with colleagues from another Inria project, namely Nachos. One objective of that collaboration was to exploit IR strategies to design a resilient numerical solution for the time-harmonic Maxwell equations discretized by discontinuous Galerkin methods on unstructured meshes [22, 59].

Chapter 5

Conclusion and perspectives

The main objective of this thesis was to explore numerical schemes for designing resilient strategies that enable parallel numerical linear algebra solvers to survive faults. In the context of linear algebra solvers, we investigated numerical approaches to regenerate meaningful dynamic data before restarting the solution schemes. We have presented new numerical resilience algorithms called IR strategies for the solution of linear systems of equations in Chapter 2. In particular, we have considered two rational policies that preserve key numerical properties of well-known linear solvers, namely the monotony decrease of the A-norm of the error of the conjugate gradient or the residual norm decrease of GMRES. We have assessed the impact of the interpolation-restart techniques, the fault rate and the amount of lost data on the robustness of the resulting linear solvers. In Chapter 3, we have tuned IR strategies to design numerical resilient techniques for iterative eigensolvers such as Arnoldi, Implicitly restarted Arnoldi or subspace iteration algorithm and the Jacobi-Davidson solver. The numerical features of this latter eigensolver offer the flexibility to design fairly efficient and robust resilient schemes for this widely used eigensolver.

Once the resilience potential of the IR strategies have been evaluated in stressful conditions simulated by high fault rates and large volume of lost data, we have focused on their extension to existing parallel linear algebra solvers. For that purpose, we have considered a fully-featured sparse hybrid (direct/iterative) solver to make its iterative solve step resilient. In the case of a single fault, we have exploited data redundancy to retrieve all lost dynamic data without any IR strategy. In the neighbor processes fault case, we have designed an IR variant that takes advantage of the features of MaPHyS's preconditioner. We have studied this IR variant ($\text{LI}^{(AS)}$) and evaluated the overhead required to regenerate a meaningful initial guess to restart the iterative solver. The results show that $\text{LI}^{(AS)}$ is very robust; that is, it computes an initial guess with a residual norm of the same order of magnitude as one computed before a fault. All the experiments show that the computation time required to regenerate an initial guess after a fault is reasonable because it has a very low overhead.

In this thesis, we have studied numerical resilient schemes in the framework of node crashes in a parallel distributed memory environment. However these numerical approaches can be extended to data corruption at lower granularity than node crashes. The interpo-

lation methods can also be tuned for well-known solvers in order to attempt to regenerate more information, in particular on the global research space to alleviate the penalty induced by the restart after each fault. For example, in [114], our IR strategies have been adapted to survive to data corruption at memory page granularity to design a resilient CG. In such a resilient CG, all corrupted memory pages of dynamic data are regenerated and there is no need to restart. This shows that IR strategies are very flexible and can also be adapted to uncorrected bit-flips or more generally to memory corruption when the location of the corrupted memory is available. Furthermore, at a low granularity, lost entries can be regenerated with an extremely low cost.

We have used a sparse direct method to perform all interpolations. However a direct method may be expensive when the amount of lost data is large. Alternatively, an iterative scheme might be considered with a stopping criterion related to the accuracy level of the iterate when the fault occurs. The $\text{LI}^{(AS)}$ variant designed for MaPHyS takes advantage of the additive Schwarz preconditioner so a direct solution is the best choice regardless of the amount of lost data. However MaPHyS gives an alternative option to the end-user consisting in using a sparse preconditioner. This option has not been investigated in this thesis. The sparse preconditioning technique consists in assembling the local Schur complement, sparsifying them concurrently before factorizing them using a sparse direct solver. In that case, the factorization of the local Schur complement would not be available at preconditioning step anymore. Consequently $\text{LI}^{(AS)}$ would require an additional and costly factorization. In that case, relying on an iterative scheme might limit the potential overhead for recovery.

Finally, our numerical resilient strategies can be efficiently combined with existing fault tolerance mechanisms such as checkpoint/restart techniques to design low overhead resilient tools for extreme scale calculations.

Bibliography

[1] Fault injection capabilities infrastructure. http://lxr.linux.no/linux/Documentation/fault-injection/.

[2] Linux Programmer's Manual: MMAP. http://man7.org/linux/man-pages/man2/mmap.2.html.

[3] Linux programmer's manual: MPROTECT. unixhelp.ed.ac.uk/CGI/man-cgi?mprotect.

[4] E. Agullo, L. Giraud, A. Guermouche, and J. Roman. Parallel hierarchical hybrid linear solvers for emerging computing platforms. *Compte Rendu de l'Académie des Sciences - Mécanique*, 339(2-3):96–105, 2011.

[5] Emmanuel Agullo, Luc Giraud, Abdou Guermouche, Jean Roman, and Mawussi Zounon. Towards resilient parallel linear Krylov solvers: recover-restart strategies. Research Report RR-8324, INRIA, July 2013.

[6] Lorenzo Alvisi and Keith Marzullo. Message logging: Pessimistic, optimistic, causal, and optimal. *IEEE Trans. Softw. Eng.*, 24(2):149–159, February 1998. ISSN 0098-5589. doi:10.1109/32.666828.

[7] S. Amari. Bounds on MTBF of systems subjected to periodic maintenance. *IEEE Transactions Reliability*, 55:469–474, September 2006. ISSN 0018-9529.

[8] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[9] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

[10] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell. Fault injection for dependability validation: a methodology and some applications. *Software Engineering, IEEE Transactions on*, 16(2):166–182, Feb 1990. ISSN 0098-5589. doi:10.1109/32.44380.

[11] W. E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Q. Appl. Math*, 9(17):17–29, 1951.

[12] Todd M. Austin. DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design. In *Proceedings of the 32Nd Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 32, pages 196–207. IEEE Computer Society, Washington, DC, USA, 1999. ISBN 0-7695-0437-X.

[13] J.H. Barton, E.W. Czeck, Z.Z. Segall, and D.P. Siewiorek. Fault injection experiments using FIAT. *Computers, IEEE Transactions on*, 39(4):575–582, Apr 1990. ISSN 0018-9340. doi:10.1109/12.54853.

[14] Leonardo Bautista-Gomez, Seiji Tsuboi, Dimitri Komatitsch, Franck Cappello, Naoya Maruyama, and Satoshi Matsuoka. FTI: high performance fault tolerance interface for hybrid systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 32. ACM, 2011.

[15] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, Katherine Yelick, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Stephen Keckler, Dean Klein, Peter Kogge, R. Stanley Williams, and Katherine Yelick. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems Peter Kogge, Editor & Study Lead. 2008.

[16] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software*, 28:135–151, 2001.

[17] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca, and J.J. Dongarra. An evaluation of User-Level failure mitigation support in MPI. DOI 10.1007/s00607-013-0331-3:1–14, May 2013.

[18] Wesley Bland, George Bosilca, Aurelien Bouteiller, Thomas Herault, and Jack Dongarra. A proposal for User-Level Failure Mitigation in the MPI-3 Standard. Technical report, Tech. rep., Department of Electrical Engineering and Computer Science, University of Tennessee, 2012.

[19] W.G. Bliss, M.R. Lightner, and B. Friedlander. Numerical properties of algorithm-based fault-tolerance for high reliability array processors *. In *Signals, Systems and Computers, 1988. Twenty-Second Asilomar Conference on*, volume 2, pages 631–635. 1988. ISSN 1058-6393. doi:10.1109/ACSSC.1988.754623.

[20] Anita Borg, Jim Baumbach, and Sam Glazer. A message system supporting fault tolerance. *SIGOPS Oper. Syst. Rev.*, 17(5):90–99, October 1983. ISSN 0163-5980. doi:10.1145/773379.806617.

[21] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov. MPICH-V: Toward a scalable fault tolerant MPI for volatile nodes. In *Supercomputing,*

ACM/IEEE 2002 Conference, pages 29–29. Nov 2002. ISSN 1063-9535. doi: 10.1109/SC.2002.10048.

[22] M. El Bouajaji, V. Dolean, M. J. Gander, and S. Lanteri. Optimized Schwarz methods for the time-harmonic Maxwell equations with damping. *SIAM J. Scientific Computing*, 34(4):A2048–A2071, 2012.

[23] Marin Bougeret, Henri Casanova, Mikael Rabie, Yves Robert, and Frédéric Vivien. Checkpointing strategies for parallel jobs. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC'11, pages 33:1–33:11. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-0771-0. doi: 10.1145/2063384.2063428.

[24] J.-F. Bourgat, R. Glowinski, P. Le Tallec, and M. Vidrascu. Variational formulation and algorithm for trace operator in domain decomposition calculations. In Tony Chan, Roland Glowinski, Jacques Périaux, and Olof Widlund, editors, *Domain Decomposition Methods*, pages 3–16. SIAM, Philadelphia, PA, 1989.

[25] Aurélien Bouteiller, Franck Cappello, Thomas Herault, Géraud Krawezik, Pierre Lemarinier, and Frédéric Magniette. MPICH-V2: A fault tolerant MPI for volatile nodes based on pessimistic sender based message logging. In *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, SC '03, pages 25–. ACM, New York, NY, USA, 2003. ISBN 1-58113-695-1. doi:10.1145/1048935.1050176.

[26] Aurelien Bouteiller, Thomas Hérault, Géraud Krawezik, Pierre Lemarinier, and Franck Cappello. MPICH-V Project: A multiprotocol automatic fault-tolerant MPI. *IJHPCA*, pages 319–333, 2006.

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] X.-C. Cai and Y. Saad. Overlapping domain decomposition algorithms for general sparse matrices. *Numerical Linear Algebra with Applications*, 3:221–237, 1996.

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

[34] J. Carreira, D. Costa, and J.G. Silva. Fault-injection spot-checks computer system dependability. *IEEE Spectrum*, 36(8):50–55, 1999.

[35] L. M. Carvalho, L. Giraud, and G. Meurant. Local preconditioners for two-level non-overlapping domain decomposition methods. *Numerical Linear Algebra with Applications*, 8(4):207–227, 2001.

[36] Tony F. C. Chan and Tarek P. Mathew. The interface probing technique in domain decomposition. *SIAM J. Matrix Anal. Appl.*, 13(1):212–238, January 1992. ISSN 0895-4798. doi:10.1137/0613018.

[37] 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.

[38] Peter M Chen, Edward K Lee, Garth A Gibson, Randy H Katz, and David A Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys (CSUR)*, 26(2):145–185, 1994.

[39] Yanqing Chen, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam. Algorithm 887: CHOLMOD, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Softw.*, 35(3):22:1–22:14, October 2008. ISSN 0098-3500. doi:10.1145/1391989.1391995.

[40] Zizhong Chen. Online-ABFT: an online algorithm based fault tolerance scheme for soft error detection in iterative methods. In *ACM SIGPLAN Notices*, volume 48, pages 167–176. ACM, 2013.

[41] 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.

[42] Zizhong Chen, Graham E Fagg, Edgar Gabriel, Julien Langou, Thara Angskun, George Bosilca, and Jack Dongarra. Fault tolerant high performance computing by a coding approach. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 213–223. ACM, 2005.

[43] Elliott Ward Cheney. Introduction to approximation theory, 1966.

[44] C. Chevalier and F. Pellegrini. PT-SCOTCH: a tool for efficient parallel graph ordering. *Parallel Computing*, 34(6-8), 2008.

[45] T.F. Coleman and J.J. Moré. Estimation of sparse Hessian matrices and graph coloring problems. *Math. Programming*, 28:243–270, 1984.

[46] Camille Coti, Thomas Herault, Pierre Lemarinier, Laurence Pilard, Ala Rezmerita, Eric Rodriguez, and Franck Cappello. Blocking vs. non-blocking coordinated checkpointing for large-scale fault tolerant MPI. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06. ACM, New York, NY, USA, 2006. ISBN 0-7695-2700-0. doi:10.1145/1188455.1188587.

[47] Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[48] Om P Damani and Vijay K Garg. How to recover efficiently and asynchronously when optimism fails. In *Distributed Computing Systems, 1996., Proceedings of the 16th International Conference on*, pages 108–115. IEEE, 1996.

[49] E. R. Davidson. The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. *J. Comput. Phys.*, 17:87, 1975.

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

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

[52] Timothy A. Davis and Yifan Hu. The University of Florida sparse matrix collection. *j-TOMS*, 38(1):1:1–1:25, November 2011.

[53] Y.-H. De Roeck and P. Le Tallec. Analysis and test of a local domain decomposition preconditioner. In Roland Glowinski, Yuri Kuznetsov, Gérard Meurant, Jacques Périaux, and Olof Widlund, editors, *Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 112–128. SIAM, Philadelphia, PA, 1991.

[54] James Demmel, Jack Dongarra, Axel Ruhe, and Henk van der Vorst. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. ISBN 0-89871-471-0.

[55] Dhananjay M Dhamdhere. *Operating Systems: A Concept-based Approach, 2E*. Tata McGraw-Hill Education, 2006.

[56] Yuvraj Singh Dhillon, Abdulkadir Utku Diril, and Abhijit Chatterjee. Soft-error tolerance analysis and optimization of nanometer circuits. *CoRR*, abs/0710.4720, 2007.

[57] W. E. Dickinson and R. M. Walker. Reliability Improvement by the Use of Multiple-element Switching Circuits. *IBM J. Res. Dev.*, 2(2):142–147, April 1958. ISSN 0018-8646. doi:10.1147/rd.22.0142.

[58] William R. Dieter and James E. Lumpp, Jr. User-level checkpointing for linuxthreads programs. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 81–92. USENIX Association, Berkeley, CA, USA, 2001. ISBN 1-880446-10-3.

[59] Victorita Dolean, Stephane Lanteri, and Ronan Perrussel. A domain decomposition method for solving the three-dimensional time-harmonic Maxwell equations discretized by discontinuous Galerkin methods. *Journal of Computational Physics*, 227(3):2044–2072, 2008. doi:10.1016/j.jcp.2007.10.004. Also published in J. Comput. Phys., Vol. 227, No. 3 pp. 2044-2072 (2007) .

[60] Jason Duell. The design and implementation of Berkeley Lab's linux checkpoint/restart. Technical report, 2003.

[61] Clément Durochat, Stéphane Lanteri, and Raphaël Léger. A non-conforming multi-element dgtd method for the simulation of human exposure to electromagnetic waves. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, 27(3):614–625, 2014. ISSN 1099-1204. doi:10.1002/jnm.1943.

[62] G.A El-Sayed and K.A Hossny. A distributed counter-based non-blocking coordinated checkpoint algorithm for grid computing applications. In *Advances in Computational Tools for Engineering Applications (ACTEA), 2012 2nd International Conference on*, pages 80–85. Dec 2012. doi:10.1109/ICTEA.2012.6462909.

[63] E.N. Elnozahy, D.B. Johnson, and W. Zwaenepoel. The performance of consistent checkpointing. In *Reliable Distributed Systems, 1992. Proceedings., 11th Symposium on*, pages 39–47. Oct 1992. doi:10.1109/RELDIS.1992.235144.

[64] Graham E Fagg, Edgar Gabriel, Zizhon Chen, Thara Angskun, George Bosilca, Antonin Bukovsky, and Jack J Dongarra. Fault tolerant communication library and applications for high performance computing. In *LACSI Symposium*, pages 27–29. 2003.

[65] M. Faverge. *Ordonnancement hybride statique-dynamique en algèbre linéaire creuse pour de grands clusters de machines NUMA et multi-coeurs.* Ph.D. thesis, LaBRI, Université Bordeaux I, Talence, France, December 2009.

[66] Robert Fitzgerald and Richard F. Rashid. The integration of virtual memory management and interprocess communication in accent. *ACM Trans. Comput. Syst.*, 4(2):147–177, May 1986. ISSN 0734-2071. doi:10.1145/214419.214422.

[67] Diederik R. Fokkema, Gerard L. G. Sleijpen, and Henk A. Van der Vorst. Jacobi-Davidson style QR and QZ algorithms for the partial reduction of matrix pencils. *SIAM J. SCI. COMPUT*, 20:94–125, 1996.

[68] Message Passing Interface Forum. MPI: A message passing interface standard. *International Journal of Supercomputer Applications*, 8(3/4):159–416, 1994.

[69] Valérie Frayssé, Luc Giraud, and Serge Gratton. Algorithm 881: A set of FGMRES routines for real and complex arithmetics on high performance computers. *ACM Transactions on Mathematical Software*, 35(2):1–12, 2008.

[70] Melina A. Freitag and Alastair Spence. Shift-invert arnoldi's method with preconditioned iterative solves. *SIAM J. Matrix Anal. Appl.*, 31(3):942–969, August 2009. ISSN 0895-4798. doi:10.1137/080716281.

[71] J. Gaidamour and P. Hénon. A parallel direct/iterative solver based on a schur complement approach. *2013 IEEE 16th International Conference on Computational Science and Engineering*, 0:98–105, 2008. doi:http://doi.ieeecomputersociety.org/10.1109/CSE.2008.36.

[72] Ana Gainaru, Franck Cappello, Marc Snir, and William Kramer. Fault prediction under the microscope: A closer look into HPC systems. *SC Conference*, 0:1–11, 2012. ISSN 2167-4329. doi:http://doi.ieeecomputersociety.org/10.1109/SC.2012.57.

[73] Al Geist and Christian Engelmann. Development of naturally fault tolerant algorithms for computing on 100,000 processors. 2002.

[74] R. Gioiosa, J.C. Sancho, S. Jiang, and F. Petrini. Transparent, incremental checkpointing at kernel level: a foundation for fault tolerance for parallel computers. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, pages 9–9. Nov 2005. doi:10.1109/SC.2005.76.

[75] L. Giraud, A. Haidar, and L. T. Watson. Parallel scalability study of hybrid preconditioners in three dimensions. *Parallel Computing*, 34:363–379, 2008.

[76] Luc Giraud and A. Haidar. Parallel algebraic hybrid solvers for large 3D convection-diffusion problems. *Numerical Algorithms*, 51(2):151–177, 2009.

[77] N.R. Gottumukkala, R. Nassar, M. Paun, C.B. Leangsuksun, and S.L. Scott. Reliability of a system of k nodes for high performance computing applications. *Reliability, IEEE Transactions on*, 59(1):162–169, March 2010. ISSN 0018-9529. doi: 10.1109/TR.2009.2034291.

[78] Jim Gray. Notes on data base operating systems. In *Operating Systems, An Advanced Course*, pages 393–481. Springer-Verlag, London, UK, UK, 1978. ISBN 3-540-08755-9.

[79] William Gropp and Ewing Lusk. Fault tolerance in MPI programs. *Special issue of the Journal High Performance Computing Applications (IJHPCA)*, 18:363–372, 2002.

[80] A. Guermouche, T. Ropars, E. Brunet, M. Snir, and F. Cappello. Uncoordinated checkpointing without domino effect for send-deterministic MPI applications. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 989–1000. May 2011. ISSN 1530-2075. doi:10.1109/IPDPS.2011.95.

[81] Prashasta Gujrati, Yawei Li, Zhiling Lan, Rajeev Thakur, and John White. A Meta-Learning Failure Predictor for Blue Gene/L Systems. *2013 42nd International Conference on Parallel Processing*, 0:40, 2007. ISSN 0190-3918. doi:http://doi.ieeecomputersociety.org/10.1109/ICPP.2007.9.

[82] G.W. Stewart. *Matrix algorithms – Volume II: Eigensystems*. SIAM, 2001.

[83] A. Haidar. *On the parallel scalability of hybrid solvers for large 3D problems*. Ph.D. dissertation, INPT, June 2008. TH/PA/08/57.

[84] R. W. HAMMING. Error detecting and error correcting codes. *Bell System Technical Journal*, 29:147–160, 1950.

[85] P. Hénon, P. Ramet, and J. Roman. PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems. *Parallel Computing*, 28(2):301–321, January 2002.

[86] Pascal Hénon and Yousef Saad. A parallel multistage ILU factorization based on a hierarchical graph decomposition. *SIAM J. Sci. Comput.*, 28(6):2266–2293, December 2006. ISSN 1064-8275. doi:10.1137/040608258.

[87] M. R. Hestenes and E. Stiefel. Methods of Conjugate Gradients for Solving Linear System. *J. Res. Nat. Bur. Stds.*, B49:409–436, 1952.

[88] D. Higham and N. Higham. Structured backward error and condition of generalized eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 20(2):493–512, 1998. doi:10.1137/S0895479896313188.

[89] 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.

[90] R.K. Iyer, N.M. Nakka, Z.T. Kalbarczyk, and S Mitra. Recent advances and new avenues in hardware-level reliability support. *Micro, IEEE*, 25(6):18–29, Nov 2005. ISSN 0272-1732. doi:10.1109/MM.2005.119.

[91] C. G. J. Jacobi. über ein leichtes Verfahren, die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen. *J. Reine Angew. Math.*, 30:51–94, 1846.

[92] Zhongxiao Jia and G.W. Stewart. An analysis of the Rayleigh-Ritz method for approximating eigenspaces. *Math. Comp*, 1999.

[93] David B Johnson and Willy Zwaenepoel. *Sender-based message logging*. 1987.

[94] James V. Jones. *Integrated Logistics Support Handbook*. TAB Books, Blue Ridge Summit, PA, USA, 1987. ISBN 0-8306-2921-1.

[95] J.-Y. Jou and J.A. Abraham. Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures. *Proceedings of the IEEE*, 74(5):732–741, May 1986. ISSN 0018-9219. doi:10.1109/PROC.1986.13535.

[96] J. Karlsson, P. Liden, P. Dahlgren, R. Johansson, and U. Gunneflo. Using heavy-ion radiation to validate fault-handling mechanisms. *Micro, IEEE*, 14(1):8–23, Feb 1994. ISSN 0272-1732. doi:10.1109/40.259894.

[97] George Karypis and Vipin Kumar. Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0. 1995.

[98] R. Kumar, P. Jovanovic, and I. Polian. Precise fault-injections using voltage and temperature manipulation for differential cryptanalysis. In *On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International*, pages 43–48. July 2014. doi: 10.1109/IOLTS.2014.6873670.

[99] X. Lacoste. *Scheduling and memory optimizations for sparse direct solver on multi-core/multi-GPU cluster systems*. Ph.D. thesis, LaBRI, Université Bordeaux, Talence, France, February 2015.

[100] Cornelius Lanczos. An iterative method for the solution of the eigenvalue problem of linear differential and integral, 1950.

[101] 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.

[102] Lie-Quan Lee, Zenghai Li, Cho Ng, Kwok Ko, et al. Omega3P: a parallel finite-element eigenmode analysis code for accelerator cavities. 2009.

[103] R. B. Lehoucq and J. A. Scott. An evaluation of software for computing eigenvalues of sparse nonsymmetric matrices, 1996.

[104] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK: Solution of Large Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods*. Available from netlib@ornl.gov, 1997.

[105] R.B. Lehoucq and D. C. Sorensen. Deflation techniques for an implicitly re-started Arnoldi iteration. *SIAM J. Matrix Anal. Appl*, 17:789–821, 1996.

[106] C.-C.J. Li and W.K. Fuchs. Catch-compiler-assisted techniques for checkpointing. In *Fault-Tolerant Computing, 1990. FTCS-20. Digest of Papers., 20th International Symposium*, pages 74–81. June 1990. doi:10.1109/FTCS.1990.89337.

[107] K. Li, J. F. Naughton, and J. S. Plank. Low-latency, concurrent checkpointing for parallel programs. *IEEE Trans. Parallel Distrib. Syst.*, 5(8):874–879, August 1994. ISSN 1045-9219. doi:10.1109/71.298215.

[108] X S Li, M Shao, I Yamazaki, and E G Ng. Factorization-based sparse solvers and preconditioners. *Journal of Physics: Conference Series*, 180(1):012015, 2009.

[109] X Sherry Li, M Shao, I Yamazaki, and EG Ng. Factorization-based sparse solvers and preconditioners. In *Journal of Physics: Conference Series*, volume 180, page 012015. IOP Publishing, 2009.

[110] Xiaoye S. Li and James W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Math. Softw.*, 29(2):110–140, June 2003. ISSN 0098-3500. doi:10.1145/779359.779361.

[111] Yinglung Liang, Yanyong Zhang, Anand Sivasubramaniam, Morris Jette, and Ramendra Sahoo. BlueGene/L Failure Analysis and Prediction Models. *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 0:425–434, 2006. doi:http://doi.ieeecomputersociety.org/10.1109/DSN.2006.18.

[112] Yudan Liu, R. Nassar, C.B. Leangsuksun, N. Naksinehaboon, M. Paun, and S.L. Scott. An optimal checkpoint/restart model for a large scale high performance computing system. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–9. April 2008. ISSN 1530-2075. doi: 10.1109/IPDPS.2008.4536279.

[113] 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.

[114] Casas Marc, Bronevetsky Greg, Labarta Jesus, and Valero Mateo. Dealing with faults in HPC systems. In *PMAA-International Workshop on Parallel Matrix Algorithms and Applications*. Lugano, Switzerland, July 2014.

[115] Tarek Mathew. *Domain Decomposition Methods for the Numerical Solution of Partial Differential Equations (Lecture Notes in Computational Science and Engineering)*. Springer Publishing Company, Incorporated, 1 edition, 2008. ISBN 3540772057, 9783540772057.

[116] N. Nakka, K. Pattabiraman, and R. Iyer. Processor-Level Selective Replication. In *Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP International Conference on*, pages 544–553. June 2007. doi:10.1109/DSN.2007.75.

[117] Thomas Naughton, Wesley Bland, Geoffroy Vallee, Christian Engelmann, and Stephen L. Scott. Fault injection framework for system resilience evaluation: Fake faults for finding future failures. In *Proceedings of the 2009 Workshop on Resiliency in High Performance*, Resilience '09, pages 23–28. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-593-2. doi:10.1145/1552526.1552530.

[118] T.J. O'Gorman, J. M. Ross, A H. Taber, J.F. Ziegler, H.P. Muhlfeld, C.J. Montrose, H. W. Curtis, and J.L. Walsh. Field testing for cosmic ray soft errors in semiconductor memories. *IBM Journal of Research and Development*, 40(1):41–50, Jan 1996. ISSN 0018-8646. doi:10.1147/rd.401.0041.

[119] N. Oh, P.P. Shirvani, and E.J. McCluskey. Error detection by duplicated instructions in super-scalar processors. *Reliability, IEEE Transactions on*, 51(1):63–75, Mar 2002. ISSN 0018-9529. doi:10.1109/24.994913.

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

[121] Behrooz Parhami. Detect, fault, error, ... , or failure. *IEEE Transactions on Reliability*, 46, December 1997. ISSN 0018-9529.

[122] Stefan Pauli and Peter Arbenz. Determining optimal multilevel Monte Carlo parameters with application to fault tolerance. Research report, November 2014. doi: 10.3929/ethz-a-010335876.

[123] Stefan Pauli, Manuel Kohler, and Peter Arbenz. A fault tolerant implementation of Multi-Level Monte carlo methods. In *PARCO'13*, pages 471–480. 2013.

[124] J. S. Plank, K. Li, and M. A. Puening. Diskless checkpointing. *IEEE Transactions on Parallel and Distributed Systems*, 9(10):972–986, October 1998.

[125] J. S. Plank, J. Xu, and R. H. B. Netzer. Compressed differences: An algorithm for fast incremental checkpointing. Technical Report CS-95-302, University of Tennessee, August 1995.

[126] James S. Plank, Youngbae Kim, and Jack J. Dongarra. Fault-tolerant matrix operations for networks of workstations using diskless checkpointing. *J. Parallel Distrib. Comput.*, 43:125–138, June 1997. ISSN 0743-7315.

[127] J.S. Plank and K. Li. ICKP: a consistent checkpointer for multicomputers. *Parallel Distributed Technology: Systems Applications, IEEE*, 2(2):62–67, Summer 1994. ISSN 1063-6552. doi:10.1109/88.311574.

[128] J.S. Plank and Kai Li. Faster checkpointing with N+1 parity. In *Fault-Tolerant Computing, 1994. FTCS-24. Digest of Papers., Twenty-Fourth International Symposium on*, pages 288–297. June 1994. doi:10.1109/FTCS.1994.315631.

[129] Katerina Goseva Popstojanova and Kishor Trivedi. Failure correlation in software reliability models. *IEEE Trans. on Reliability*, 49, 2000.

[130] G. Radicati and Y. Robert. Parallel conjugate gradient-like algorithms for solving nonsymmetric linear systems on a vector multiprocessor. *Parallel Computing*, 11:223–239, 1989.

[131] S. Rajamanickam, E. G. Boman, and M. A. Heroux. ShyLU: A hybrid-hybrid solver for multicore platforms. *Parallel and Distributed Processing Symposium, International*, 0:631–643, 2012. ISSN 1530-2075. doi:http://doi.ieeecomputersociety.org/10.1109/IPDPS.2012.64.

[132] Narasimha Raju, Gottumukkala, Yudan Liu, Chokchai B. Leangsuksun, Raja Nassar, and Stephen Scott2. Reliability Analysis in HPC clusters. *Proceedings of the High Availability and Performance Computing Workshop*, 2006.

[133] S. Rao, L. Alvisi, and H.M. Vin. The cost of recovery in message logging protocols. *Knowledge and Data Engineering, IEEE Transactions on*, 12(2):160–173, Mar 2000. ISSN 1041-4347. doi:10.1109/69.842260.

[134] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial & Applied Mathematics*, 8(2):300–304, 1960.

[135] J. Rexford and N.K. Jha. Algorithm-based fault tolerance for floating-point operations in massively parallel systems. In *Circuits and Systems, 1992. ISCAS '92. Proceedings., 1992 IEEE International Symposium on*, volume 2, pages 649–652 vol.2. May 1992. doi:10.1109/ISCAS.1992.230168.

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

[137] Y. Saad and M. H. Schultz. GMRES: A Generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Scientific and Statistical Computing*, 7:856–869, 1986.

[138] Youcef Saad. Projection methods for solving large sparse eigenvalue problems. In *Matrix Pencils*, pages 121–144. Springer, 1983.

[139] Youcef Saad. Overview of krylov subspace methods with applications to control problems, 1990.

[140] Youcef Saad and Martin H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, July 1986. ISSN 0196-5204.

[141] Yousef Saad. *Numerical Methods for Large Eigenvalue Problems*. Manchester University Press, Manchester, UK, 1992.

[142] Pablo Salas. *Physical and numerical aspects of thermoacoustic instabilities in annular combustion chambers*. Ph.D. thesis, Université Bordeaux 1, November 2013.

[143] Pablo Salas, Luc Giraud, Yousef Saad, and Stéphane Moreau. Spectral recycling strategies for the solution of nonlinear eigenproblems in thermoacoustics. Research Report RR-8542, INRIA, May 2014.

[144] Olaf Schenk, Klaus Gärtner, Wolfgang Fichtner, and Andreas Stricker. PARDISO: A high-performance serial and parallel sparse linear solver in semiconductor device simulation, 2000.

[145] M. Scholzel. Reduced Triple Modular redundancy for built-in self-repair in VLIW-processors. In *Signal Processing Algorithms, Architectures, Arrangements and Applications, 2007*, pages 21–26. Sept 2007. doi:10.1109/SPA.2007.5903294.

[146] Bianca Schroeder and Garth A. Gibson. Poster reception - the computer failure data repository (CFDR): collecting, sharing and analyzing failure data. In *SC*, page 154. ACM Press, 2006. ISBN 0-7695-2700-0.

[147] Bianca Schroeder and Garth A. Gibson. A Large-Scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing*, 7(4):337–351, 2010. ISSN 1545-5971. doi:http://doi.ieeecomputersociety.org/10.1109/TDSC.2009.4.

[148] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. DRAM errors in the wild: A large-scale field study. *SIGMETRICS Perform. Eval. Rev.*, 37(1):193–204, June 2009. ISSN 0163-5999. doi:10.1145/2492101.1555372.

[149] Manu Shantharam, Sowmyalatha Srinivasmurthy, and Padma Raghavan. Characterizing the impact of soft errors on iterative methods in scientific computing. In *Proceedings of the international conference on Supercomputing*, pages 152–161. ACM, 2011.

[150] H. Sharangpani and H. Arora. Itanium processor microarchitecture. *Micro, IEEE*, 20(5):24–43, Sep 2000. ISSN 0272-1732. doi:10.1109/40.877948.

[151] Gerard L. G. Sleijpen and Henk A. Van der. A Jacobi–Davidson iteration method for linear eigenvalue problems. *SIAM Rev.*, 42(2):267–293, June 2000.

[152] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, Pavan Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, Andrew A. Chien, P. Coteus, N. A. Debardeleben, P. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, Sriram Krishnamoorthy, Sven Leyffer, D. Liberty, S. Mitra, T. S. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen. Addressing failures in exascale computing. 2013. ISSN ANL/MCS-TM-332.

[153] Rob Strom and Shaula Yemini. Optimistic recovery in distributed systems. *ACM Trans. Comput. Syst.*, 3(3):204–226, August 1985. ISSN 0734-2071. doi:10.1145/3959.3962.

[154] Keita Teranishi and Michael A. Heroux. Toward local failure local recovery resilience model using MPI-ULFM. In *Proceedings of the 21st European MPI Users' Group Meeting*, EuroMPI/ASIA '14, pages 51:51–51:56. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2875-3. doi:10.1145/2642769.2642774.

[155] N.F. Vaidya. A case for two-level recovery schemes. *Computers, IEEE Transactions on*, 47(6):656–666, Jun 1998. ISSN 0018-9340. doi:10.1109/12.689645.

[156] T.N. Vijaykumar, I Pomeranz, and K. Cheng. Transient-fault recovery using simultaneous multithreading. In *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*, pages 87–98. 2002. ISSN 1063-6897. doi:10.1109/ISCA.2002.1003565.

[157] John von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies*, pages 43–98, 1956.

[158] Chao Wang, Frank Mueller, Christian Engelmann, and Stephen L. Scott. Hybrid full/incremental checkpoint/restart for MPI jobs in HPC environments. In *Dept. of Computer Science, North Carolina State University*. 2009.

[159] Yi-Min Wang and W.K. Fuchs. Lazy checkpoint coordination for bounding rollback propagation. In *Reliable Distributed Systems, 1993. Proceedings., 12th Symposium on*, pages 78–85. Oct 1993. doi:10.1109/RELDIS.1993.393471.

[160] Yi-Min Wang, Yennun Huang, Kiem-Phong Vo, Pe-Yu Chung, and C. Kintala. Checkpointing and its applications. In *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on*, pages 22–31. June 1995. doi: 10.1109/FTCS.1995.466999.

[161] Chris Weaver and Todd M. Austin. A Fault Tolerant Approach to Microprocessor Design. In *Proceedings of the 2001 International Conference on Dependable Systems and Networks (Formerly: FTCS)*, DSN '01, pages 411–420. IEEE Computer Society, Washington, DC, USA, 2001. ISBN 0-7695-1101-5.

[162] Wallodi Weibull. A statistical distribution function of wide applicability. *Journal of Applied Mechanics*, 18:293–297, 1951.

[163] James H. Wilkinson. *Rounding Errors in Algebraic Processes*. Dover Publications, Incorporated, 1994. ISBN 0486679993.

[164] J.H. Wilkinson. *The Algebraic Eigenvalue Problem*. Monographs on numerical analysis. Clarendon Press, 1988. ISBN 9780198534181.

[165] J. Xu, Z. Kalbarczyk, and R.K. Iyer. Networked windows nt system field failure data analysis. In *Dependable Computing, 1999. Proceedings. 1999 Pacific Rim International Symposium on*, pages 178–185. 1999. doi:10.1109/PRDC.1999.816227.

[166] Ichitaro Yamazaki and Xiaoye S. Li. On techniques to improve robustness and scalability of a parallel hybrid linear solver. In *VECPAR*, pages 421–434. 2010.

[167] Gengbin Zheng, Xiang Ni, and Laxmikant V Kalé. A scalable double in-memory checkpoint and restart scheme towards exascale. In *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*, pages 1–6. IEEE, 2012.