EDSPI - UNIVERSITÉ DES SCIENCES & TECHNOLOGIES DE LILLE

Mémoire de Thèse pour obtenir le titre de

Docteur en Informatique

Présenté par

France BOILLOD-CERNEUX

Nouveaux algorithmes numériques pour l'utilisation efficace des architectures de calcul multi-coeurs et hétérogènes

Soutenue le 13 Octobre 2014

Jury :

Rapporteurs :	Michel DAYDE	-	Professeur à l'ENSEEIHT,
	Michael HEROUX	-	Directeur de Recherche, Sandia National Laboratories
Directeur de Thèse :	Serge PETITON	-	Professeur à l'Université Lille 1 des Sciences et Technologies
Encadrant de Thèse :	Christophe CALVIN	-	Docteur en Mathématiques Appliquées, HDR en Informatique, CEA Saclay
Examinateurs :	Nahid EMAD	-	Professeur à l'Université de Versailles Saint Quentin en Yvelines
	Satoshi MATSUOKA	-	Professeur à l'Institut de Technologies de Tokyo
	Marc SNIR	-	Directeur de la Division Mathematiques & Informatique, Argonne National Laboratory

EDSPI - SCIENCES & TECHNOLOGIES LILLE UNIVERSITY

PhD Thesis to obtain the title of

PhD of Computer Science

Defended by

France BOILLOD-CERNEUX

New Algorithms for Efficient Use of Multi-Cores and Heterogeneous Architectures

Defended on October the 13^{th}, 2014

Jury :

Reviewers :	Michel DAYDE	-	Professor from ENSHEEIT
	Michael HEROUX	-	Distinguished Member of the Department of the Scalable Algorithms, Sandia National Laboratories
Ph.D Director :	Serge PETITON	-	Professor from Sciences & Technologies Lille 1 University
Ph.D Advisor :	Christophe CALVIN	-	Applied Mathmatics PhD, Computer Science HDR, CEA Saclay
Examinators :	Nahid EMAD	-	Professor from Versailles Saint Quentin en Yvelines University
	Satoshi MATSUOKA	-	Professor from Tokyo Institute Technology
	Marc SNIR	-	Director of the Mathematics and Computer Science Division, Argonne National Laboratory

Résumé

Depuis la naissance des supercalculateurs jusqu'à l'arrivée de machines pétaflopiques, les technologies utilisées n'ont cessé d'évoluer à une vitesse fulgurante. Cette course à la performance se heurte aujourd'hui au passage à l'Exascale, qui se démarque des autres échelles par les difficultés qu'elle impose pour la franchir. Les conséquences qui en découlent bouleversent tous les domaines scientifiques relatifs au Calcul Haute Performance (HPC).

Nous nous plaçons dans le contexte des problèmes à valeurs propres, largement répandus: du page ranking aux simulation de systèmes nucléaires, astronomie, explorations pétrolifères...

Notre démarche comporte deux thématiques complémentaires: Nous proposons d'étudier puis d'améliorer la convergence de la méthode de recherche de valeurs propres *Explicitly Restarted Arnoldi Method (ERAM)* en réutilisant les informations générées. L'étude de la convergence et sa caractérisation sont indispensables pour pouvoir mettre en place des techniques de Smart-Tuning, c'est à dire l'adaptation durant l'éxécution parallèle de certains paramètres numériques ou informatique (au sens large) de la méthode considérée. La phase d'amélioration consiste à utiliser les valeurs de Ritz de manière efficace afin d'accélerer la convergence de la méthode sans coûts supplémentaires en terme de communications parallèles ou de stockage mémoire, paramètres indispensables pour les machines multi-coeurs et hétérogènes.

Enfin, nous etudions deux méthodes pour générer des matrices de très large dimensions aux spectres imposés afin de constituer une collection de matrices de tests qui seront partagées avec la communauté du HPC. Ces matrices servent à valider numériquement des solveurs de systèmes à valeurs propres d'une part, et d'autre part de pouvoir évaluer leur performances parallèles grâce à leur propriétés adaptées aux machines petaflopiques et au-delà.

Mots Clés: Architectures Hétérogènes et Homogènes, Communications Asynchrones, Extreme-Scale, Méthodes de Krylov, Co-Méthodes, Méthodes Hybrides, Problèmes à valeurs Propres, Auto/Smart-Tuning, Stratégies de Redémarrage.

Summary

Supercomputing architectures and programming paradigms have dramatically evolve during the last decades. Since we reached the petaflop scale, we forecast to overcome the exaflop scale. Crossing this new scale implies many drastic changes, concerning the overall High Performance Computing (HPC) scientific fields.

In this Thesis, we focus on the eigenvalue problems, implied in most of the industrial simulations. As an non-exhaustive list, we can cite page rank, nuclear physics simulations, oil & gas exploration, astronomy ...

This thesis focuses on two complementary topics, where we propose to study and improve the Explicitly Restarted Arnoldi Method (ERAM) convergence. The first step is to study and characterize the ERAM convergence: such metric is the basis of every upcoming smart/auto-tuning heuristic. We propose to ameliorate the ERAM convergence by using efficiently the Ritz eigenpairs without any additional parallel communications nor memory storage, two key points in the ultra-scale computing context.

We then study two matrix generators, starting from a user-imposed spectrum. Such collections of matrix are used to numerically check and approve extreme-scale eigensolvers, as well as measure and improve their parallel performances on ultra-scale supercomputers.

Keywords: Heterogeneous and Homogeneous Architectures, Asynchronous Communication, Extrem-Scale Computing, Krylov Methods, Co-Methods, Hybrid Methods, Eigenvalue Systems, Linear Systems, Auto/Smart-Tuning, Restarting Strategies, Restarting Vector.

à Anthony,
à ma famille.

Remerciements

Je remercie infiniment mon directeur de thèse Serge PETITON, Professeur à l'Université des Sciences et Technologies de Lille. Merci de m'avoir donné ma chance et de m'avoir suivie pendant ces trois belle années. Merci pour votre soutien, merci de m'avoir fait confiance.

Je remercie particulièrement mon encadrant de thèse Christophe CALVIN, Chef de Projet en Simulation Thermohydraulique au CEA Saclay. Merci pour ton expertise, merci de m'avoir accompagnée dans mes recherches et enfin, merci pour ta patience.

Serge, Christophe, il me semble que c'était à peine hier ou je vous demandais fébrilement de réaliser mon premier stage sous votre tutelle.

Je tiens à remercier Tony DRUMMOND, Computer Scientist au Lawrence Berkeley National Laboratory. Tony merci pour ton encadrement à Berkeley, ta gentillesse et ton accueil.

Je remercie Nahid EMAD, pour m'avoir donné l'occasion d'enseigner, cette expèrience m'a beaucoup apporté et c'est grâce à toi.

Je remercie chaleureusement Michel DAYDE, Professeur à l'ENSEEIHT et Directeur de l'IRIT ainsi que Michael HEROUX, Chercheur distingué au Sandia National Laboratories, d'avoir accepté d'être rapporteurs de ma thèse. J'apprécie vos recommandations et votre aide, merci infiniment.

Mes remerciements les plus sincères vont également à Marc Snir, merci d'avoir été présent à mon jury de thèse, et plus encore, un grand merci pour vos précieux conseils.

Je remercie Satoshi MATSUOKA Professeur à l'Université Tokyo Institute of Technology, pour sa participation à ma soutenance de thèse. Les discussions partagées avec vous étaient riches.

Fan, ces deux années passées avec toi étaient riches professionnellement et personnellement.

Mes remerciements vont à l'équipe de la Maison de la Simulation, pour leur accueil et leur bonne humeur quotidienne.

Contents

Introduction **1**

2 The Extreme-Scale Challenge **5**
2.1 Supercomputing Rise until the Petaflop scale . . . 5
2.1.1 Supercomputers History, Past & Present . . . 6
2.2 The Exascale Energy Wall . . . 7
2.2.1 The Energy Consumption Metric as Reference . . . 9
2.3 The Extrem-Scale Supercomputers Architectures . . . 9
2.3.1 Toward Hardware-Aware Algorithms . . . 11
2.4 The New Programming Paradigms to Overcome the Exascale Challenge . . . 11
2.4.1 Manage the Hardware Dependencies . . . 12
2.4.2 Algorithms Adapted to Multi-Level Parallelism . . . 12
2.5 The Resiliency as a Key Parameter for the Exascale Computing . . . 13
2.5.1 Ensure and Manage the Operating Systems Resiliency . . . 13
2.5.2 Languages & Numerical Libraries to Support the Resiliency . . . 13
2.5.3 Toward Resilient Algorithms . . . 14
2.6 The Mathematics Challenge for the Exascale Wall . . . 14
2.6.1 The Eigenvalue Problems, Still Alive . . . 16

3 Krylov Method for Eigenvalue Problems **21**
3.1 The Arnoldi Method . . . 21
3.1.1 The Arnoldi Method Convergence . . . 23
3.1.2 The Arnoldi Method Accuracy Dependancies . . . 23
3.1.3 The Arnoldi Method with Incomplete Orthogonalization . . . 25
3.2 The Restarted Arnoldi Method . . . 26
3.3 The Explicitly Restarted Arnoldi Method . . . 29
3.4 The Explicitly Restarted Arnoldi Method with Deflation . . . 31
3.5 The Implicitly Restarted Arnoldi Method . . . 33
3.6 The Krylov Eigen Solvers . . . 34
3.6.1 Fixing the Restarted Arnoldi Method Parameters . . . 34
3.7 The Multiple Restarted Arnoldi Method . . . 35
3.7.1 Resiliency Property . . . 37
3.7.2 Reduce the Global and Blocking Communications . . . 38
3.7.3 Multiple Levels of Parallelism . . . 39
3.7.4 Exploit Heterogeneous Architectures . . . 39
3.8 The Multiple Restarted Arnoldi Method Variants . . . 40
3.8.1 The Multiple Explicitly Restarted Arnoldi Method . . . 40
3.8.2 The Multiple Implicitly Restarted Arnoldi Method . . . 41

4 Describe and Characterize the ERAM Convergence 45
4.1 Define the ERAM Convergence . . . 45
4.1.1 The Convergence Criteria . . . 45
4.1.2 Convergence Study applied to the Krylov Method for Linear System 46
4.2 Detect and Define the ERAM Convergence at the Runtime Execution . . . 47
4.2.1 The ERAM Local Convergence . . . 48
4.2.2 The ERAM Multi-Levels Convergence . . . 48

5 Matrix Generators for Extreme-Scale Computing 63
5.1 Extreme-Scale Eigenvalue Solvers Performances Evaluation . . . 63
5.1.1 The Existing Collections . . . 64
5.1.2 The Spectrum as the Key Parameter . . . 64
5.1.3 The Matrix Generator Computational Hypothesis . . . 66
5.2 The Dense Matrix Generator . . . 66
5.2.1 The Dense Matrix Generator Results . . . 69
5.3 The Diagonal-Band Matrices Generator . . . 74
5.3.1 Diagonal-Band Matrix Generator Results . . . 76

6 Restarting Strategies Influence on the ERAM Convergence 89
6.1 ERAM Restarting Strategies . . . 89
6.1.1 The Restarting Strategies . . . 89
6.2 The ERAM Restarting Strategies Convergence . . . 91
6.2.1 The Restarting Strategy Efficiency . . . 102
6.3 The Twin Restarting Strategies . . . 104

7 ERAM with Mixed-Restarting Strategies 109
7.1 The ERAM Restarting Vector with Mixed Restarting Strategies . . . 109
7.2 The ERAM with Mixed Restarting Strategies Experimentations . . . 111

8 Toward an ERAM Restarting Strategies Auto-Tuning 129
8.1 The ERAM with Auto-Tuned Restarting Strategies . . . 129
8.2 On the Choose of the ERAM Auto-Tuned Parameters . . . 130
8.2.1 The ERAM with Mixed Restarting Strategies Auto-Tuning Prerequisite . . . 132
8.2.2 The ERAM with Mixed Restarting Strategies Auto-Tuning Choice . 134
8.3 A Basic Mixed Restarting Strategy Tuning Addressed to the ERAM . . . 135
8.3.1 Basic Mixed Restarting Strategy Tuning Results . . . 136
8.3.2 Improving the ERAM with Mixed-Restarting Strategy Auto-Tuning 153
8.4 Compute the Restarting Vector According to the ERAM Mixed Restarting Strategy . . . 154
8.4.1 Erase the Cyclic Krylov Subspaces . . . 157
8.4.2 Mixed Restarting Strategy & Best Ritz Eigenpairs Tuning Results . 158

Bibliography 183

Communications **193**

A The Dense Matrix Generator Results **195**

B The ERAM with Mixed-Restarting Strategies **201**

C The ERAM with Mixed-Restarting Strategies Smart-Tuning **209**

List of Figures

2.1 Peak Floatting Point Operations per Watt, Double Precision 10

4.1 Mixtank_new Matrix Sparsity 51
4.2 Ex11 Matrix Sparsity 51
4.3 Rim Matrix Sparsity 52
4.4 Ex11 Matrix, m=20, Algorithm 10 Results. 53
4.5 Ex11 Matrix, m=15, Algorithm 10 Results. 54
4.6 Ex11 Matrix, m=25, Algorithm 10 Results. 55
4.7 *Mixtank_new* Matrix, m=15, Algorithm 10 Results. 56
4.8 *Mixtank_new* Matrix, $m = 25$, Algorithm 10 Results. 57
4.9 Rim Matrix, $m = 25$, Algorithm 10 Results. 58
4.10 Rim Matrix, $m = 30$, Algorithm 10 Results. 59

5.1 The Fission Matrix Spectrum, Size is $10,000$ 70
5.2 The $Fission_{2^{15}}$ Matrix Spectrum, Size is 2^{15} 71
5.3 Dense Fission Matrix Spectrum, SLEPc Eigen Solvers Numerical Comparison 72
5.4 Dense $Fission_{2^{15}}$ Matrix Extended Spectrum, SLEPc Eigen Solvers Numerical Comparison 73
5.5 Spectrum $Spec_{2^{15}}$ 77
5.6 $Spec_{2^{15}}$ $M_t{}^1$, SLEPc Krylov-Schur 79
5.7 $Spec_{2^{15}}$ $M_t{}^1$, SLEPc Arnoldi 79
5.8 $Spec_{2^{15}}$ $M_t{}^2$, SLEPc Krylov-Schur 80
5.9 $Spec_{2^{15}}$ $M_t{}^2$, SLEPc Arnoldi 80
5.10 $Spec_{2^{15}}$ $M_t{}^4$, SLEPc Krylov-Schur 81
5.11 $Spec_{2^{15}}$ $M_t{}^4$, SLEPc Arnoldi 81
5.12 Diagonal Band Matrices, SLEPc Krylov-Schur Accuracy 83
5.13 Diagonal Band Matrices, SLEPc Arnoldi Accuracy 84

6.1 *Ex*11 Matrix, ERAM Restarting Strategies Convergence. 93
6.2 *Ex*11 Dense Matrix, ERAM Restarting Strategies Convergence with respect to the number of restarts. 94
6.3 *Fission* Dense Matrix, Restarting Strategies Convergence. 96
6.4 $Fission_{2^{15}}$ Dense Matrix, ERAM Restarting Strategies Convergence. . . . 97
6.5 *Mixtank_new* Matrix, Restarting Strategies Convergence. 98
6.6 *Mixtank_new* Dense Matrix, Restarting Strategies Convergence. 99
6.7 *Rim* Matrix, Restarting Strategies Convergence. 100
6.8 *Rim* Dense Matrix, Restarting Strategies Convergence. 101
6.9 *Mixtank_new* Dense, α_{Res} & α_{Def}. 105
6.10 *Rim* Dense, α_{Res} & α_{Def}. 105

6.11 *Rim* Dense, α_{La} & α_{LaRes}. 106

7.1 Ex11 Dense Matrix, α_{Def} Mixed Restarting Strategies. 112
7.2 *Fission* Dense Matrix, Mixed Restarting Strategies Convergence with respect to the number of restarts. 115
7.3 *Rim* Dense Matrix, Mixed Restarting Strategies Convergence. 117
7.4 Rim_{Dense} Mixed Restarting Strategies GFlop Gain/Losts. 119
7.5 Rim_{Dense} Mixed Restarting Strategies Time (seconds) Gains/Losts. 120

8.1 Bayer04 Matrix, α_{LaRes} Mixed Restarting Strategies. 133
8.2 *Fission* Dense Matrix, Mixed Restarting Strategies Tuning *Choice 1.* . . . 139
8.3 $Fission_{2^{15}}$ Dense Matrix, Mixed Restarting Strategies Tuning *Choice 1.* . . 142
8.4 *Ex*11 Dense Matrix, Mixed Restarting Strategies Tuning *Choice 1.* 144
8.5 Mixtank_new Dense Matrix, Mixed Restarting Strategies Tuning *Choice 1.* 146
8.6 *Rim* Dense Matrix, Mixed Restarting Strategies Tuning *Choice 1.* 148
8.7 Diagonal Band 2^{16} Matrix, Mixed Restarting Strategies Tuning *Choice 1.* . 150
8.8 Diagonal Band 2^{16} Matrix, Mixed Restarting Strategies Tuning *Choice 1.* . 151
8.9 Diagonal Band 2^{16} Matrix, Mixed Restarting Strategies Tuning *Choice 1.* . 152
8.10 *Ex*11 Dense, Cyclic Krylov Subspace Illustration. 157
8.11 *Fission* Dense Matrix, Mixed Restarting Strategies/Best Ritz Eigenpairs Tuning. 161
8.12 $Fission_{2^{15}}$ Dense Matrix, Mixed Restarting Strategies/Best Ritz Eigenpairs Tuning. 163
8.13 *Ex*11 Dense Matrix, Mixed Restarting Strategies/Best Ritz Eigenpairs Tuning. 165
8.14 Mixtank_new Dense Matrix, Mixed Restarting Strategies/Best Ritz Eigenpairs Tuning. 167
8.15 *Rim* Dense Matrix, Mixed Restarting Strategies/Best Ritz Eigenpairs Tuning. 169
8.16 Diagonal Band 2^{16} Matrix, Mixed Restarting Strategies/Best Ritz Eigenpairs Tuning. 171
8.17 Diagonal Band 2^{16} Matrix, Mixed Restarting Strategies/Best Ritz Eigenpairs Tuning. 172

A.1 The Ex11 Matrix Spectrum, Size is $16,614$ 195
A.2 Dense Ex11 Matrix, SLEPc Eigen Solvers Numerical Comparison 196
A.3 The Mixtank_new Matrix Spectrum, Size is $29,957$ 197
A.4 Dense Mixtank_new Matrix, SLEPc Eigen Solvers Numerical Comparison 198
A.5 The Rim Matrix Spectrum, Size is $22,560$ 199
A.6 Dense Rim Matrix, SLEPc Eigen Solvers Numerical Comparison 200

B.1 Mixtank_new Dense Matrix, α_{Def}/ α_{La} Mixed Restarting Strategies. . . . 202
B.2 Mixtank_new Dense Matrix, α_{Def}/ α_{LaRes} Mixed Restarting Strategies. . . 202
B.3 Mixtank_new Dense Matrix, α_{Def}/ α_{Li} Mixed Restarting Strategies. . . . 203

B.4 Mixtank_new Dense Matrix, α_{Def}/ α_{LiRes} Mixed Restarting Strategies. . . 203

C.1 Diagonal Band 2^{16} Matrix, Mixed Restarting Strategies/Best Ritz Eigenpairs Tuning. 210

List of Tables

2.1 Top500 June 2014 List, the 5^{th} Most Powerful Supercomputers 8
2.2 Top500 June 2014 List, the 5^{th} Most Powerful Supercomputers Peak Performances per Watt. 8

4.1 Target Matrices to Test the Algorithm 10. 51

5.1 Target Matrices to Test the Dense Matrix Generator 69
5.2 Diagonal-Band Matrix $Spec_{2^{15}}$ Characteristics 78
5.3 Diagonal-Band Matrices Characteristics 82

6.1 The Restarting strategies coefficients 90
6.2 Target Matrices to Test the Restarting Strategies. 92
6.3 Diagonal Band Matrix $Spec_{2^{16}}, M_t^4$ Restarting Strategies Convergence . . . 102
6.4 Restarting Strategies Efficiency 103

7.1 Target Matrices to Test the Mixed Restarting Strategies. 110
7.2 Ex11 Dense Matrix, Algorithm 10 Results. 111
7.3 $Ex11_{Dense}$ Mixed Restarting Strategies Efficiency 113
7.4 $Ex11_{Dense}$ Mixed Restarting Strategies Efficiency, Execution Time and GFlop gains 114
7.5 $Fission_{Dense}$ Convergence Status. 115
7.6 $RimDense$ Convergence Status. 116
7.7 Rim_{Dense} Mixed Restarting Strategies Efficiency 118
7.8 Diagonal Band 2^{15} Matrix, ERAM Restarting Strategies. 122
7.9 Diagonal Band 2^{15} Matrix Convergence Status. 123
7.10 Diagonal Band 2^{15} Matrix, Mixed Restarting Strategies Execution Time Gains/Losses. 124
7.11 Diagonal Band 2^{15} Matrix, Mixed Restarting Strategies Execution Time Gains/Losses. 124

8.1 The Target Matrices to Test the ERAM with Mixed Restarting Strategies Auto-Tuning. 137
8.2 Comparison of the Algorithms 17 and 13 Performances. 162

B.1 $Mixtank\_newDense$ Convergence Status. 201
B.2 Diagonal Band Matrix 2^{15} Flop & Memory gains 204
B.3 Diagonal Band 2^{15} Matrix, Mixed Restarting Strategies Mixed Restarting Strategies Gains/Losses. 205
B.4 Diagonal Band 2^{15} , Mixed Restarting Strategies Gains/Losses. 207

List of Algorithms

1 The Arnoldi Method using CGS . . . 22
2 The Arnoldi Method using CGSR . . . 25
3 The Incomplete Arnoldi Method using CGS . . . 26
4 The Restarted m-step Arnoldi Method . . . 29
5 The Explicitly Restarted Arnoldi Method . . . 31
6 The Explicitly Restarted Arnoldi Method with Deflation . . . 32
7 The Implicitly Restarted Arnoldi Method . . . 33
8 The Multiple Restarted Arnoldi Method . . . 36
9 Restart per Restart Convergence Estimation Algorithm . . . 48
10 Multi-Levels Convergence Algorithm . . . 49
11 The Dense Matrix Generator . . . 68
12 The Diagonal-Band Matrix Generator . . . 76
13 Basic Restarting Strategy Tuning ERAM . . . 135
14 ERAM Convergence Snapshot . . . 136
15 ERAM with the Best Ritz Eigenpairs Algorithm . . . 155
16 Erase the Cyclic Krylov Subspaces . . . 158
17 the ERAM with Mixed Restarting Strategy and Best Ritz Eigenpairs Tuning 159

Introduction

In the large shade of scientific applications and simulations, solving an eigenvalue problem is a common occurrence. This diagnosis is not new, as evidenced by the famous title *"150 Years Old and Still Alive: Eigenproblems"* [Vorst 1997].

Recently, eigenproblems have been particularly highlighted with search engines success. The truth is, eigenproblems guide many scientific applications. As a non-exhaustive list, we can mention the Oil & Gas exploration, astronomy/aerospace simulations, climatology, mechanical simulations, nuclear physics modeling... Many other scientific fields can be quoted, the list is certainly not limited to the previous mentioned above.

The *"nature"* of the eigenproblem itself differs, depending on modeled system properties. We will illustrate this purpose on two nuclear reactor physics applications.

The neutron transport is the heart of physical processes in nuclear reactor simulations. The Boltzmann equation (also called neutron transport equation) is a part of the neutron transport simulation. Such equation is solved by using whether deterministic or Monte Carlo methods. As a part of the global system resolution, computing the dominant eigenpair of a large non-Hermitian matrix leads to compute the so-called *"criticality coefficient"*. Among the large palette of eigensolvers, a subset present specific properties, turning them to the most-efficient eigensolvers to compute the dominant eigenpair of a large system.

On the other side, some applications require to compute a (large) subset of dominant eigenpairs of a large non-Hermitian matrix. This is illustrated by the Fission matrix method, used to improve the Monte Carlo process convergence to solve the neutron transport equation: The more eigenpairs we compute, the more we improve the Monte Carlo process convergence. Computing many dominant eigenpairs implies to use eigensolvers that can focus on such specific eigen systems.

Additionally to the mathematical properties of each eigen systems, we have to bear in mind that such eigen solvers address to large simulations, therefore large systems. The computing capability is one of the leading parameter to solve such huge eigenproblems. In some cases, eigen solver mathematical properties and scalability go hand in hand. In many other cases, we must balance mathematical efficiency with scalability properties. Finding an optimum equilibrium between these two major metrics (numerical efficiency and scalability) of a large eigen systems is an endless topic.

This debate is all the more true in present context of the *"High Performance Computing (HPC)"*. Fast evolution of supercomputers during the last decades is now shaken by the up-coming exascale supercomputers. Most of the scientific fields related to HPC (from hardware architecture to the level of application design) are constrained to re-think acquired knowledge and technologies. This upheaval may seem intuitive or natural for some fields, such as programming paradigms or hardware architecture... It is less obvious to apprehend and understand that exascale revolution also deeply impacts other scientific fields such as algorithm design and applied mathematics.

This is the backdrop to this thesis, we focus on improvement of Krylov eigensolvers for large non-Hermitian matrices. The leading parameter will remain optimum of numerical efficiency (in this context, convergence of the Krylov eigen solver) and parallel efficiency. This thesis covers the Explicitly Restarted Arnoldi Method improvement by studying its convergence scheme. Such preliminary work is essential to implement an auto-adaptive (also called Smart-Tuning) ERAM algorithm. In this context, we particularly focus on the ERAM restarting strategies, as they may considerably accelerate ERAM convergence without any disruption regarding parallel performances (in terms of memory storage, global/blocking communications and number of operations). This study is completed by two matrix generators implementation to verify on one hand exactness of computed eigenpairs and on the other hand whose dimensions can be adapted to ultra-scale supercomputers.

We will first introduce the current context of High Performance Computing to highlight locks and challenges raised by the exascale target. The next chapter will present some Krylov eigensolvers, focus on their differences and emphasize *"tricky"* parameters that athirst parallel performances. The third chapter will present a method to evaluate and characterize Krylov eigensolvers convergence: such heuristic forms the basis for upcoming Auto/Smart-Tuning tools to improve ERAM performances. We will then present matrix generators and their huge potential for many scientific fields that (will) use ultra-scale supercomputers. We will emphasize their numerical efficiency and use generated matrices during the next chapters. The chapter 5 will present ERAM restarting strategies behavior and their impact on ERAM convergence. The last chapter will suggest some ways forward by outlining avenues for further reflection to dynamically tune ERAM restarting strategies with respect to its convergence.

Chapter 2
The Extreme-Scale Challenge

This chapter displays an overview of the *High Performance Computing (HPC)*, since it's inception to today. The first part focuses on the fast evolution of HPC to reach the petaflop scale. We emphasize that HPC is thus crucial for the understanding of our environment (in the broadest sense of the term) and why its evolution is so important to improve our current scientific knowledge. Secondly, we will detail locks that prevent us to reach the exaflop scale and the tracks of reflexion to over come exascale.

2.1 Supercomputing Rise until the Petaflop scale

In this section, we are going over the history of HPC, from its beginning until now. This introduction presents briefly HPC evolution and its incredibly fast development. Most of all, we highlight the revolution that is currently shaking up the entire HPC community.

Since the first supercomputers, many scientific achievements have been successful thanks to simulations. When we mention simulations, we generally think to computational science of *"non-feasible"* experiences, such as universe visualization [Springel 2011] as an illustration.

Simulation term thankfully encompasses much more experiences, both *"feasible"* and *"non-feasible"* ones. Regardless simulations, they are all issued from an advanced computational science study and a powerful supercomputer.

From a "global and simplified" point of view, scientific knowledge is gained through simulations and simulations are gained through supercomputers. Drawing on this, HPC science gathers fundamental sciences (such as physics, applied mathematics ...) with computer science (from the hardware until the programming languages). Compared to fundamental sciences, HPC is relatively recent but its evolution is incredibly fast. As knowledge is probably the most powerful weapon of the world, supercomputers are the most qualified soldiers to create and handle it.

A straightforward representation of HPC is that the more computing power you get, the more knowledge you are susceptible to get. We intentionally add "susceptible" as taking advantage of supercomputers computing power recquires first a "taming" phasis of the supercomputer itself.

Supercomputer's performance is generally the reduction, through the expanding notion of computing power to the floating point operations capacity ([Daly 2011], page 7). The higher the number of floating point operations per second (Flops) is, the more powerful supercomputer is, therefore the Flops measure is one of the reference metrics to appreciate

supercomputers performance. Still, reducing supercomputers performance only to this single metric remind a simplistic scheme ([Booth 2011], page 10).

2.1.1 Supercomputers History, Past & Present

HPC birth date is a controversial topic. Should we consider that HPC is born with the emergence of parallel computing or supercomputer concept? We will certainly not be taking part in this open debate.

If *"Super Computing"* concept has been first introduced in the early 1930s [New 1929], supercomputers as we consider them nowadays emerged in the 1950s.

Among many others, we can mention as computational science pioneers John von Neumann and Grace Hopper (We often refer to Grace Hopper as *"the woman who gave voice to computers"*). Many innovations were made possible thanks to their huge contributions to computational science. The computing power race which is indeed a global race, started out from this period and still goes on.

Since the first supercomputers, physicists, biologists and many other scientists have expressed their needs for more computational power. The first Moore's law ([Moore 1965],[Moo 2005]) suggested a strong and extended growth of power computing through chips computing capacities improvement. Supported by many scientific industries and fields, computer science community has made important innovations, in terms of hardware, software and programming languages/libraries.

In light of this, HPC leads to a conducive mutual enrichment of both computer and fundamental sciences. The megaflop scale was reached in the 1970s and no more than ten years later, the gigaflop scale was crossed. At that point, hardware components technology was still evolving and computing power was rapidly acquired by multiplying the number of components on a single chip.

We finally reached the teraflop scale [Mattson 1997] in the 1990s thanks to supercomputers with thousands of processors. The thirst for knowledge has driven us to petaflop supercomputers [Roa 2008] in only 5 decades since supercomputers appeared.

Based on existing technologies, supercomputers enhanced their power capacity by using more and more processors on a single chip, therefore by confirming the Moore's law. However, getting more parallelism level(s) inside of a node itself arises new issues, especially about the supercomputers operating systems, but also about algorithms designs and programming paradigms: How can you get efficiency from all of these parallelism levels? Over and above these problems, new components appeared to enhance power computing, such as accelerators (Graphic Process Units (GPU) and Intel® accelerators as an illustration).

Heterogeneous architectures (this point will be detailed in paragraph 2.3) have upset classic programming paradigms. If the theoretical peak performance offered by such supercomputers is definitely interesting, taking advantage of it requires many efforts at an expensive price (in both financial and working terms). Industries simulations frameworks are mainly long term development frameworks. With such architectures, a new

constraint arose, which is to optimize the fragile balance of efficiency versus re-usability, maintainability and portability of frameworks.

Petascale supercomputers gave access to an unprecedented power computing through many parallelism levels (dozens of thousand nodes, each having potentially many processors, each having many cores ...). Based on this observation, one computing node has itself at least 3 parallelism degrees and nodes may have different components, potentially leading to more intra-node parallelism. The increasing number of cores per node also implies a high memory hierarchy level, introducing new memory constraints.

All these constraints (among many others that will not be detailed in this thesis) are part of challenges to over-come to build a petascale simulation. If petascale constitutes itself a challenge, the exascale one is by far the highest in the whole HPC history.

As soon as the first petaflop supercomputer was built, HPC scientific communities were already dreaming of future exaflop supercomputers. Exascale has enhanced many other locks that we must override, so that the exascale dream may become true ... Somewhere in the (near?) future: As an illustration, the first exascale supercomputer is expected by the year 2018/2020 according to the expert advices of M. Snir, W. Gropp and P. Kogge in ([Snir 2011], page 2).

Latest changes introduced by petascale computing point to a better understanding of major research areas, such as nuclear science ([Ashby 2010], pages 20-24), climate ([Ashby 2010], pages 16-17), astrophysics ([Ashby 2010], pages 14-16), aerospace ([Ashby 2010], pages 11-14), combustion ([Ashby 2010], pages 17-20) and many others.

Current exascale research has been urged by many scientific communities thirsting for larger and more accurate simulations that will considerably extend our current knowledge. Every fields we mentioned above definitely need exascale supercomputers to get to a better understanding/visualization of current simulations/models ([Ashby 2010], pages 25-46).

Thanks to the computational power growth from the 1960^{th} until today, the scientific progress were realizable making sure that previous simulations implementations were improved with respect to new supercomputers characteristics. Far more effort is needed to extend current petascale simulations into exascale simulations ([Daly 2011], page 11). In fact, the term "extension" is not appropriate anymore, we should rather refer to a "metamorphose". Multiple exascale hurdles have been identified, as well as a small part of the tools to achieve it.

2.2 The Exascale Energy Wall

As a primary observation, exascale computing is unobtainable by using the past and present strategies to increase supercomputers power ([Dongarra 2014], pages 7-8). Supercomputers achieved more performances due to parallelism and clock rate improvements. However, due to technical constraints, the CPUs clock rate stopped its speed rate and supercomputers builders must now deal with it ([Brown 2010], page v). Both energy and technological barriers are major exascale constraints, we can definitely say that exascale marks a strong break with HPC knowledge acquired so far. To highlight the energy wall

for exascale, we illustrate this purpose on the current 5 most powerful supercomputers from Top500's list (June 2014) [Meuer]:

Top500's Rank	Supercomputer	Sustainable Peak PetaFlops/second	Theoretical Peak PetaFlops/second	Power Mega Watt
1	Tianhe-2	33,8627	54,9024	17,808
2	Titan	17,5900	27,1125	8,209
3	Sequoia	17,1732	20,1327	7,890
4	K-Computer	10,5100	11,2804	12,660
5	Mira	8,5866	10,0663	3,945

Table 2.1: The Top500 List is actualized twice a year. Starting in 1993, it provides the list of the 500 most powerful supercomputers in terms of Flops performance.

An extremely simplistic vision but a good introduction to exascale energy challenge is to multiply by 10^2 the energy consumption of each most powerful supercomputer ([Decker 2010], page 9) listed in the Table 2.1.

Getting Tianhe-2 to an exaflop system would consume 1700,808 Mega Watt. The highest limit of supercomputers energy consumption has been evaluated around $20-30$ Mega Watts with the expert advice of the United State Department of Energy ([Brown 2010], page 9; [Ashby 2010], page 49; [Decker 2010], page 9; [Beckman 2012], page 7/21).

A relatively recent (compared to the Top500 "birth date") trend is to classify supercomputers with respect to their Performance per Watt [?]. Based on the Table 2.2, the most efficient supercomputer (id est that has the best ratio of Flops versus consumed energy) list does not corresponds to the Top500 one (for both sustainable and theoretical peak performances).

PFlops/MW Rank	Top500 Rank	Supercomputer	Sustainable Peak Perf. per Watt $(PFlops).(s)^{-1}.(MW)^{-1}$	Theoretical Peak Perf. per Watt $(PFlops).(s)^{-1}.(MW)^{-1}$
3	1	Tianhe-2	1,9	3,08
2	2	Titan	2,14	3,3
1	3	Sequoia	2,18	2,55
4	4	K-Computer	0,83018	0,89103
1	5	Mira	2,18	2,55

Table 2.2: Starting from the Table 2.1, we computed the peak performance per Mega Watt for each most powerful supercomputer according to the Top500 List (June 2014).

The current supercomputer (from Green500 June 2014 List [Gre]) with the best Performance per Watt ratio is TSUBAME-KFC from the GSIC Center, Tokyo Institute of Technology (Japan) with a $4,38982$ $Pflops/MW$.

We recall that the previous tables (2.2 and 2.1) that presented supercomputers energy consumption do not take into account neither cooling system consumption nor module inverter, nor building consumption ... We estimate that a complete computing center energy consumption represents about 30% to 40% of supercomputer consumption itself.

During the last decade, the Flops measure was definitely the performance reference metric, while nowadays, flops performance per watt is a more pertinent metric, leading to a "low-consumption flops" performance ([Brown 2010], page v). According to the Table 2.1, an exaflop supercomputer using the current architecture models would consume many giga Watts: Such energy consumption scale is not financially viable.

Starting out this naive representation of the energy wall, one may glimpse the reason why we are all talking about HPC metamorphosis rather than an extension of our current knowledge. The power challenge is definitely the main piece of exascale achievements ([Brown 2010], page v; [Dongarra 2014], page 7-8), outcoming on dramatic transformations from fundamental mathematics until the highest programming level.

2.2.1 The Energy Consumption Metric as Reference

It is recognized that the greediest part in terms of energy consumption of parallel applications is data movement ([Booth 2011], page 10; [Daly 2011], page 13). The previous supercomputers architectures lead to bulk-synchronous applications and energy consumption has never been a priority.

In this context, overcome the energy wall definitely implies to favor the asynchronous communications and avoid (to the extend possible) global and blocking communications.

If OS ability to measure energy consumption may appear as an evidence, languages and numerical scientific libraries must also include such tools, so that simulation execution parameters may be adapted with respect to energy consumption at runtime.

This implies that algorithm design themselves must take into account energy consumption metric. Basically, we expect to get algorithms able to dynamically tune some numerical/simulation/hardware parameters to reduce its energy consumption, while conserving the method consistency.

2.3 The Extrem-Scale Supercomputers Architectures

With petaflop supercomputers arise heterogeneous architectures. The use of accelerators with CPUs or different CPU components is widespread in supercomputers architecture. As an illustration, one of the supercomputer we used during this thesis has three different partitions.

We emphasize that heterogeneous architecture is neither *the* unique trend nor *the* best one, as the $K-Computer$ achieves very good performances using a homogeneous

architecture (among many other supercomputers with homogeneous architectures).

The use of heterogeneous architectures aim to extract the most computing power as possible as they maximize the Flops per Watt ratio. The two following figures issued from [Rupp 2013] present peak floating point operations per watt with single and double precision respectively. These figures are very interesting as they compare the recent accelerators used in current supercomputers.

From 2007 until 2013 the GPUs were providing a significantly better peak floating point operations per watt ratio than the classic CPUs. This explains the democratization of heterogeneous supercomputers architectures based on CPU-GPUs during the last decade.

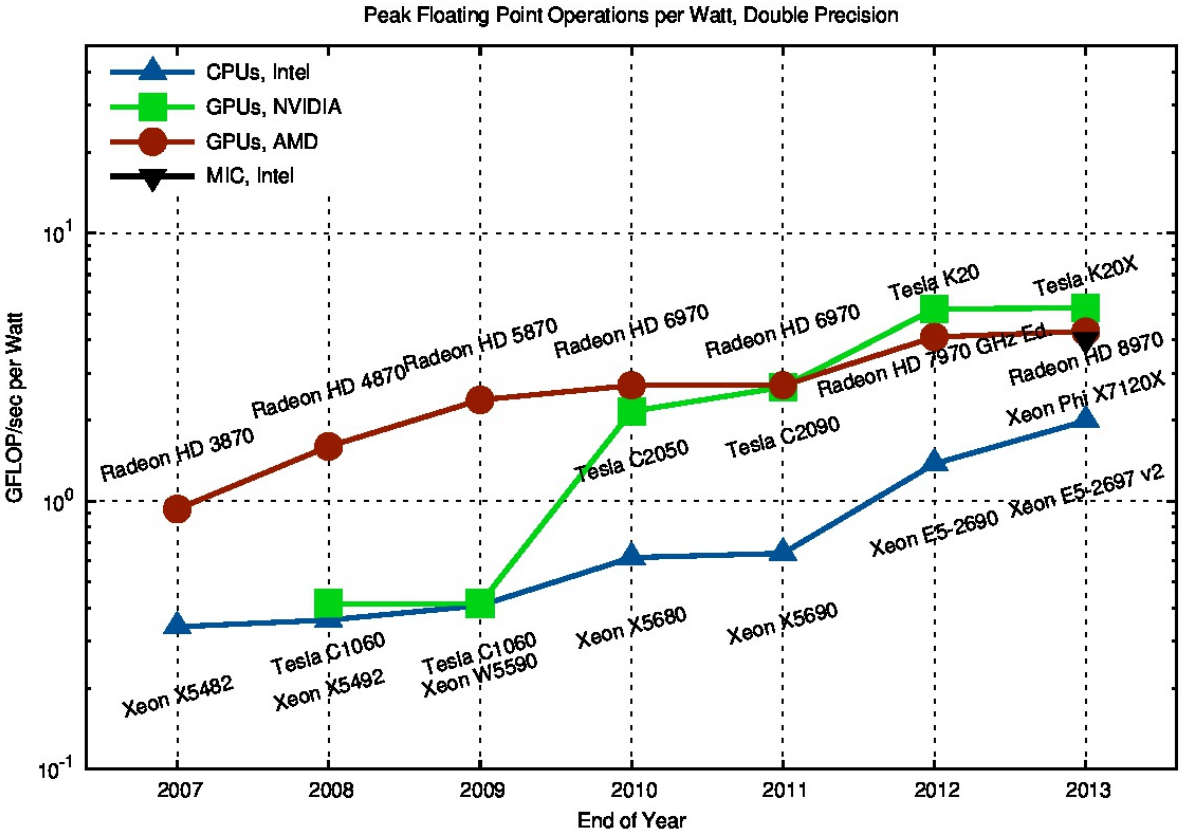

Figure 2.1: Peak Floatting Point Operations per Watt, Double Precision

Intel® CPU, GPU (NVIDIA® and AMD®) and Intel® MIC GFLOP/sec per Watt Comparison in the case of double precision arithmetics. The higher, the better [Rupp 2013].

If accelerators may be a good compromise to increase the peak floating point operations performances, one may notice that performance for a heterogeneous supercomputer is increasing non-uniformly compared to a homogeneous architecture ([Daly 2011], page 15): therefore, this *"performance gain"* is not *"free"* and many efforts are required to

take advantage of heterogeneous architectures.

As each hardware has its own specificities (parallelism levels, peak performance et caetera), it may also have languages and programming specificities. As an illustration, use of a *NVIDIA*® GPU for HPC computation requires to dispose of an implementation of the simulation framework with *CUDA*® language. On the other side, use of an *AMD*® GPU for the same purpose is not compatible with *CUDA*® language, neither we can use *CUDA*® to implement a framework destinate to a pure CPUs architecture.

The scientists can not afford the luxe to have a framework design specific to each architecture. However, developers and algorithm designers must work off their energy to benefit from each hardware power computation as much as possible and as long as the framework re-implementation costs do not overcome a certain amount. In this context, programming languages that abstract hardware language dependency arise. As an non-exhaustive list, we can mentioned the Open Computing Language (OpenCL) and OpenACC regarding the hardware specific languages.

Since 2011, the gap between the Intel CPUs and the accelerators is reduced (Figure 2.1), leading to the fact that supercomputers architectures may be chosen depending on the applications maintainability, portability and developments requirements with respect to hybrid architecture rather than on peak performance only.

2.3.1 Toward Hardware-Aware Algorithms

Heterogeneous architectures have neither a uniform performance metric nor uniformly parallelism levels. A partition may have some parallelism degrees while another one will not have the same. Therefore algorithms must consider this important constraint to exploit every parallelism levels of every partitions otherwise performance may be very poor compared to sustainable peak performance.

In this context, some simulations algorithms may be more adapted to accelerators or pure CPU architectures. Such constraints must intervene at algorithm design level, but this parameter is limited by the fact that supercomputers architectures evolve much more faster than simulations framework(s).

2.4 The New Programming Paradigms to Overcome the Exascale Challenge

Petascale supercomputers have already emphasized this additional complexity, as taking advantage of highly parallel multiple cores architectures has an expensive price. Methods themselves must be able to exploit more parallelism levels, from the lowest grain parallelism until the highest one ([Booth 2011], pages 23-24; [Brown 2010], page 23).

New parallelism levels may not be exploited by algorithm and/or previous programming paradigms.

Process units have drastically increased to provide more computation power. In fact, the growth of parallelism within chips had so far an exponential evolution. From the

computational power point of view, we can consider this additional parallelism levels as a benefit, but we can also qualify it as a complexity level.

2.4.1 Manage the Hardware Dependencies

Taking advantage of the lowest parallelism level may require a framework implementation very specific to the hardware used, getting to the efficiency versus portability/maintainability dilemma ([Beckman 2012], page 10).

The lowest grain parallelism definition may depend on both the algorithm and hardware point of view.

Exploiting multi cores parallelism and getting the best efficiency of it requires a multi-threading programming taking into account the hardware specificities. Develop current simulations framework to exploit hardware specificities implies a considerable reduction of framework portability and may engender deep modifications of frameworks development. One difficulty is to adapt these frameworks on future exascale machines while using them on the current supercomputers. In this context, new programming paradigms appear, using low intrusive language (such as pragma directive languages for example): as an illustration, we cite StarPU, Cilk or TBB. Each of these libraries/languages has its own benefits and defaults compared to each others. As a common point, they all aim to maintain portability, maintainability of framework while improving hardware parallelism efficiency ([Beckman 2012], page 12).

2.4.2 Algorithms Adapted to Multi-Level Parallelism

Multiple parallelism levels must be integrated directly at the algorithm design. Petaflop supercomputers have many parallelism levels and it remains arduous to exploit them from the highest one (inter-node) until the lowest one (processor). Such parallelism levels are defined by the memory hierarchy, that becomes more and more complex due to components and hybrid architecture. It is admitted that data movement is one of the greediest part in terms of energy consumption of parallel applications. Reducing data movement must not be limited to inter-node communications but also to intra-node parallelism.

In this context, new tools appeared to efficiently exploit each parallelism levels and conserve at the same time a certain maintainability and portability of application framework.

As an illustration, the finest parallelism levels will be sensitive to efficient management of resources inside a node: this implies to use libraries dedicated to shared-memory parallelism architecture and efficiently manage memory hierarchy inside a node, such as considering the multi-caches and their specificities so as to limit data-movement inside a node.

On the other side, coarse grain parallelism levels must be efficiently managed. As an illustration, the coarse grain parallelism may represent several collaborating methods or

parts of a method itself. According to graphic representations, the coarse grained parallelism may be modeled by tasks scheduler programming paradigms. As an illustration we can mention *YvetteML (YML)* ([yml]), a graph description language that separates communications and computation parts. Tasks communication are modeled using a graphic, starting from this the scheduler optimizes tasks communications. Graph descriptions are adapted tools to describe coarse (but not only) parallelism level while maintaining framework re-usability.

If we almost overcame some of these issues for petascale supercomputers, they arise again as exascale supercomputers are expected to have at least two to three additional orders of parallelism levels compared to petascale computers. Getting more scalability will require again expensive efforts from likely every HPC scientific community.

2.5 The Resiliency as a Key Parameter for the Exascale Computing

On previous supercomputers, resiliency was more considered under hardware and operating system ([Beckman 2012], page 7/20) fields. Basically, only hardware builder and supercomputer software designers worried about the resiliency challenge. It is assumed that the more the number of components increase, the more the *Mean Time Before Failure (MTBF)* decreases. Regarding the costs of a simulation, this is an absolute necessity. The MTBF is obviously expected to decrease with the exascale computing.

2.5.1 Ensure and Manage the Operating Systems Resiliency

OS designers and developers are first concerned by the resiliency challenge. Firstly, OS will have more trouble to detect faults regarding hardware complexity architecture and definitely more difficulties to communicate failure notifications across such large system. As OS are expected to be able to recover in a very short time, this adds an other level of complexity.

2.5.2 Languages & Numerical Libraries to Support the Resiliency

Many scientific frameworks are not using from now resilient parallel communication libraries. If having a perform OS regarding the resiliency property is not sufficient, we must add it to the languages and scientific numerical libraries to ensure the framework simulation execution even in the case of hardware failure. As an example, a massive effort from MPI library is provided to integrate the resiliency property [MPI] so that the entire communicator can survive to task subsets failure.

2.5.3 Toward Resilient Algorithms

With today's supercomputers having dozens of thousands cores, the resiliency begins to interest designers algorithms and developers. Many tools (algorithmic and mathematical ([Daly 2011], page 12; [Dongarra 2014], pages 8-9)) arise to recover data in case of hardware/software failure. There is a urge need to conceive resilient algorithms that must be able to recover eventual lost data. In the context of exascale applications, it is an absolute necessity to maintain simulation execution and get the right results even though a part of the method has failed (due to a cores and/or task failure). Resiliency property integrated in algorithms is not anymore an *"extra"* but a *"sine qua none"*.

2.6 The Mathematics Challenge for the Exascale Wall

Each exascale player presented above are all strongly related to each others and their improvements obviously depend from each other. If these correlations seem naturally related to HPC, applied mathematical research in accordance with HPC revolution is less evident.

In this section, we highlight mathematical needs for the exascale challenge. The complexity in the exascale challenge is that both computer science and algorithm design must evolve hand in hand. Algorithms must be developed according to supercomputer architecture and (future) supercomputer architectures must be compatible with previous/current/future algorithms...

To reduce algorithm energy consumption, we may limit data movements. To improve algorithm scalability, we may use asynchronous communications instead of synchronous ones ([Alexander 2011], page 14). To satisfy resiliency property, we may share each tasks results with each other (this enters into direct confrontation with communications constraints) or use data redundancy (this enters into direct confrontation with memory constraints) ([Daly 2011], page 12; [Dongarra 2014], pages 8-9/40-42). Many of possible ameliorations presented above may depends on varied parameters and a part of these parameters may themselves depend on the execution itself.

For each of these constraints, we disrupt the proper implementation of algorithms and potentially its mathematical properties. As an example, deleting arbitrarily parallel communications or turn them to asynchronous ones lead to an incoherent simulation in terms of mathematical results. Many research has been done to reduce drastically communications ([Alexander 2011], pages 6-7) or at least optimize them such as we optimize simulation exactitude versus energy consumption ratio. Programming paradigms appear such as tasks schedulers to optimize the communications as an example. We are moving towards concurrent methods models rather than complementary and dependent method models.

In this context, new algorithms, qualified as self-adaptive (or auto-tuned) appeared, driving to smart methods able to modify some of their parameters with respect to runtime execution ([Beckman 2012], pages 10/21/43-44; [Daly 2011], page 14; [Ang 2012], pages

13-14), to satisfy to the listed constraints above. Modified parameter(s) or metric(s) used to perform changes may evolve independently from each other depend on hardware constraints, data storage constraint, numerical constraints ...

Two cases are particularly relevant:

- the metric depends and/or relies on numerical properties and/or elements of the method,
- the modified parameter depends and/or relies on numerical properties and/or elements of the method.

In these two cases, auto-tuned algorithm aims to optimize the ratio of numerical accuracy versus parallel execution performance (in a broad sense) ([Ang 2012], pages 15-16). Bearing this in mind, whether numerical parameters is a metric or is changed, mathematical properties and validity must be ensured, while accuracy must be at least maintained. The introduction of uncertainties coming from programming and/or hardware parameters must be represented, quantified and integrated into mathematical models ([Brown 2010], page 43; [Alexander 2011], page 8-9; [Ang 2012], page 15-16; [Dongarra 2014], page 10; [Booth 2011], page 12-17).

As an illustration, parallel normalization of a vector using the Euclidean norm 2 should be completely avoided from all algorithms to satisfy to exascale constraints. Let's illustrate our purpose: We consider a vector of size n distributed onto np tasks. For simplicity, we consider that each task dispose of $\frac{n}{np}$ contiguous data. Each tasks executes its local scalar product, leading to the necessity to share local scalar product among np tasks, sum them and finally send them to all tasks. In the specific but very common case that the Euclidean norm 2 of a distributed vector is needed to pursue the computation, such operation may constitute a bottleneck. How can you replace it and conserve the mathematical properties and consistency of the results?

This is the case with the Gram-Schmidt orthogonalization process [Gram 1883], [Schmidt 1908], widely used in many numerical algorithms. However, such algorithm is definitely misfit in terms of parallelism: global and blocking communications, operations depending from each others. If we want to adapt the Gram-Schmidt orthogonalization process to exascale computing, we must delete all global and blocking communications, getting to an inconsistent system that does not satisfy at all to mathematical properties of algorithm.

New methods aim to delete a part of global and/or blocking communications ([Alexander 2011], pages 6-7) to reduce the exascale locks and minimize at the same time disturbances of model mathematical accuracy. Many methods exists and they are all based on the optimization of parallel scalability (and execution time) with respect to mathematical accuracy and consistency of the model.

If the metric or changed parameters depends on runtime execution and influence it ([Alexander 2011], pages 9; [Dongarra 2014], pages 42-43), how can you provide reproducibility? Even worst, how can you ensure, for every cases the model consistency?

HPC community need to go back to mathematical roots of methods, consider at this level nondeterministic aspects ([Alexander 2011], pages 8-9) and auto-tuning methods to ensure at every levels the model mathematical consistency.

It is ironic that the exascale computing aims to provide more precision of existing models while the future framework scalability depends on the nondeterministic and approximation introduction into existing methods. In the same way as we are missing many information regarding hardware, software, programming models, we are definitely missing mathematical background to reinforce algorithms and framework models.

In this thesis, an effort has been provided to model from a mathematical point of view presented methods and algorithms. We aim to mathematically model every elements, especially in the case of auto-adaptive method, regarding metrics used to change its parameters. By identifying them, we aim to provide some tools to re-design method using mathematical models.

2.6.1 The Eigenvalue Problems, Still Alive

The eigenvalue problem is a key point in many simulations: Many systems "simplification(s)" and/or "transformation(s)" drive to an eigenvalue problem. Then, the eigenvalue problem itself differs by the number of eigenpairs the system requires:

- whether you must compute the dominant (respectively the lowest) eigenpair,
- whether you must compute a subset of eigenpairs,
- whether you must compute the entire spectrum.

To each eigenvalue problem quoted above corresponds a class of eigensolvers.

Among all of methods that use the dominant eigenpair, we can cite page rank problem which is nothing more than finding the dominant eigenpair as fast as possible more information in [Kamvar 2004] and [Bianchini 2005]). The modeling of epidemic spread is a very similar problem to page rank one. In a different context, we can cite the Boltzmann neutron transport equation [Lewis 2004]. All these problems need to compute at a point the dominant eigenpair of a system.

For the eigenvalue problem in general, we can also mention seismic, climate modeling, oil driving, fluid structure interaction, vibration accoustic, et caetera. The eigenvalue problems are so current that quoting all concerned areas would constitute a thesis by itself.

As each simulation has its own requirement, eigenvalue problem has it too. Depending on matrix properties, the eigenvalue problem will be more or less complicated. Most of HPC community gets its test matrices from its own applications or from open-source matrices.

Using matrices issued from simulation(s) is definitely better as it reflects at best the system we aim to solve, but it may arise confidentiality problems. The reproducibility and (results) validation of simulations impose to dispose of a set of test matrices. As

an illustration, the CEA applications use matrices coming from nuclear reactor model. These matrices are fully confidential, it is out of question to let them being open-source matrices for tests and results validations.

Two widely used open-source sites propose many matrices coming from different problems, such as (un)directed graph, optimization problem, circuit simulation problems, computational fluid dynamics problems and many others ... Having such matrices is a huge benefit to tests and validate our applications, however, the matrices properties do not necessarily match with our problem requirements. As an example, the eigenvalue repartition is not compatible with our system matrices, the size may be too small ...

We aim to dispose of matrices generators, such that starting out a given spectrum, we generate a matrix that conserves this spectrum and has the required mathematical properties. We take off the confidentiality lock, matrices properties requirements and additionally have a matrix that matches perfectly with the problem to solve.

Conclusion

We presented in this chapter the current revolution that is shaking up HPC community. Supercomputers performance was enhanced by the increasing computation units, but this scheme is now stopped by the exascale energy wall.

The performance must be lead by the use of efficient algorithms with auto-adaptive tools in order to improve the parallelization and/or the numerical accuracy of the method.

As the eigenvalue problem is a key operation in nuclear physics, among many other scientific fields, there is a need to improve the eigen values solver(s) with respect to the scalability and numerical efficiency. This goes with the needs of matrix generators starting from imposed spectrums in order to validate the computed eigenvalues and get systems according to the ultra-scale supercomputers.

These thematics will be developed during the following chapters.

Chapter 3
Krylov Method for Eigenvalue Problems

Computing eigenvalues and/or eigenvectors of a large non-hermitian matrix is not easy and remains costly in terms of parallel computations costs.

In this context, Krylov methods are good candidates to compute a subset of eigenpairs of a large non hermitian matrix. Krylov eigensolvers builds an eigen-system whose dimension is negligible compared to the initial system and whose mathematical properties verify it approximates the original eigen-system. Krylov eigen solvers philosophy is based on the equilibrium of numerical accuracy of the approximated eigen system and computation costs to solve it.

In this chapter we present some of the commonly used Krylov eigen solvers. It exists of course many variants of the following presented methods. We will focus on restarting strategies for each eigen solver presented below and conclude on their parallel ability with respect to exascale requirements.

3.1 The Arnoldi Method

The Arnoldi Method [Arnoldi 1951] is the basis of the Restarted Arnoldi Method: this last one is an iterative method that aims to compute the spectrum of a large non-Hermitian matrix $A \in \mathbb{C}^{n\times n}$. The Arnoldi Method generates an incomplete Hessenberg decomposition of A [Saad 1980],[Saad 2011], so as to compute A eigenpairs.

Introduced in 1951, the Arnoldi Method is based on the Gram-Schmidt orthogonalization process [Gram 1883], [Schmidt 1908]. It exists several variants of the Gram-Schmidt orthogonalization process, but the Classic Gram-Schmidt (CGS) is the most common one among many other variants (such as Modified Gram-Schmidt, Classic Gram-Schmidt with Re-orthogonalization... More details can be found in [Dubois 2011a] pages 33).

Starting from a random unit norm vector $v \in \mathbb{C}^n$, the Arnoldi Method computes iteratively an orthogonal projection of A onto a Krylov subspace (introduced in 1931 by A. N. Krylov) that we denote by $\mathbb{K}_{m,v}$, where $\mathbb{K}_{m,v}$ is defined by:

$$\begin{cases} \mathbb{K}_{m,v} = Span\{v, Av, ..., A^{m-1}v\}, \\ dim(\mathbb{K}_{m,v}) = m, n >> m, \end{cases} \tag{3.1}$$

m is the size of the generated Krylov subspace $\mathbb{K}_{m,v}$ and is commonly called the Krylov subspace size. A Krylov subspace that verifies the equation 3.1 is the result of a m-step Arnoldi Method.

The Algorithm 1 presents the Arnoldi Method using a Classic Gram-Schmidt orthogonalization process.

We first introduce some notations used in the Algorithm 1 below:

- For a given matrix G with $n \in \mathbb{N}$ lines and $m \in \mathbb{N}$ columns, the vector g_i refers to the i^{th} column of the matrix G while $g_{i,j}$ the element of matrix G on i^{th} row and j^{th} column,
- the $G_{k,l}$ matrix refers to the sub-matrix of G containing the k first lines and the l first columns of G,
- during all the following chapters, unless we specify it explicitly, the norm $||.||$ refers to the Euclidean norm $||.||_2$,
- the operator ^{T} is the transpose operation and ^{H} is the Hermitian transpose operation,
- $\forall i \in \mathbb{N}^*$, $e_i \in \mathbb{C}^i$ is the i^{th} Cartesian basis vector of $\mathbb{C}^i$.

During all the following chapters, we will retain these notations.

Algorithm 1 The Arnoldi Method using CGS

Input: $A \in \mathbb{C}^{n\times n}$, $v_1 \in \mathbb{C}^n$, $\varepsilon > 0$, $m \in [1, n]_{\mathbb{N}}$
$H \in \mathbb{C}^{(m+1)\times m}$
$V \in \mathbb{C}^{n\times(m+1)}$

$v_1 = \frac{1}{||v_1||} v_1$
for $k = 1, m$ **do**
  $v_{k+1} = Av_k$
  $h_k = v_{k+1}^H V_{n,k}$
  $v_{k+1} = v_{k+1} - V_{n,k} h_k$
  $h_{k+1,k} = ||v_{k+1}||$
  if $\varepsilon \geq h_{k+1,k}$ **then**
    stop
  else
    $v_{k+1} = \frac{1}{h_{k+1,k}} v_{k+1}$
  end if
end for
Output: $H \in \mathbb{C}^{(m+1)\times m}$, $V \in \mathbb{C}^{n\times(m+1)}$

The Arnoldi Method generates two matrices:

- The matrix $H \in \mathbb{C}^{(m+1)\times m}$,

- The matrix $V \in \mathbb{C}^{n\times(m+1)}$.

In fact, the incomplete Hessenberg decomposition of A is H by deleting its last row (We denote it by $H_{m,m}$).

The resulting matrix V from the Arnoldi method has many properties. We denote by $V_{n,m}$ the matrix V by deleting its last column. At the end of a m-step Arnoldi Method, the vector v_{m+1} is orthogonal to $V_{n,m}$ but not normalized. Considering the Algorithm 1, each columns of $V_{n,m}$ are orthogonal and have a unit norm. $V_{n,m}$ is then an orthonormal matrix, which constitutes a basis of $\mathbb{K}_{m,v}$.

3.1.1 The Arnoldi Method Convergence

The Arnoldi Method stops if and only if we have reached the maximum number of iteration m or if $\varepsilon \geq h_{k+1,k}$ (Algorithm 1, step 6) [Saad 2011]. If $\varepsilon \geq h_{k+1,k}$ is true, this means that $\mathbb{K}_{m,v}$ has converged to an invariant subspace. In this specific case, the vector v_{m+1} is not computed during the mth step of the Arnoldi Method. Starting from this hypothesis, we obtain the equation:

$$A = V_{n,m} H_{m,m} V_{n,m}^T, \tag{3.2}$$

If we satisfy the equation 8.1, then $H_{m,m}$ eigenpairs are the exact A eigenpairs [Saad 2011].

3.1.2 The Arnoldi Method Accuracy Dependancies

The full Arnoldi Method using a CGS orthogonalization process necessitates :

- $\frac{m(m+1)}{2}$ axpys operations (step 5 of the Algorithm 1),
- m matrix/vector products (step 3 of the Algorithm 1),
- $\frac{m(m+1)}{2}$ dots operations (step 4 of the Algorithm 1),
- m norm 2 operations (step 6 of the Algorithm 1),
- and finally m scal operations (step 10 of the Algorithm 1).

In a parallel distributed memory environment, due to the large n value, the matrices A and V are distributed while the matrix H is replicated one each processor, as $H \in \mathbb{C}^{(m+1)\times m}$ dimension is negligible compared to the initial system matrix A ($n >> m$). We recall that V is a dense matrix, while A may be dense or sparse.

At each restart (that we denote by $k \in \mathbb{N}$) of the Arnoldi Method, we build a new vector $v_{k+1} \in \mathbb{C}^n, k \in [1, m]_{\mathbb{N}}$ and a new column of H matrix. Therefore, data size increases, adding n+m+1 elements at each restart.

Additionally to memory storage, each Arnoldi Method restart is costly in terms of number of parallel computations. Especially, operations such as normalization of v_{k+1}

and matrix vector products steps are expensive in terms of number of blocking and global communications. The more m increases, the more the Arnoldi Method is expensive in terms of computational resources.

The m value is fixed such that $n >> m$ in order to satisfy computational and memory costs constraints, however, fixing correctly m value is not that simple.

It is assumed that a large m value will produce a better $\mathbb{K}_{m,v}$ in terms of numerical accuracy [Châtelin 1988]. With a large m, the $H_{m,m}$ matrix is a better projection of A onto $\mathbb{K}_{m,v}$. However, we recall that obtaining this numerical accuracy is not free in terms of computational costs.

There is a tricky equilibrium between the Arnoldi Method numerical accuracy and its computational costs (both memory storage and parallel communications), most of it depends on m parameter. A large m value will favor $\mathbb{K}_{m,v}$ numerical accuracy but may considerably increase its execution time per iteration and lead to an energy-greedy algorithm. On the opposite way, a small m will provide poor approximation of the initial system but in a fast execution time per iteration due to the decrease of blocking and global prallel communications.

Considering these two factors independently from each other is a mistake: they are intrinsically related to each other and they must be optimized simultaneously in order to reach the optimum configuration between numerical accuracy and parallel execution time of the Arnoldi Method.

Additionally to parallel computation costs, m value has a direct impact on numerical accuracy of the projected system: A large m value favors a lack of orthogonality [Giraud 2005] and may engender an inaccurate Krylov basis. Indeed, the larger m is, the more we face to Floating Point rounding errors during $V_{n,m}$ computation. The Arnoldi Method can rely on other variants of Gram-Schmidt orthogonalization process, in order to improve $V_{n,m}$ orthogonality. If $V_{n,m}$ orthogonality is poor, then $H_{m,m}$ is consequently an instable projection of A onto $\mathbb{K}_{m,v}$ ([Braconnier 2000]).

It is possible to re-orthogonalize each vector of $V_{n,m}$ to ensure a numerical accuracy. If such method is numerically efficient [Dubois 2011d],[Dubois 2011a], but this algorithm is computationally very greedy. Such process is called the Classic Gram-Schmidt with Re-orthogonalization (CGSR) process. The Arnoldi Method using the CGSR process is presented in the Algorithm 2 bellow:

Algorithm 2 The Arnoldi Method using CGSR

Input: $A \in \mathbb{C}^{n\times n}$, $v_1 \in \mathbb{C}^n$, $\varepsilon > 0$, $m \in [1,n]_{\mathbb{N}}$

$v_1 = \frac{1}{||v_1||} v_1$

for $k = 1, m$ **do**
  $v_{k+1} = Av_k$
  $h_k = v_{k+1}^H V_{n,k}$
  $v_{k+1} = v_{k+1} - V_{n,k} h_k$
  $c_k = v_{k+1}^H V_{n,k}$
  $v_{k+1} = v_{k+1} - V_{n,k} c_k$
  $h_{k+1,i} = ||v_{k+1}||$
  if $\varepsilon \geq h_{k+1,k}$ **then**
    stop
  else
    $v_{k+1} = \frac{v_{k+1}}{h_{k+1,k}}$
  end if
end for
$H = H + C$

Output: $H \in \mathbb{C}^{(m+1)\times m}$, $V \in \mathbb{C}^{n\times(m+1)}$

It also exists some variants of the Arnoldi-CGSR algorithm. Several studies emphasize the possibility to reduce the re-orthogonalization process, by using a selective re-orthogonalization [Aquilanti 2011b], [Aquilanti 2011a]. The selective re-orthogonalization consists in decreasing the number of global communications while maintaining as much as possible $V_{n,m}$ orthogonality.

Finally, there exists a variant of the Arnoldi Method using a classic CGS or CGSR orthogonalization scheme, which is called the Arnoldi Method with incomplete orthogonalization. It consists to execute (whatever the orthogonalization scheme you choose) a partial Arnoldi Method, meaning that we execute only a subset of restarts of the m-step Arnoldi Method.

3.1.3 The Arnoldi Method with Incomplete Orthogonalization

We generally start from previously computed data resulting from a k-step Arnoldi Method, where $k \in [1,m]_{\mathbb{N}}$. This method has been introduced by Y. Saad [Saad 1980] (more information can be found in [Saad 2011]).

Let's consider that we performed a k-step Arnoldi Method, obtaining $H \in \mathbb{C}^{(k+1)\times k}$ and $V \in \mathbb{C}^{n\times(k+1)}$ matrices. One may use $H_{k+1,k}$ and $V_{n,k+1}$ matrices to extend them until $H_{m+1,m}$ and $V_{n,m+1}$ via a $(m-k)$-step Arnoldi Method. As $m \geq k$, it results that $\mathbb{K}_{k,v} \subset \mathbb{K}_{m,v}$. The Incomplete Arnoldi Method using CGS orthogonalization scheme is presented below:

Algorithm 3 The Incomplete Arnoldi Method using CGS

Input: $A \in \mathbb{C}^{n\times n}$, $v_k \in \mathbb{C}^n$, $\varepsilon > 0$, $m \in [1,n]_{\mathbb{N}}$, $k \in [1,m]_{\mathbb{N}}$

1: $v_k = \frac{1}{||v_k||} v_k$

2: **for** $i = k, m$ **do**
3: $v_{i+1} = Av_i$
4: $h_i = v_{i+1}^H V_{n,i}$
5: $v_{i+1} = v_{i+1} - V_{n,i} h_i$
6: $h_{i+1,i} = ||v_{i+1}||$
7: **if** $\varepsilon \geq h_{i+1,i}$ **then**
8: stop
9: **else**
10: $v_{i+1} = \frac{1}{h_{i+1,i}} v_{i+1}$
11: **end if**
12: **end for**

Output: $H \in \mathbb{C}^{(m+1)\times m}$, $V \in \mathbb{C}^{n\times(m+1)}$

We recall that the Incomplete Arnoldi Method can be executed with every orthogonalization schemes presented above (CGS, CGSR or MGS ...).

The Arnoldi Method presented in this section is not self-sufficient to provide A approximated eigenpairs, unless $\mathbb{K}_{m,v}$ subspace has reached an invariant subspace. In most of the cases, we must iteratively builds several $\mathbb{K}_{m,v}$ using more and more accurate eigen-information to favor $\mathbb{K}_{m,v}$ to the desired eigen-subspace.

The Restarted Arnoldi Method (based on the Arnoldi Method) uses $H_{m,m}$ and $V_{n,m}$ properties to extract pertinent eigen-information and compute from it a new Krylov subspace. By restarting such process, we aim to obtain a satisfiable convergence of $\mathbb{K}_{m,v}$ to an invariant subspace.

3.2 The Restarted Arnoldi Method

In this section, we present the Restarted Arnoldi Method, how we reuse computed data of the Arnoldi Method to compute a more accurate Krylov subspace.

Starting from the previous section, we already mentioned that $H_{m+1,m}$ is a unitary projection of A onto $\mathbb{K}_{m,v}$. From the Arnoldi Method, we have the following equation [Saad 2011]:

$$AV_{n,m} = V_{n,m}H_{m,m} + h_{m+1,m}v_{m+1}e_m^H, \tag{3.3}$$

In scientific literature, we denote by $f_m = h_{m+1,m}v_{m+1}$. In what follows, we will retain this notation.

In the section 3.1, we mentioned that the Arnoldi Method is used to compute the spectrum of matrix A. In fact, we must moderate this point. The Arnoldi Method tends

to converge towards the outer eigenvalues of A spectrum [Saad 2011], [Châtelin 1988].

As $H_{m,m}$ is an Hessenberg superior matrix, computing its eigenpairs is relatively easy. Additionally to this important property, $n >> m$ which justifies that computing $H_{m,m}$ eigenpairs is not costly in terms of computation time but still, we need to moderate this point as it depends on m value.

We compute a Schur decomposition from $H_{m,m}$ to extract its eigenvalues:

$$H_{m,m} = S_m^T T_m S_m, \tag{3.4}$$

The matrix S_m is a unitary matrix and T_m is triangular. $H_{m,m}$ eigenvalues are contained on T_m main diagonal: We will denote by $\Theta_m \in \mathbb{C}^m$ the spectrum of $H_{m,m}$ and by $Y_m \in \mathbb{C}^{m \times m}$ the corresponding eigenvectors.

Multiplying the equation 3.3 by Y_m we obtain the following equation 3.5:

$$\begin{cases} AV_{n,m}Y_m &= V_{n,m}H_{m,m}Y_m + f_m e_m^H Y_m \\ AV_{n,m}Y_m &= \Theta_m V_{n,m}Y_m + f_m e_m^H Y_m \end{cases} \tag{3.5}$$

We denote by $U_{n,m} = V_{n,m}Y_m$, $\beta_m = e_m^* Y_m$ and replace them in the equation 3.5:

$$\begin{cases} AU_{n,m} &= U_{n,m}\Theta_m + f_m\beta_m \\ AU_{n,m} - U_{n,m}\Theta_m &= f_m\beta_m \end{cases} \tag{3.6}$$

Basically, the equation 3.6 means that $\{U_m, \Theta_m\}$ provides an approximation at $f_m\beta_m$ precision of A eigenpairs.

$$\begin{cases} \forall\, \mathrm{j} \in [1,m]_{\mathbb{N}} & \\ Au_j - u_j\theta_j &= f_m e_m^H y_j \\ ||Au_j - u_j\theta_j|| &= |h_{m+1,m}y_{m,j}|, \end{cases} \tag{3.7}$$

The equation 3.7 is available for $\forall\, \mathrm{j} \in [1,m]_{\mathbb{N}}$ in theory. In practice, only a subset of $\{U_m, \Theta_m\}$ verifies these observations with a satisfiable residual. Additionally, this method is not adapted to compute all eigenpairs of A: it would require to fix $m = n$, which is no possible because of computational reasons that we presented in section 3.1 (memory storage and parallel communications costs).

Θ_m eigenvalues may not be ordered during their computation step. We denote by $\lambda_j \in$ the j^{th} dominant (in terms or real or imaginary modulus) eigenvalue of A (we will remain this notation during this thesis). After ordering $H_{m,m}$ eigenvalues, one may observe that θ_j does not necessarily corresponds to λ_j, as an illustration, θ_j may converge to λ_{j+1} or λ_{j-1}. Such phenomenas arise during first restarts of the Arnoldi Method, when $\mathbb{K}_{m,v}$ is not accurate enough. Nevertheless, such behavior also depends on A mathematical properties and its eigenvalues distribution: conjugate eigenpairs are a classic example of such problems.

Starting from the equation 3.6, there are two estimators to estimate the accuracy of approximated eigenpairs.

We will denote these two indexes by :

- $resTr_j = ||Au_j - u_j\theta_j||$ where θ_j is the j^{th} dominant (in terms of real or imaginary modulus) approximated eigenvalue of A (also called Ritz eigenvalue) and u_j its associated approximated eigenvector (also called Ritz eigenvector),
- $resTh_j = |h_{m+1,m}y_{m,j}|$ where $h_{m+1,m}$ is the last element of H matrix and $y_{m,j}$ is the m^{th} (last) element of the j^{th} dominant eigenvector of $H_{m,m}$ (H by deleting its last row).

During all the following chapters, we will retain these notations. Theoretically speaking, the equation 3.7 proves that $resTr_j = resTh_j, \forall j \in [1, m]_{\mathbb{N}}$.

The theoretical residual $resTh_j$ for each eigenpair is a simple multiplication of $h_{m+1,m}$ with the last element of each $H_{m,m}$ eigenvectors. First of all, this is a very simple operation that implies neither temporary memory storage nor parallel communications.

The other residual $resTr_j$ implies for each desired eigenpairs a matrix/vector product, a scalar operation and a dot operation. For each desired eigenpairs, computing this residual requires global and blocking parallel communications, which is quite costly:

- m matrix (sparse or dense)/vector (dense) products,
- m linear operation (neither blocking nor global communications for this point),
- m dot operations.

Moreover, these parallel operations induce the propagation of many roundings. Because of the computational differences, it results that the equation 3.7 is not true computationally speaking.

In the widely used parallel numerical libraries offering the Restarted Arnoldi Method (or some variants that we will detail below), both residuals can be computed. However, $resTr_j$ is considered as the reference residual as the $resTh_j$ numerical value is not as accurate as $resTr_j$. Numerical accuracy is favored compared to computational costs.

The Algorithm of the Restarted Arnoldi Method is an extension of the previously presented Algorithm 1. In what follows, we will denote by s the number of desired eigenpairs, fixed by the user. It is obvious that $m \geq s$, still this point will be detailed later. We will remain this notation throughout all this thesis.

Algorithm 4 The Restarted m-step Arnoldi Method

Input: $A \in \mathbb{C}^{n\times n}$, $v_1 \in \mathbb{C}^n$, $\varepsilon_{Arnoldi} > 0$, $\varepsilon > 0$, $m \in [1,n]_{\mathbb{N}}$, $s \in [1,m]_{\mathbb{N}}$

1: $v_1 = \frac{1}{||v_1||} v_1$

2: Execute m-step Arnoldi Method using $\{A,\ v_1,\ \varepsilon_{Arnoldi},\ m\}$
3: Solve the eigen problem $H_m Y_m = \Theta_m Y_m$
4: $\forall j \in [1,m]_{\mathbb{N}}, resTh_j = |h_{m+1,m} y_{m,j}|$
5: $U_{s,m} = V_{n,m} Y_{m,s}$
6: $\forall j \in [1,s]_{\mathbb{N}}, resTr_j = ||Au_j - u_j\theta_j||$
7: **if** $\varepsilon \geq \max_{j=1,s}\{resTr_j\}$ **then**
8: stop.
9: **else**
10: $v_1 = u_1$, go to step 1
11: **end if**

Output: $U_{m,s} \in \mathbb{C}^{n\times s}, \Theta_{s,1} \in \mathbb{C}^s, ResTr_{s,1} \in (\mathbb{R}^+)^s$

Depending on the accuracy we desire for desired eigenpairs, s Ritz eigenpairs (denoted by $\Theta_{s,1}$ and $U_{m,s}$ issued from the Restarted Arnoldi Method) may be a good approximation of A eigenpairs. In the other case, we must restart the Arnoldi Method in order to minimize $resTr_j$ value $\forall j \in [1,m]_{\mathbb{N}}$.

There exists several variants to restart the Restarted Arnoldi Method.

Each restarting method reuses previously computed data such as Ritz eigenpairs. However, each method uses eigen-information differently, implying different characteristics in terms of numerical convergence and parallel computation properties.

The Krylov Eigensolvers presented below are all derivated from the Restarted Arnoldi Method, each of them using different methods to restart.

In a large sense, Krylov Eigensolvers aim to solve the initial eigenvalue problem onto the projected system, because of its Hessenberg form and its small size. Once we get eigenpairs of the projected system, we project them onto the Krylov subspace and obtain finally Ritz eigenpairs that approximate A eigenpairs.

We restart the process until Ritz eigenpairs provide a satisfiable approximation of A eigenpairs.

3.3 The Explicitly Restarted Arnoldi Method

The Explicitly Restarted Arnoldi Method (ERAM) [Saad 1980],[Saad 2011] uses the Ritz eigenvectors to restart the Arnoldi Method. ERAM is the most straightforward version of the Krylov eigen solvers.

To restart an ERAM, we must compute a new initial guess v_1 that will be introduced in the Restarted Arnoldi Method in order to generate more accurate eigenpairs than the previous restart. In order to distinct each ERAM restart, we introduce a new index on

each presented data. As an illustration, $v_1^{(i)}$ is the initial guess used for the i^{th} ERAM restart. If $v_1^{(1)}$ is arbitrarily chosen, the next initial guess must be computed carefully.

$v_1^{(i+1)}$ is generally chosen such that it takes into account previously computed eigenvectors $U_{n,s}$. In this context, the new Krylov subspace $\mathbb{K}_{m,v_1}^{(i+1)}$ convergence to desired eigenpairs will be favored. We denote by γ the number of Ritz eigenvector we take into account to compute the restarting vector: it is obvious that $m \geq \gamma \geq s$. In what follows, we will keep this notation.

The most classic formula to compute the new initial guess $v_1^{(i+1)}$ is presented in equation 3.8:

$$v_1^{(i+1)} = \sum_{j=1}^{\gamma} \Re(u_j^{(i)}), m \geq \gamma \geq s \geq 1 \tag{3.8}$$

Where $\Re(u_j^{(i)})$ denotes the real part of the j^{th} dominant Ritz eigenvector computed at the i^{th} restart.

We uniformly weight each approximated eigenvectors to favor the convergence of ${\mathbb{K}_{m,v_1}}^{(i+1)}$ to s desired eigenvectors.

We use only $\Re(u_j^{(i)})$ of approximated eigenvectors, so that we avoid all complex arithmetic difficulties. This choice is consistent in the case of conjugated eigenpairs, as $\Re(u_j^{(i)})$ remains the same for $u_j^{(i)}$ and $\overline{u_j{}^{(i)}}$.

The ERAM convergence may be affected especially in the case of conjugated eigenvalues. In these cases, one may observe that approximated eigenpairs have a satisfiable accuracy, while eigenvectors do not correspond at all to A eigenvectors. Such situations are tricky to detect, as only the residual is used to track eigenpairs convergence. In these cases that have been identified and studied in [Chen 2005], [Jia 2004b], it appears that the Ritz eigenvalue converges while its associated eigenvector is wrong. Therefore, restart the ERAM with $v_1^{(i+1)}$ that takes into account such vectors may considerably disrupt the ERAM convergence.

Including the equation 3.8 into the Algorithm 4, we obtain the ERAM Algorithm 5 presented below. We denote by max_{ERAM} the maximum number of restarts for the ERAM, this number is fixed by the user. In what follows, we will keep this notation.

Algorithm 5 The Explicitly Restarted Arnoldi Method

Input: $A \in \mathbb{C}^{n\times n}$, $v_1 \in \mathbb{C}^n$, $\varepsilon_{Arnoldi} > 0$, $\varepsilon > 0$, $m \in [1,n]_{\mathbb{N}}$, $s \in [1,m]_{\mathbb{N}}, max_{ERAM} \in \mathbb{N}$, $\gamma \in [s,m]_{\mathbb{N}}$

1: $v_1 = \frac{1}{||v_1||} v_1$

2: **while** $(\varepsilon \geq \max_{j\in[1,s]_{\mathbb{N}}} \{resTr_j\})$ or $(max_{ERAM} > i)$ **do**

3: Execute m-step Arnoldi Method using $\{A,\ v_1,\ \varepsilon_{Arnoldi},\ m\}$

4: Solve the eigen problem $H_m Y_m = \Theta_m Y_m$

5: $U_{n,s} = V_{n,m} Y_{m,s}$

6: $resTr_j = ||Au_j - u_j\theta_j||, \forall j \in [1,s]_{\mathbb{N}}$

7: $v_1 = \sum_{j=1}^{\gamma} u_j^{(i)}$

8: **end while**

Output: $U_{n,s} \in \mathbb{C}^{n\times s}, \Theta_s \in \mathbb{C}^s, ResTr_s \in (\mathbb{R}^+)^s$

The ERAM Algorithm 5 adds an operation compared to the Restarted Arnoldi Method, which the restarting vector computation (step 8): It is a simple addition of γ approximated eigenvectors. Note that this operation does not imply neither additional parallel communication nor memory storage compared to the Restarted Arnoldi Method. By using such combination, we aim to favor convergence to the s desired eigenpairs simultaneously. However, the Arnoldi Method favors mostly the convergence to the dominant eigenpair. In this context, other restarting methods exist to take into account this specificity.

3.4 The Explicitly Restarted Arnoldi Method with Deflation

The Explicitly Restarted Arnoldi Method with Deflation [Saad 1980],[Saad 2011] drifts from the ERAM, adding a condition to the restarting vector and Krylov basis so that we take into account the Arnoldi Method characteristics regarding the dominant eigenpairs convergence.

The ERAM with Deflation restarts its process until the dominant eigenpair reaches the convergence: $\varepsilon \geq resTr_1$. At this point, we lock the u_1 approximated eigenvector into $V_{n,m}$ basis, getting to $V_{n,1} = u_1$. We restart the ERAM with Deflation using an Incomplete Arnoldi Method (please, refer to the Algorithm 3) of $m-1$ steps.

We illustrated the ERAM with Deflation considering that only the dominant eigenpair reached the convergence. It is entirely possible that first and second dominant eigenpairs

reached simultaneously the convergence.

In this case, we lock respectively u_1 getting to $V_{n,1} = u_1$, then orthogonalize u_2 with u_1 getting to $\tilde{u_2}$ (id est $\tilde{u_2} = V_{n,1} \perp u_2$) and finally proceed to a $m-2$ step Arnoldi Method with $V_{n,1} = u_1$ and $V_{n,2} = \tilde{u_2}$.

Algorithm 6 The Explicitly Restarted Arnoldi Method with Deflation

Input: $A \in \mathbb{C}^{n \times n}$, $v_1 \in \mathbb{C}^n$, $\varepsilon_{Arnoldi} > 0$, $\varepsilon > 0$, $m \in [1, n]_{\mathbb{N}}$, $s \in [1, m]_{\mathbb{N}}, max_{ERAM} \in \mathbb{N}$, $s \in [1, m]_{\mathbb{N}}$

1: $k = 1$
2: $v_k = \frac{1}{||v_k||} v_k$
3: **while** $(\varepsilon \geq \max_{j \in [1,s]_{\mathbb{N}}} \{resTr_j\})$ or $(max_{ERAM} > i)$ **do**
4: Execute $(m-k)$-step Arnoldi Method using $\{A, v_k, \varepsilon_{Arnoldi}, m, k\}$
5: Solve the eigen problem $H_{m-k} Y_{m-k} = \Theta_{m-k} Y_{m-k}$
6: $U_{m-k,s} = V_{n,m-k} Y_{m-k,s}$
7: $resTr_j = ||Au_j - u_j \theta_j||, \forall j \in [k, s]_{\mathbb{N}}$
8: Build $\tilde{u_k} = V_{m-k} \perp u_k$ and $||\tilde{u_k}||_2 = 1$
9: $v_k = \tilde{u_k}$
10: i=i+1
11: **if** $\varepsilon \geq resTr_k$ **then**
12: k=k+1
13: **end if**
14: go to step 4
15: **end while**

Output: $U_{n,s} \in \mathbb{C}^{n \times s}, \Theta_{s,1} \in \mathbb{C}^s, ResTr_{s,1} \in (\mathbb{R}^+)^s$

The ERAM with Deflation favors convergence to the (next) dominant (in terms or real or imaginary eigenvalue modulus) eigenpair. In the specific case of conjugated and/or clustered eigenpairs, the ERAM with Deflation is a numerically powerful algorithm.

In the case that one ERAM and one ERAM with Deflation are doing exactly the same number of restarts until convergence, one may notice that the ERAM with Deflation saves operations and global communications via the Incomplete Arnoldi Method. The ERAM with Deflation performs a $m-k$ step Arnoldi Method versus a m-step Arnoldi Method for the ERAM. However, we add orthogonalization operation for each desired eigenpairs (step 9 of the Algorithm 6).

The accuracy of approximated eigenpairs strongly depends on $V_{n,m}$ quality. Indeed, the quality of the Krylov basis fixes the quality of A projection onto $\mathbb{K}_{m,v_1}$. The deflation locks the converged approximated eigenvector to coerces $\mathbb{K}_{m,v_1}$ convergence to the desired eigenvectors subspace. It exists other variants, still butting in the Krylov basis to accelerate desired eigenpairs convergence.

3.5 The Implicitly Restarted Arnoldi Method

The Implicitly Restarted Arnoldi Method [Saad 1980],[Saad 2011] (IRAM) accelerates the convergence of s desired eigenpairs by using shift method. It is assumed that shifting the smallest A eigenpairs accelerates convergence of $\mathbb{K}_{m,v_1}$ onto the dominant desired eigen subspace. The reader may refer to [Saad 2011], [Carden 2011], [Sorensen 1990] (as a non-exhaustive list) to read more information about IRAM. In this paragraph, we focus mainly on the restarting step.

The IRAM fixes a shift value that we denote by $p \in [1, m-s]$: p represents undesired eigenvalues among m ones computed when solving the eigenvalue problem on $H_{m,m}$ matrix. Instead of shifting undesired eigenvalues on A, we shift them on $H_{m,m}$ matrix. Because of $H_{m,m}$ small size, this requires low computation costs and memory storage. To ensure the numerical consistency of Krylov basis, we operate some modifications on $V_{n,m}$ matrix. The IRAM Algorithm is presented bellow:

Algorithm 7 The Implicitly Restarted Arnoldi Method

Input: $A \in \mathbb{C}^{n\times n}$, $v_1 \in \mathbb{C}^n$, $\varepsilon_{Arnoldi} > 0$, $\varepsilon > 0$, $m \in [1,n]_{\mathbb{N}}, max_{IRAM} \in \mathbb{N}$, $s \in [1,m]_{\mathbb{N}}$

$p = 0$
while $(\varepsilon \geq \max_{j\in[1,s]_{\mathbb{N}}} \{resTr_j\})$ or $(max_{IRAM} > i)$ **do**
  $v_1 = \frac{1}{||v_1||} v_1$
  Execute $(m-p)$-step Arnoldi Method using $\{A, v_1, \varepsilon_{Arnoldi}, m, p\}$
  Solve the eigen problem $H_m Y_m = \Theta_m Y_m$
  $U_{m,s} = V_{n,m} Y_{m,s}$
  $resTr_j = ||Au_j - u_j\theta_j||, \forall j \in [1,s]_{\mathbb{N}}$
  if $\forall j \in [1,s]_{\mathbb{N}}, \varepsilon \geq resTr_j$ **then**
    stop.
  else
    Fix $p \in [1, m-s]_{\mathbb{N}}$ the number of eigenvalues to shift
    for $j = 1, p$ **do**
      Build a QR factorization of $\tilde{H_m} - \theta_j I$
      $\tilde{H_m} = {Q_j}^* \tilde{H_m} Q_j$ where $\tilde{H_m} - \theta_j I = Q_j R_j$
      $V_m = V_m Q_j$
      $v_1 = v_1^H V_m$
    end for
  end if
end while

Output: $U_{n,s} \in \mathbb{C}^{n\times s}, \Theta_{s,1} \in \mathbb{C}^s, ResTr_{s,1} \in (\mathbb{R}^+)^s$

The IRAM Algorithm 7 is numerically very robust thanks to the shift operation.

Similarly to the ERAM Deflation, the IRAM performs an Incomplete Restarted Arnoldi Method, savings operations compared to the ERAM algorithm. However, it adds a QR factorization (which is a sequential operation), a dense matrix/matrix multiplication (of size (n,m) $\times$ (m,m))and a matrix/vector operation.

3.6 The Krylov Eigen Solvers

Each restarting strategy of the presented Krylov Solvers have benefits and/or drawbacks in terms of parallel computing and numerical accuracy.

3.6.1 Fixing the Restarted Arnoldi Method Parameters

For all the presented Krylov eigen solvers, we emphasized that m value is the key parameter that leads de facto the numerical accuracy and the execution time.

It is assumed that the larger m is, the more accurate the approximated eigenpairs will be, but this point must be moderated as opposite behavior may be observed (we will return to this point later).

It is also known that the larger m is, the longer the execution time per Arnoldi Method iteration will be due to blocking and global parallel communications. Additionally, one may have observed that in the case of IRAM, computing QR step is a sequential operation, therefore a large m value can rapidly lead to a bottleneck. We aim to fix m such as its value is not too large to satisfy the parallel operations constraints but ensures the convergence for the desired eigenpairs.

In terms of energy consumption, a Restarted Arnoldi Method (RAM) with a too small m may not converge fast enough, leading to a poor flops/watt ratio. On the other side, a large m value will increase memory storage of the Krylov system as well as global and blocking parallel communications used to build the Krylov subspace: it may imply more memory transferts and default-cache that will increase energy consumption per restart of the RAM. One may see that this fragile equilibrium is neither simple to reach nor its evolution is uniform according to subspace size. Krylov subspace size has many effects, some are directs and many indirects.

There is no general method to fix this m value for all variants of the Restarted Arnoldi Method with respect to s and definitely no methods to ensure the convergence to s desired eigenpairs with a m value optimizing the execution time per restart.

Many research has been done to provide a relevant m value, the reader may find $m = 2s$ which is the most common used value: it is fixed to satisfy on one side the system convergence and on the other side the parallel computation/communication costs. As an illustration, this is the default m value in the Restarted Arnoldi versions implemented in *Scalable Library for Eigenvalue Problem Computations (SLEPc)* [SLE], [Hernandez 2007a], [Hernandez 2006].

However, for some matrices, $m = 2s$ may be not sufficient and never provide the convergence at a satisfiable tolerance. Unfortunately, such configurations can not be

prevented at the runtime. Potentially, you may run several times the Restarted Arnoldi Method with different m values to obtain the convergence at the desired threshold.

Some tricky cases also appears considering m value. It may happen that an ERAM with a m_1 value converges faster (numerically speaking but also in terms of execution time) than an ERAM using m_2 such that $m_2 > m_1$.

For some cases, it may be wised to compute a larger number of eigenvalues to get s desired ones faster. As the Arnoldi Method favors dominant eigenpairs convergence, computing γ eigenpairs such that $m \geq \gamma \geq s \geq 1$ may accelerate the convergence of s desired eigenpairs. However, additional eigenpairs must have a correct threshlod (compared to the one we desire), otherwise we may add only numerical disturbance, leading to slow-down convergence.

Select a relevant $\{m, s, \gamma\}$ for the complete Restarted Arnoldi Method (and its variants presented above) is very tricky. It depends on the matrices itself but also on the eigenvalue distribution, the convergence rate, the eigenvectors convergence et caetera. Some of these parameters depends on runtime execution and can not be predicted before.

As a remedy, Hybrid methods have been developed to accelerate the presented Krylov eigen solvers convergence. The Multiple Restarted Arnoldi Method executes concurrent Restarted Arnoldi Methods, all solving the same eigenvalue problem but with different input parameters, so as to generate different eigen-information on each solver. Each solver shares asynchronously its eigen-information with others to accelerate its own process.

3.7 The Multiple Restarted Arnoldi Method

With upcoming exascale, the current trend is to avoid problematic parallel operations and communications from our current algorithms to improve their scalability.

As an illustration, an optimum exascale version of the Arnoldi Method would avoid every dot products and matrix/vector products, but such scheme leads to an unconsistent Arnoldi Method.

The mathematicians, numericians and computer scientists must find compromises to reduce global and blocking parallel communications/operations while maintaining mathematical validity of the method.

The Multiple Restarted Arnoldi Method grew out of this compromise. We recall that the Multiple Restarted Arnoldi Method (MRAM) is not new, as it appeared in the 1990's [Edjlali 1994], [Edjlali 1995], [Edjlali 1996]. Many research still goes on this topic [Emad 2001],[Emad 2005],[Dubois 2011a]. The Multiple Restarted Arnoldi Method is in fact a large concept.

To illustrate our purpose, let's consider μ Restarted Arnoldi Method (RAM) and we denote by RAM_k the k^{th} Restarted Arnoldi Method. Each RAM_k solves the same eigenvalue problem, id est computing s approximated eigenpairs of A with a fixed threshold ε: these two parameters are fixed among every Restarted Arnoldi Method inside a MRAM solver. Other parameters, such as m, γ and v_1 can differ. Algorithm 8 describes the concept of Hybrid Restarted Arnoldi Methods. This Algorithm is intentionally generic,

we will refine it later. In what follows, we use the two indexes $(k,l) \in [1,\mu]_{\mathbb{N}}^2$ such that $l \neq k$ to distinct two RAM solvers inside a MRAM.

Algorithm 8 The Multiple Restarted Arnoldi Method

Input: $A \in \mathbb{C}^{n\times n}$, $k \in \mathbb{N}^*$, $\varepsilon > 0$, $s \in \mathbb{N}$

1: **while** $\forall k \in [1,\mu]_{\mathbb{N}}$, RAM_k has not converged **do**
2: RAM_k executes a restart (cf Algorithm 3.2) with its RAM parameters $\{v_1^{(i_k)}, m_k, max_{ERAM,k}, \gamma_k\}$ using np_k tasks
3: **if** RAM_k has converged **then**
4: Send ASYNCHRONOUSLY STOP signal to RAM_l
5: Stop.
6: **else**
7: **if** RAM_k has received an ASYNCHRONOUS message from RAM_l, $\forall l \in [1,\mu] l \neq k$ **then**
8: **if** STOP signal **then**
9: Stop.
10: **else**
11: **if** RAM_l eigen-data are more accurate **then**
12: RAM_k uses RAM_l, $\forall l \in [1,\mu] l \neq k$, eigen-data to compute $v_1^{(i_k+1)}$
13: **else**
14: RAM_k uses its eigen-data to compute $v_1^{(i_k+1)}$
15: RAM_k Sends ASYNCHRONOUSLY its eigen-data to RAM_l, $\forall l \in [1,\mu] l \neq k$
16: **end if**
17: go to step 2.
18: **end if**
19: **end if**
20: **end if**
21: **end while**

Output: $U_{n,s} \in \mathbb{C}^{n\times s}, \Theta_{s,1} \in \mathbb{C}^s, ResTr_{s,1} \in (\mathbb{R}^+)^s$

We recall that m, γ and v_1 have an impact on the numerical efficiency and the execution time of one RAM:

- m value has an impact on Ritz values accuracy: it assumed that a RAM with a large m value will execute less restarts than a RAM with a small m value[1]. m has a second impact, in terms of parallel execution time of the RAM. A RAM with a small m value will execute faster each of its restarts compared to a RAM with a large m. We conclude on the fact that there is an equilibrium (that depends on m) between the parallel execution time per restarts and the number of restart to execute to reach the convergence.

[1] Nevertheless, we recall that the opposite behavior can be observed.

- γ (the number of Ritz eigenpairs used to compute the restarting vector) has a numerical impact on the RAM convergence. Its impact on the execution time per restart is negligible compared to m impact. If $\gamma > s$, then we add $\gamma - s$ matrix/matrix multiplications (two dense matrices of size $n \times m$ and $m \times (\gamma - s)$ respectively) as well as $\gamma - s$ matrix/vector (sparse or dense matrix whose size is $n \times n$ and a dense vector of size n) and finally $\gamma - s$ dot operations.

- Computing the restarting vector v_1 has no impact on the parallel execution time. This operations weight the Ritz eigenvectors with coefficients and sum them. Therefore, the restarting vector is an interesting parameter to reduce *"at no cost"* the number of restarts and therefore the parallel execution time of the complete RAM.

Keeping these capital informations in mind, we will present the MRAM abilities in the context of ultra-scale computing.

3.7.1 Resiliency Property

With future exascale supercomputers, solvers resiliency is essential. It is assumed that the more nodes we have, the shorter the Mean Time Between Failure (MTBF) is, leading to the fact that an exascale simulation has higher probability to face to a system failure during its execution. In fact, the decrease of MTBF can not be attributed only to the higher number of nodes (and cores). The hybridization of supercomputers leads to more complex software to manage such architectures. The management of many parameters that where so far considered as *"non prius"* adds complexity to the applications, software and resources operations.

If the resiliency may appear as being only part of the hardware and software domains, it must to be integrated into the algorithms design.

As an illustration, one may note that RAM is not endowed of resiliency property. During RAM parallel execution, if one task fails (for any reason mentioned above), its results are lost and the global system resolution is then lost. Basically, we must restart the RAM from the very beginning.

As we can not afford this luxe, resiliency must be integrated at every levels of High Performance Computing. The resiliency property is inherent to the MRAM concept. Indeed, if one RAM is affected by a hardware and/or software breakdown, only the communicator associated to this RAM will be down. As all RAMs compute their eigenproblem independently from each others, they will not be affected by the failed RAM. A study of MERAM resiliency has been conducted in [Dubois 2011a].

Even better, in the case that we can resurrect the failed RAM, it will begin its own process with a restarting vector coming from a converging RAM. All this scheme is available in the case that we are using a MPI implementation that supports local fails, id est sub-communicators are not affected by one (or more) failed tasks used in an other sub-communicator.

3.7.2 Reduce the Global and Blocking Communications

The MRAM eigensolver has many other benefits for exascale supercomputing. The structure of independent Multi-Methods but collaborating numerically altogether adduces many assets. We evoked in section 3.1 that the RAM efficiency in terms of numerical results and swiftness to compute the Ritz eigenpairs is dependent on the Krylov subspace size m value.

The MRAM eigensolver aims to optimize both parameters "numerical accuracy" and "execution time". The MRAM will use RAMs with a large m value, and others with a small m value. In that case, the RAMs with a large m value will produce more accurate results but they also will be slower in terms of parallel execution time (each restart has a longer parallel execution time compared to a RAM with a small m value). Therefore, there are two possible scenarios:

- RAMs with a **small** subspace size will provide more accurate data than the RAMs with a **large** subspace size. Such behavior may be due itself to two factors:
  - Whether RAMs with a **small** subspace size will execute more restarts as they are faster in terms of execution time per restart and therefore their convergence has more progressed compared to the RAM with a **large** m. Therefore, they may accelerate the ERAMs with a **large** subspace size in this case.
  - Whether RAMs with a **small** subspace size provide more accurate data than RAMs with a **large** subspace size, as this phenomena can be observed.
- ERAMs with a **large** subspace size will provide more accurate data than RAMs with a **small** subspace size. RAMs with a small subspace size will take into account the most accurate eigen-data among their asynchronous receptions. In this case, still two scenarios may happen:
  - RAMs with a **small** subspace size may converge first, starting from an accurate restarting vector and executing faster each of their restarts.
  - RAMs with a **large** subspace size is the first to converge, leading to a *Eater/Feeder* behavior: RAMs with **small** subspace size impact will be null. This scheme is not satisfiable as it means that we used resources for RAMs that could not participate neither to the eigen-data enrichment nor to reduction of parallel execution time.

The balance between numerical accuracy and execution time is managed by these two (forsooth 4) scenarios and of course by the nondeterministic behavior of the MRAM: The asynchronous characteristic of the inter-ERAMs communications does not allowed to prevent or control the MRAM behavior.

We now introduce an other important property of the MRAM in terms of parallel communications. Let's consider one MRAM, running on np_{MRAM} tasks and one RAM running on np_{RAM} tasks.

The single RAM running on np_{RAM} tasks will execute its blocking and global parallel communications on np_{RAM} tasks.

In the case of the MRAM, each RAM will compute the same operations on its sub-communicator, id est on np_k ($np_{MRAM} > np_k$) tasks, where np_k is the number of tasks used for the k^{th} RAM. The MRAM reduces the size of the blocking and global parallel communications which is a crucial asset in the context of up-coming exascale computing.

Moreover, MRAM may optimize the parallel execution time with respect to the subspace size by dedicating more tasks to RAMs with a large subspace size and less tasks to the RAM with a small subspace size. This aims to balance parallel execution time of all RAMs to favor the eigen-data exchange and limit *Eater/Feeder* behavior mentioned earlier.

3.7.3 Multiple Levels of Parallelism

In the same area, the MRAM has an additional parallel level compared to a single RAM. If this may be seen as an additional complexity, this is in fact a key asset in the context of current and future supercomputers. With petascale area, the levels of parallelism have increased but the algorithms have not necessarily the possibility to exploit them.

The highest parallel level of the RAM is represented by the numerical kernels, such as the Arnoldi Method, followed by the projection of $H_{m,m}$ eigenvectors onto $\mathbb{K}_{m,v}$ and finally computation of desired eigenpairs residuals.

With the MRAM algorithm, we add a higher parallel level than RAM, which gathers the asynchronous communications between each RAMs. This one does not affect the lowest and medium RAM parallel levels, it just adds and exploits more parallelism and enhance the parallelism/scalability of the original RAM.

3.7.4 Exploit Heterogeneous Architectures

Due to its algorithm (8), the MRAM is an hybrid[2] eigensolver. In the case that we dispose of several implementations of RAM, each of them implemented and optimized for a specific architecture, MRAM allows to exploit them altogether.

Each RAM solves its own problem asynchronously from each other and sends at each restart its own eigen-data. This completely matches with the supercomputers having an heterogeneous architecture. Indeed, the current widely used accelerators such as Graphic Process Units (GPU) or Many Integrated Cores (MIC, Intel®) may exchange messages by using the PCI-Express. We can execute a MRAM using many RAMs while none of them is executed on the same hardware.

[2]Hybrid in terms of supercomputer architectures.

3.8 The Multiple Restarted Arnoldi Method Variants

In this section, we exposed the MRAM concept, by using the most basic eigensolver presented so far, the Restarted Arnoldi Method.

The same concept can be applied to all the Krylov eigensolvers presented above, id est the Explicitly Restarted Arnoldi Method, the Explicitly Restarted Arnoldi Method with Deflation and finally the the Implicitly Restarted Arnoldi Method.

As a general extension, all these solvers could be mixed together, leading to a Multiple Hybrid Restarted Arnoldi Method, using as an illustration an ERAM and an IRAM.

Nevertheless, the ERAM is the most convenient Krylov eigen solver (among the one presented in this chapter) to constitute a MRAM (in this case, a Multiple Explicitly Restarted Arnoldi Method). We will detail this point in what follows.

3.8.1 The Multiple Explicitly Restarted Arnoldi Method

The MERAM is composed of several independent instances of ERAM, each solving the same eigenvalue problem, which is finding the s dominant[3] eigenpairs of a large non-hermitian matrix A.

We emphasized in section 3.7 the impact of m, γ and v_1 parameters onto the numerical accuracy and the ERAM parallel execution time. It appears that v_1 optimizes this ratio, as it requires neither blocking/global parallel communications nor memory storage.

In the context of MERAM, each ERAM will send to its companions $n \times s$ eigenvectors, s eigenvalues and finally s residuals, leading to $n \times s + 2s$ data. Therefore, in the most general case (we suppose that there is no filter to asynchronously send the eigen-data) : An ERAM (as a member of a MERAM) asynchronously sends (respectively receives) $(\mu - 1) \times (n \times s + 2s)$ data at maximum.

Note that the ERAM may have several ways to compute the restarting vector v_1 this point particularly will be detailed in this thesis). This means that we do not only vary the subspace size in the case of MERAM (leading to embedded Krylov subspaces).

Many research have been developed to improve the MERAM since its creation in 1994 [Edjlali 1994]. Several configurations appeared, we will cite two major contributions to MERAM but this is a non-exhaustive list.

- A relatively recent MERAM version aimed to exchange only the restarting vector v_1 between ERAMs. This means that $ERAM_k$ computed $v_1^{(i_k+1)}$ with its eigen-data and sends it to $ERAM_l$. As a first observation, this considerably reduces the size of asynchronous messages. At maximum, $ERAM_k$ receives (respectively sends) $(\mu - 1) \times n$ data. If this is a good point, it also reduces drastically the *degrees of freedom* of MERAM (mentioned above), which is the driving force of MERAM (these implementation has been discussed in [Dubois 2011a]).

[3] In terms of real or imaginary eigenvalue modulus

- The second implementation allows each ERAM to get different parameters (such as γ, m and v_1 by asynchronously share the Ritz eigenpairs and their associated residuals or a specific restarting vector.

3.8.2 The Multiple Implicitly Restarted Arnoldi Method

The MIRAM consists in several independent instances of the IRAM. Each IRAM is solving the same eigenvalue problem, which is finding the s dominant[4] eigenpairs of a matrix A.

The Multiple Implicitly Restarted Arnoldi Method (MIRAM) shares eigen-data to accelerate each IRAM convergence. If MERAM has one liberty degree regarding the data we can asynchronously share, the IRAM has no choice.

The IRAM restarts its process using the matrices $H_{m,m}$ and $V_{n,m}$: these two matrices contain the "historic" of present and passed shift operations. Therefore, if $IRAM_k$ wants to send its data to accelerate the $IRAM_l$, it must send basically its matrices ${H_{m,m}}^{(i_k)}$ and ${V_{n,m}}^{(i_k)}$.

In this context, each IRAM will asynchronously send to its companions $n \times m$ vectors and $m \times m$ data to each IRAM. Therefore, in the most general case (we sill suppose there is no filter to asynchronously send the data) : An IRAM (as a member of a MIRAM) asynchronously sends (respectively receives) $(\mu - 1) \times (m(m + n))$ data at maximum.

First of all, this amount of data -even though those are asynchronously sent- is considerably larger than the MERAM one. Secondly, this implicitly implies that the only *"degree of freedom"* of MIRAM is the subspace size m, while MERAM has three *"degrees of freedom"*.

We will not detail the MERAM with Deflation case, as it remains the same problematic as MIRAM: as ERAM Deflation restarts its process by *disrupting* the $V_{n,m}$ basis, it leads to the same conclusion as MIRAM, meaning that MERAM with Deflation asynchronously exchange the Krylov basis and upper Hessenberg matrix of each component.

[4] In terms of real or imaginary eigenvalue modulus

Conclusion

In this chapter, we presented a palette of Krylov eigen solvers. We recall that it exists on one hand many other variants of eigensolvers, and on the other hand, many other Krylov eigenvalue solvers. In the context of nuclear physics simulation, the Krylov eigensolvers are particularly relevant.

We presented the Krylov subspace size issue and how it leads (in an opposite direction) the parallel and numerical efficiency of the presented Krylov eigensolvers. We emphasized the restarting strategy role for each Krylov eigensolver, how this parameter influences their numerical efficiency and affect the parallel computation part.

We then presented each Krylov eigensolver in the context of Multiple Restarted Arnoldi Method, by highlighting the benefits and drawbacks regarding the degrees of freedom offered by such hybrid methods.

The MERAM eigen solver is the most relevant in terms of degrees of freedom compared to MIRAM of a MERAM Deflation. We argued that MERAM composed of ERAM with many different parameters, especially the restarting vector v_1 can lead to a great improvement of the convergence compared to a single ERAM.

What if, thanks to a Smart-MERAM, we could evaluate/measure each $ERAM_k$ convergence rate, appreciate (or depreciate?) each $ERAM_k$ parameter set and turned them to (more) efficient ones? What if we dispose of Smart-ERAM, that could adapt their parameters dynamically with respect to the received eigen information and their own eigen information?

Participate to this debate firstly required a in-depth review of the ERAM eigen solver itself. In what follows, we will focus on the ERAM convergence, especially how could we measure and characterize it in order to further appreciate the input parameters. In a second time, we will present matrix generators, starting from an input spectrum, that helped us to proceed to numerical performance tests. Such matrix generators have a huge interests in the context of ultra-scale eigensolvers.

Finally, we will present the sensibility of the ERAM convergence with respect to the restarting vector computation and the orthogonalization process.

Chapter 4

Describe and Characterize the ERAM Convergence

This chapter presents a heuristic to define and characterize the ERAM convergence. This heuristic is the basis for smart-tuning methods to enhance the equilibrium of the ERAM convergence and parallel abilities.

4.1 Define the ERAM Convergence

We denoted by $res_j^{(i)}$ the residual associated to the j^{th} dominant (in what follows, we will focus on the real modulus eigenvalue) eigenpair, $\forall j \in [1,s]_{\mathbb{N}}$. We consider that an ERAM has converged at the ε threshold ($\varepsilon \in \mathbb{R}^{*+}$) when the s desired eigenpairs have reached this threshold themselves, id est if and only if $\forall j \in [1,s]_{\mathbb{N}}, \varepsilon \geq res_j$.

4.1.1 The Convergence Criteria

There exists some variants to estimate the ERAM eigenpairs convergence. The most commonly used are listed in the Equation 4.1:

$$\begin{cases} \forall j \in [1,m]_{\mathbb{N}}, res_j^{(i)} = & ||Au_j^{(i)} - \theta_j^{(i)} u_j^{(i)}||, \\ = & \dfrac{||Au_j^{(i)} - \theta_j^{(i)} u_j^{(i)}||}{||A|| + |\theta_j^{(i)}|}, \\ = & \dfrac{||Au_j^{(i)} - \theta_j^{(i)} u_j^{(i)}||}{|\theta_j^{(i)}|}, \end{cases} \tag{4.1}$$

- $||Au_j^{(i)} - \theta_j^{(i)} u_j^{(i)}||$: This convergence criteria is the most common criteria. It simply uses $resTr_j^{(i)}$ formula (equation 3.7).
- $\frac{||Au_j^{(i)} - \theta_j^{(i)} u_j^{(i)}||}{||A|| + |\theta_j^{(i)}|}$: This convergence criteria takes into account the matrix norm, which is a pertinent parameter. Computing $||A||$ may be costly in terms of parallel operations and communications, however it remains available for all the ERAM restarts.
- $\frac{||Au_j^{(i)} - \theta_j^{(i)} u_j^{(i)}||}{|\theta_j^{(i)}|}$: This convergence criteria is relative to the Ritz eigenvalues only. This criteria is a good compromise between the two mentioned above in terms of

accuracy and parallel operations/communications costs. We emphasize that this residual is the default value of SLEPc Library [SLE]. In what follows, we will keep this residual formula.

Each eigenpair convergence is evaluated by its own $res_j^{(i)}$. Therefore, we can easily estimate each eigenpair convergence starting from its own index. The Arnoldi process favors the exterior eigenvalues convergence, id est the dominant and lowest eigenpairs: therefore, the j^{th} eigenpair may converge slower than the $j+1^{th}$ one ($j \in [1, \gamma]_{\mathbb{N}}$). Each eigenpair convergence may have a *"chaotic"* convergence: by *"chaotic"*, we mean that considerable or relatively small gap may be observed for the residual evolutions.

Based on this observations, it is arduous to describe the global system convergence. Nevertheless, there exists methods to measure and estimate the convergence of the iterative Krylov subspaces to an invariant subspace, but those are applied to the linear system solvers. One major difference is that the Krylov methods for the linear systems must converge to one vector solution, while the Krylov methods applied to the eigen systems must converge to a subspace of Ritz eigenvectors. The convergence algorithms applied to the Krylov methods for the linear systems use as metric the successive Krylov subspace angles.

4.1.2 Convergence Study applied to the Krylov Method for Linear System

The Generalized Minimum Residual Method (GMRES) is a widely spread Krylov Method to solve the linear system problems. It has been introduced by Yousef Saad and Martin H. Schultz in 1986 [Saad 1986] and aims to solve linear system of equations such that $Ax = b$ where $A \in \mathbb{C}^{n \times n}$ is a large dense or sparse non-symmetric matrix.

We choose to present the convergence criteria of this method as its concept is very similar to the ERAM one.

Based on the Arnoldi Method (cf Chapter 3), the GMRES is an iterative solver that computes an approximation $\tilde{x} \in \mathbb{C}^n$ of the real solution $x \in \mathbb{C}^n$ such that $Ax = b$.

We first choose a random initial guess x_0 and compute its associated residual $r_0 = b - Ax_0$. The first GMRES restart computes $x_1 \in x_0 + \mathbb{K}_{m,r_0}$ where $\mathbb{K}_{m,r_0} = Span\{r_0, Ar_0, ..., A^{m-1}r_0\}$.

The GMRES iteratively builds $x_i \in x_{i-1} + \mathbb{K}_{m,r_{i-1}}$ meaning that it provides at the i^{th} restart a x_i vector that approximates x at r_i threshold. The complete GMRES reaches the convergence when $\varepsilon \geq r_i$ where $\varepsilon \in \mathbb{R}^{*+}$ is the desired threshold fixed by the user.

Similarly to the ERAM, the GMRES is restarted due to the memory storage and computational operations required at each restart (more details can be found in [Saad 1986] and [Saad 2003] (pages 164-184)).

Such as the ERAM, it is assumed that the larger m value, the better the GMRES converges. However, in some cases the opposite behavior can be observed (such as ERAM).

The GMRES is facing to the tricky equilibrium between improving its convergence rate (by using a large m value) and reduce its memory, computation operation and block-

ing/global parallel communications consumption [Joubert 1994] for the exact same reasons as ERAM.

Many research has been done to efficiently fix the m parameter with respect to its convergence status. There are different approaches to fix the subspace size values, as a non-exhaustive list, we can increase or (respectively and) decrease its value leading to a one (respectively two) direction(s) evolution.

The new m value is fixed depending on the GMRES convergence and aims to optimize the computational constraints (memory storage, blocking/global parallel communications, number of operations) versus the system convergence (number of restarts).

There are many possibilities to fix the m value depending on the metric used. If heuristics may differ, all auto-tuning tools have a common point, which is the GMRES convergence estimation. This metric is a key parameter for the GMRES Krylov subspace size auto-tuning, however, its value may not be the same for every auto-tuning.

The metric used in [Baker 2005] is a classic metric that has inspired many research on subspace size tuning for the GMRES. The work presented in [Baker 2005] has been widely used in recent work [Katagiri 2012] to improve their heuristic.

Other research trying to optimize the parallel computation ratio and numerical efficiency for the GMRES have been done using different metrics and different heuristics ([H. Kuroda 2000] as an example)

In our case, the same convergence metric as the GMRES can not be used for the ERAM, as we aim to converge to an eigen subspace instead of a single vector. The mathematical properties of the GMRES convergence metric presented in [Baker 2005] are no longer available in the case of ERAM. However, the idea remains close from it.

4.2 Detect and Define the ERAM Convergence at the Runtime Execution

All the work and results presented in this chapter have been executed in the Lawrence Berkeley National Laboratory with the expert advices of M. Leroy Anthony Drummond. I gratefully Thanks M. Leroy Anthony Drummond for this rewarding experience.

The Krylov methods convergence is pretty chaotic, especially during the first restarts. The aim of the ERAM convergence algorithm is to detect, the earliest as possible the divergence or the stagnation behavior, in order to change some ERAM parameters.

The final objective is to avoid or limit the stagnation and divergence status by using pertinent ERAM parameters, leading to a smoother ERAM convergence.

Firstly, we need an index to estimate the global ERAM convergence, based on this index value we may define the ERAM behavior. There may be several possibilities regarding the ERAM criteria convergence. Throughout all this thesis, we will denote by $res_{CV}^{(i)}$ the convergence criteria used at the $i^{(th)}$ restart.

In our study, we will focus on the lowest convergence eigenpair. The convergence of $\theta_j^{(i)}$ does not evolves at the same rate $\forall$ j $\in [1, s]_{\mathbb{N}}$. Therefore, a good criteria to define

the ERAM convergence evolution is:

$$res_{CV}^{(i)} = \max_{j \in [1,s]_{\mathbb{N}}} \left(\frac{||Au_j^{(i)} - \theta_j^{(i)} u_j^{(i)}||}{|\theta_j^{(i)}|}\right), \tag{4.2}$$

4.2.1 The ERAM Local Convergence

We use the convergence criteria $res_{CV}^{(i)}$ to estimate the ERAM convergence based on two successive restarts. This aims to detect the ERAM convergence behavior restart per restart. The following basic heuristic measures the $res_{CV}^{(i)}$ evolution at each restart thanks to an interval fixed by the user: we denote by $(f_{inf}, f_{sup}) \in]0,1[^2$ the two parameters fixed by the user to compute the convergence tolerance interval. We shall shortly return on this subject in what follows.

Algorithm 9 Restart per Restart Convergence Estimation Algorithm

Input: residuals $(res_{CV}^{(i)}, res_{CV}^{(i-1)}) \in \mathbb{R}^{+2}$, $(f_{inf}, f_{sup}) \in]0,1[^2$.

status=undefined
$born_{sup} = \frac{res_{CV}^{(i-1)}}{f_{sup}}$
$born_{inf} = res_{CV}^{(i-1)} f_{inf}$
if $born_{sup} \geq res_{CV}(i) \geq born_{inf}$ **then**
    status=stagne
else if $born_{inf} > res_{CV}(i)$ **then**
    status=converge
else
    status=diverge
end if

Output: status

We define a tolerance range $[born_{inf}, born_{sup}]$ depending on the previous residual value. We consider that if current residual is in this interval, the evolution is not significant (whether $res_{CV}^{(i)} > res_{CV}^{(i-1)}$ or $res_{CV}^{(i-1)} > res_{CV}^{(i)}$) and qualify it as stagnant.

This choice is inspired from [Baker 2005] except that modifications were required to fix the interval. The Algorithm 9 is an "instantaneous" convergence algorithm but this does not provide an accurate tendency of the ERAM convergence. We improved this heuristic in order to consider the global ERAM state convergence.

4.2.2 The ERAM Multi-Levels Convergence

The Algorithm 10 studies the convergence status using different rates. Starting from the Algorithm 9, we analyze based on several successive restarts the ERAM convergence.

Algorithm 10 Multi-Levels Convergence Algorithm

Input: residuals $(res_{CV}^{(i)}, res_{CV}^{(i-1)}) \in \mathbb{R}^{+2}$, $(f_{inf}, f_{sup}) \in]0,1[^2$, $max_{count} \in \mathbb{N}^*$.

status=undefined
$born_{sup} = \frac{res_{CV}^{(i-1)}}{f_{sup}}$; $born_{inf} = res_{CV}^{(i-1)} f_{inf}$
$div_{sup} = res_{CV}^{(i-1)} * 10$
$diverge_{it} = 0$, $stagne_{it} = 0$
$local_{MAX} = 0$, $local_{MIN} = +\infty$
status=undefined
if $born_{sup} \geq res_{CV}(i) \geq born_{inf}$ **then**
  $stagne_{it} = stagne_{it} + 1$
  if $stagne_{it} \geq max_{count}$ **then**
    $stagne_{it} = 1$
    status=stagne
  end if
end if
if $res_{CV}^{(i)} > born_{sup}$ **then**
  $diverge_{it} = diverge_{it} + 1$
  if $(diverge_{it} \geq max_{count})$ OR $(res_{CV}^{(i)} \geq div_{sup})$ **then**
    $diverge_{it} = 1$
    status=divergence
  end if
end if
if divergence OR stagnation **then**
  $local_{MAX} = 0$
  $local_{MIN} = +\infty$
end if
if $tolerance_{inf} > res_{CV}^{(i)}$ **then**
  if $(res_{CV}^{(i-1)} \geq local_{MAX})$ **then**
    $local_{MAX} = res_{CV}^{(i-1)}$
    $interval_{begin} = i - 1$
    status=convergence
  end if
  if $(local_{MIN} \geq res_{CV}^{(i)})$ **then**
    $local_{MIN} = res_{CV}^{(i)}$
    $interval_{end} = i$
  end if
  if $interval_{end} \neq interval_{begin}$ **then**
    $rate = log_{10}(\frac{res_{CV}^{(interval_{end})}}{res_{CV}^{(interval_{begin})}})$
    if $rate \geq 4$ **then**
      status=high convergence
    end if
  end if
end if

Output: status

We highlight that the Algorithm 10 uses data that have been previously computed: we mean that we do not add neither computation operation nor communication to measure the ERAM convergence, so as the Algorithm 10 computation costs is negligible.

The Algorithm 10 to detects as a priority the sudden convergence peak or stagnation state. These two status are our priority as we want to avoid them as soon as possible. This version of convergence treats with accuracy the convergence state. Indeed, preserve this status remains important. The $[born_{sup}, born_{inf}]$ interval coupled with max_{count} value will favor some specific convergence schemes, as explained below:

- A large interval $[born_{inf}, born_{sup}]$ and a large max_{count} value:

  The larger the interval $[born_{inf}, born_{sup}]$ is, the more we favor the stagnation state in case of a slow evolution. However having a high value of max_{count} limits some "abusive" stagnation status. The high $born_{inf}$ and $born_{sup}$ values will limit the slow convergence and divergence status compared to other configurations.

- A large interval $[born_{inf}, born_{sup}]$ and a small max_{count} value: In this configuration, we favor the detection of the convergence status if $born_{inf}$ is quite high. This configuration favors also the stagnation status and slow divergence detection. Some stagnation parts may be considered as a slow convergence due to the interval gap.

- A small interval $[born_{inf}, born_{sup}]$ and a small max_{count} value: This configuration favors the slow evolution for each status. We mean that slow divergence/convergence will be detected thanks to such configuration. If $born_{sup}$ (respectively $born_{inf}$) value is low (respectively high) then the stagnation (respectively convergence) status will be dominant.

- A small interval $[born_{inf}, born_{sup}]$ and a high max_{count} value: This configuration favors the slow convergence detection and limit the stagnation detection interval.

In what follows, we apply the Algorithm 10 to the target matrices summarized in the Table 4.1. These matrices spectrum distribution are very different (the matrices spectrum will be detailed in the next chapter), such as their sparse distribution schemes. All of these matrices are non-symmetric.

Name	Size	Field	Source
Ex11	16614	computational fluid dynamics problem	UF SMC[Davis]
Mixtank_new	29,957	computational fluid dynamics problem	UF SMC[Davis]
Rim	22,560	computational fluid dynamics problem	UF SMC[Davis]

Table 4.1: Target Matrices to Test the Algorithm 10.

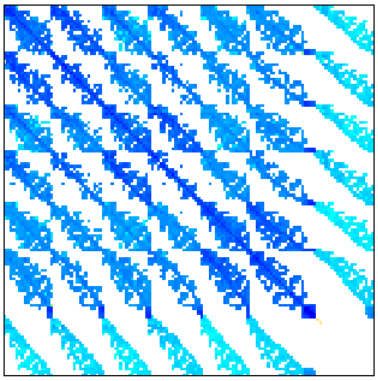

Figure 4.1: Mixtank_new Matrix Sparsity

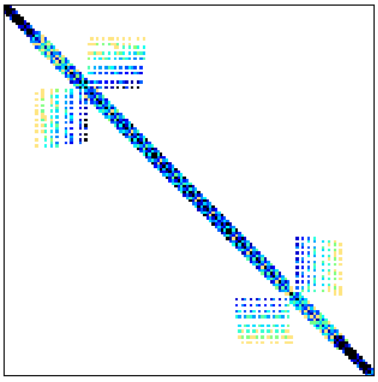

Figure 4.2: Ex11 Matrix Sparsity

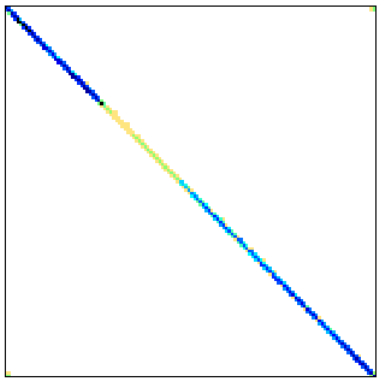

Figure 4.3: Rim Matrix Sparsity

In the next paragraphs, we present the Algorithm 10 results applied to the target matrices listed above. The Algorithm 10 parameters are $max_{count} = 3, born_{sup} = 0.2$ and $born_{inf} = 0.8$.

Each presented ERAM uses $\gamma = s = 5$ (where s is the number of desired eigenpairs and γ the number of Ritz eigenvectors used to compute the restarting vector) and a CGSR orthogonalization process. Restarting vector is computing as presented in the equation 3.8.

All the results presented below have been executed on the PRACE Curie supercomputer [1].

[1] The 26^{th} most powerful supercomputer according to the Top500 list of June 2014

4.2.2.1 The $Ex11$ Matrix

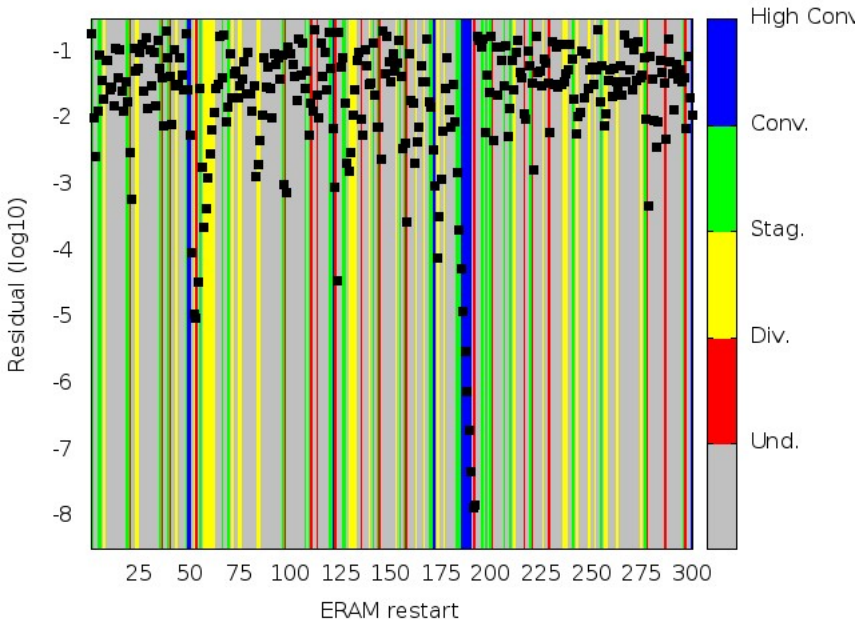

Figure 4.4: Ex11 Matrix, m=20, Algorithm 10 Results.

The ERAM has $s = \gamma = 5$, $m = 20$ and a CGSR orthogonalization scheme. The Algorithm 10 parameters are $max_{count} = 3, born_{sup} = 0.2$ and $born_{inf} = 0.8$. We used 39 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. We present on this figure the Algorithm 10 results: the black points refer to the dominant eigenpair residuals at each ERAM restart. Each colored stripe corresponds to the convergence status defined by the Algorithm 10. The gray one (undefined) means that no status could be determined at this restart. The red one (divergence) means that a divergence has been detected, the yellow one (stagnation) means that the ERAM stagnates at this restart, the green one (convergence) means that the ERAM converges at this restart and finally the blue one (high convergence) means that the ERAM highly converges at the considered restart.

The Figure 4.4 presents the Algorithm 10 results applied to the $Ex11$ matrix. The Krylov subspace size is $m = 20$. We present the $res_{CV}^{(i)}$ evolution with respect to the convergence status detected by the Algorithm 10.

As a primary observation, most of the divergence schemes are detected, which was our primary objective. The *"Undefined"* status (*"Und"* on the Figure 4.4) means that we could not affect a convergence status to the ERAM at this point.

The stagnation status could be more present, due to the residuals repartition. For the same configuration, a higher $borne_{sup}$ value allows to detect more stagnation states.

This means that with a higher $borne_{sup}$ value, there are more stagnation status detected, less undefined one. Nevertheless the stagnation intervals remains the same as the Figure 4.4, their size is simply smaller than a configuration using a higher $borne_{sup}$ value. The only difference will be about the number of restarts that remained in the

stagnation status.

We choose to favor large convergence status rather than large stagnation status, as we aim to intervene at the beginning of a stagnation status.

The Algorithm results 10 presented on figure 4.4 are satisfying as they properly identify the stagnation and divergence status, leading to a pertinent intervention to modify some ERAM parameters with respect to the convergence status. This point will be detailed later in this thesis.

We executed the same tests by using a smaller Krylov subspace size $m = 15$ (Figure 4.5 below).

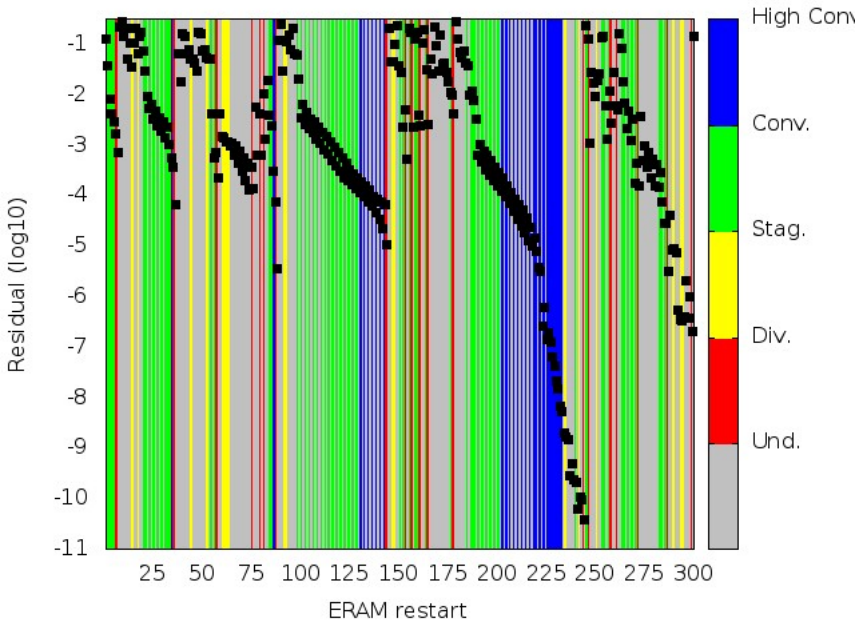

Figure 4.5: Ex11 Matrix, m=15, Algorithm 10 Results.
The ERAM has $s = \gamma = 5$, $m = 15$ and a CGSR orthogonalization scheme. The Algorithm 10 parameters are $max_{count} = 3, born_{sup} = 0.2$ and $born_{inf} = 0.8$. We used 39 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. One may note that the ERAM using m=15 has a better convergence scheme than the ERAM using m=20 above. The black points refer to the dominant eigenpair residuals at each ERAM restart. Each colored stripe corresponds to the convergence status defined by the Algorithm 10.

The input parameters of the Algorithm 10 still detects the divergence and stagnation parts, allowing us to modify some ERAM parameter at the appropriate restart.

The Algorithm 10 successfully detects the slow convergence, which prevents of abusive ERAM parameters changes. Conserve the convergence is as important as properly detect the stagnation and divergence status.

In this context, we applied the Algorithm 10 to the Ex11 matrix with a "large" subspace size: we mean that this subspace size is large enough to get a very smooth convergence. Note that the convergence on Figure 4.4 is better than the convergence

presented on Figure 4.5. This perfectly illustrates the tricky issue of m value, meaning that we can observe a better convergence with a smaller subspace size.

We applied he Algorithm 10 to the $Ex11$ matrix, using a subspace size 25 and computing 5 eigenpairs. The results are presented on the Figure 4.6 below.

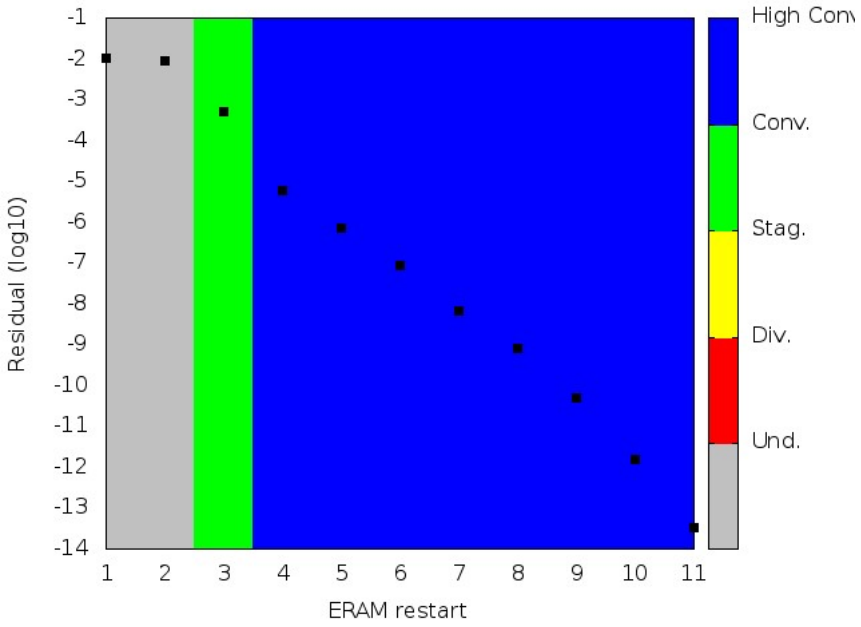

Figure 4.6: Ex11 Matrix, m=25, Algorithm 10 Results.

The ERAM has $s = \gamma = 5$, $m = 25$ and a CGSR orthogonalization scheme. The Algorithm 10 parameters are $max_{count} = 3, born_{sup} = 0.2$ and $born_{inf} = 0.8$. We used 39 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. The black points refer to the dominant eigenpair residuals at each ERAM restart. Each colored stripe corresponds to the convergence status defined by the Algorithm 10.

If the input parameters of the Algorithm 10 are wrongly chosen, some intervals of the ERAM convergence presented on the Figure 4.6 could be characterized as a stagnation. In this case, the risk is to change an (some) ERAM parameter(s) to remedy to these *"false"* stagnation, therefore we would uselessly disrupt the convergence. The Algorithm 10 detects and maintains the high convergence which is a priority for our study.

4.2.2.2 The $Mixtank\_new$ Matrix

We present some results of the Algorithm 10 with the same input parameters applied on the $Mixtank\_new$ matrix. This matrix has a convergence completely different from the $Ex11$ Matrix, we aim to show that the Algorithm 10 still successfully detect and characterize the ERAM convergence. We recall that $max_{count} = 3, born_{sup} = 0.2$ and $born_{inf} = 0.8$.

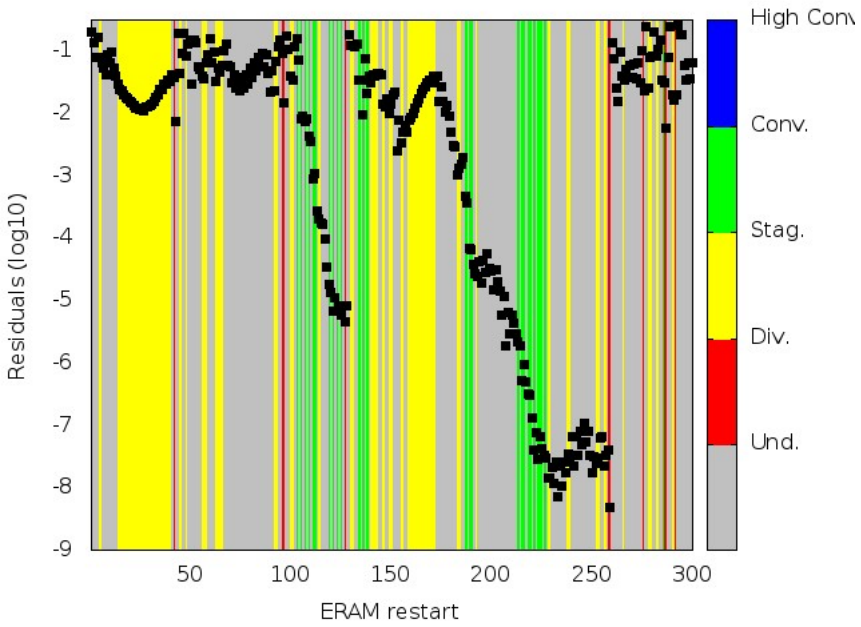

Figure 4.7: *Mixtank_new* Matrix, m=15, Algorithm 10 Results.

The ERAM has $s = \gamma = 5$, $m = 15$ and a CGSR orthogonalization scheme. The Algorithm 10 parameters are $max_{count} = 3, born_{sup} = 0.2$ and $born_{inf} = 0.8$. We used 29 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. The black points refer to the dominant eigenpair residuals at each ERAM restart. Each colored stripe corresponds to the convergence status defined by the Algorithm 10. The gray color (undefined status) means that no status could be determined at the considered restart.

The *Mixtank_new* Matrix is quite slow to converge as presented on the Figure 4.7: Nevertheless, the Algorithm 10 detects successfully the two convergence intervals as well as the diverse stagnation intervals, much more presents in this configuration. Most of the divergence status are detected due to sudden peaks.

We applied the Algorithm 10 to the *Mixtank_new* matrix, using a subspace size 25 and computing 5 eigenpairs. The results are presented on the Figure 4.8 below.

The *Mixtank_new* eigenvalues of the are still pretty slow to converge, especially until the 80^{th} restart. For this configuration, two scenarios may be considered:

- Somehow, we may consider that the ERAM only converges and does not stagnate. Nevertheless, we reached the convergence at the 120^{th} restarts, therefore this configuration is no comparable to the smooth and fast convergence observed on he Figure 4.6.

- We consider that the matrix alternates between convergence and stagnation. We bet on the fact that modifying an ERAM parameter will provide better results than keeping this configuration.

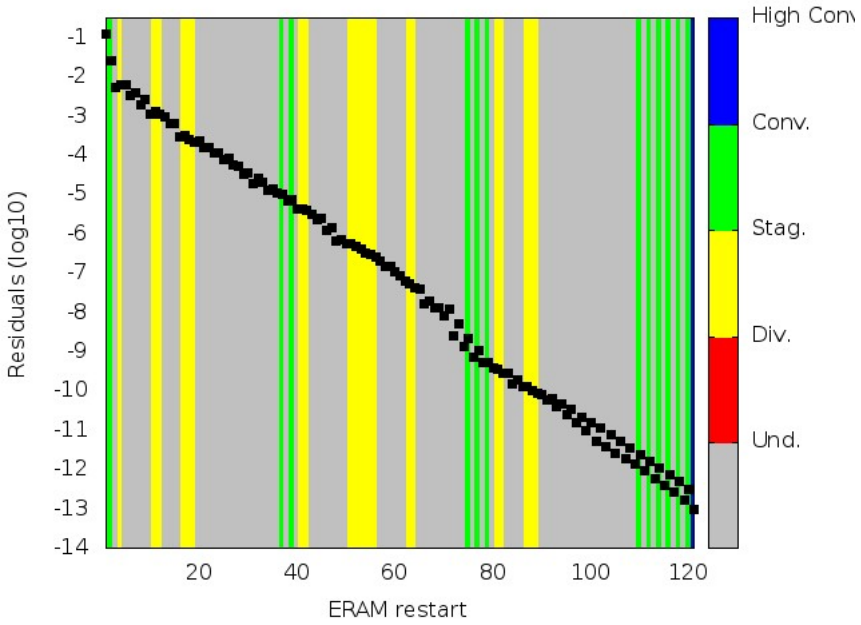

Figure 4.8: *Mixtank_new* Matrix, $m = 25$, Algorithm 10 Results.
The ERAM has $s = \gamma = 5$, $m = 25$ and a CGSR orthogonalization scheme. The Algorithm 10 parameters are $max_{count} = 3, born_{sup} = 0.2$ and $born_{inf} = 0.8$. We used 29 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. The black points refer to the dominant eigenpair residuals at each ERAM restart. Each colored stripe corresponds to the convergence status defined by the Algorithm 10. The gray color (undefined status) means that no status could be determined at the considered restart.

4.2.2.3 The *Rim* Matrix

To enhance our conclusion, we realized the same study on the *Rim* matrix.

The Rim matrix convergence presented on the Figure 4.9 is relatively *"chaotic"*, especially after the 55 restart. From the first until the 40^{th} restart, one may consider that the ERAM stagnates. However, If we look closer, the ERAM successively converge, then diverge, then converge ... The gap are too high to be considered as a stagnation and too disrupted to be considered as a convergence or a divergence scheme. Ideally, it would be characterized as a new status, that we could qualified as *"cyclic"* or *"chaotic"* convergence.

As a second observation, one may considered that the convergence status between the 55 and 96 restarts is not a stagnation but a divergence status. Somehow, this is true. However, we recall that we aim to detect **as soon as possible** the stagnation to the divergence status: Therefore we judge that it was no necessary to consider and take into account a slow divergence.

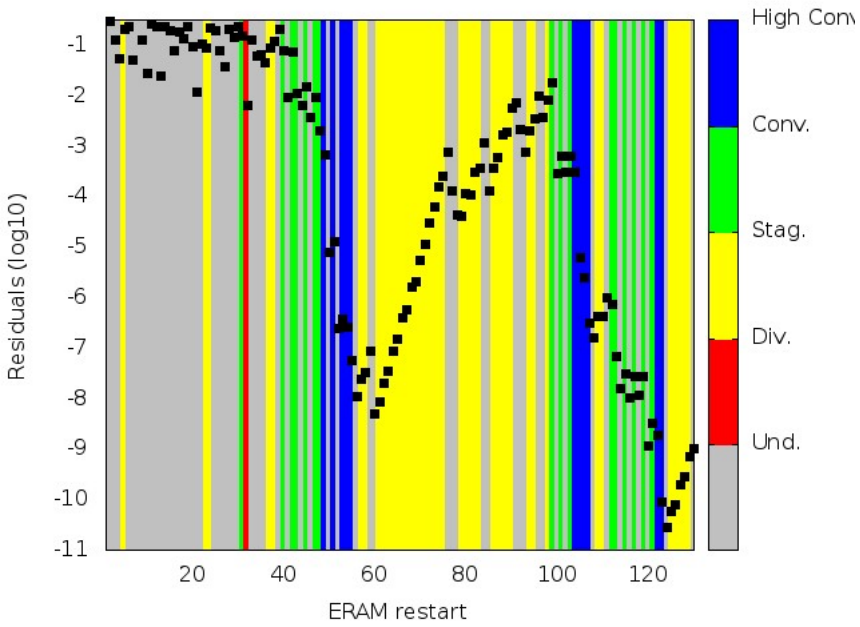

Figure 4.9: Rim Matrix, $m = 25$, Algorithm 10 Results.
The ERAM has $s = \gamma = 5$, $m = 25$ and a CGSR orthogonalization scheme. We used 120 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. The Algorithm 10 parameters are $max_{count} = 3, born_{sup} = 0.2$ and $born_{inf} = 0.8$. The black points refer to the dominant eigenpair residuals at each ERAM restart. Each colored stripe corresponds to the convergence status defined by the Algorithm 10. The gray color (undefined status) means that no status could be determined at the considered restart.

We present the Algorithm 10 results applied on the same matrix using a subspace size $m = 30$. This ERAM configuration is definitely characterized as a high and smooth convergence (like the Figure 4.6). Such convergence scheme is successfully detected by the Algorithm 10.

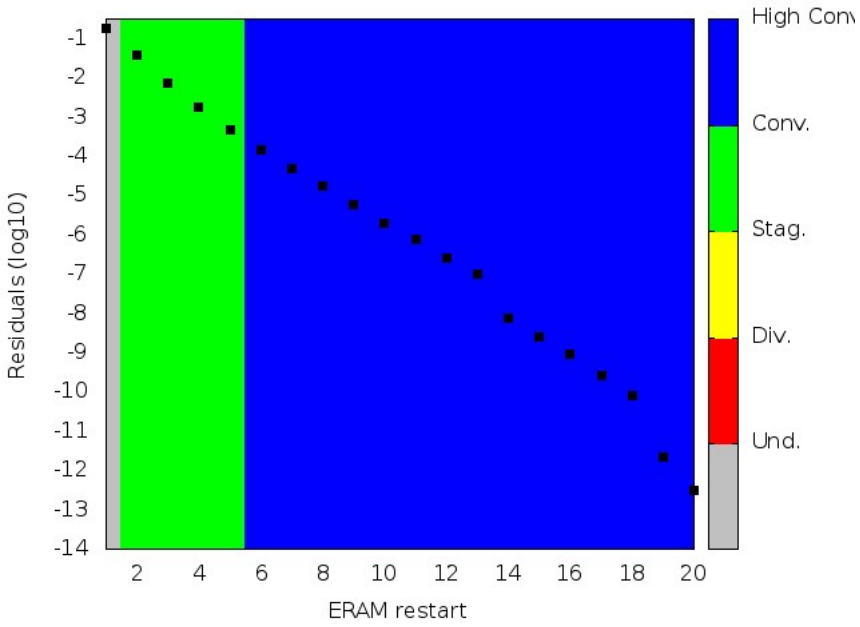

Figure 4.10: Rim Matrix, $m = 30$, Algorithm 10 Results.

The ERAM has $s = \gamma = 5$, $m = 30$ and a CGSR orthogonalization scheme. We used 120 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. The Algorithm 10 parameters are $max_{count} = 3, born_{sup} = 0.2$ and $born_{inf} = 0.8$. The black points refer to the dominant eigenpair residuals at each ERAM restart. Each colored stripe corresponds to the convergence status defined by the Algorithm 10. The gray color (undefined status) means that no status could be determined at the considered restart.

Conclusion

The study presented in this chapter consists of assessing the ERAM convergence. We exposed that the Krylov method GMRES used to solve the linear systems dispose of such method to appreciate its numerical convergence at the runtime execution.

The Krylov eigen and linear system solvers are very similar: the concept is pretty close, the nature of the system to solve just differs. Fixing the subspace size value for both ERAM and GMRES remains the trickiest part.

In the case of the GMRES solver, the convergence metric is used to optimize the subspace size value with the convergence rate. In the case of a strong convergence, the subspace size will be decreased, so as to reduce the parallel execution time per restart while maintaining the GMRES convergence at a satisfiable rate. Conversely, the opposite modification will be done so as the GMRES can reach the convergence. We presented some (ongoing) research to estimate the GMRES convergence and how the subspace size is fixed with respect to its convergence: such algorithms use the measure of iterative Krylov subspaces angles as the convergence metric.

Such scheme would be interesting to apply in the case of the ERAM solver, however, the GMRES convergence heuristics are no longer available for the ERAM one.

We presented in this chapter an adaptation of the GMRES heuristic to measure and characterize the ERAM convergence. The first step is then to determine a convergence metric to appreciate the ERAM convergence. Based on this metric, we presented an algorithm that detects the ERAM convergence behavior.

We fixed two priorities regarding the ERAM convergence detection. First, the divergence or stagnation status must be quickly detected. We aim to avoid such behavior by modifying ERAM parameters so as to reach the convergence faster. This implies the second condition, which is properly detect the convergence behavior. Indeed, if a convergence status is *"wrongly"* interpreted as a stagnation, then we may change ERAM parameters and disrupt the ERAM convergence instead of ameliorate it.

Such heuristic will be used in this thesis so as to dynamically modify some ERAM parameters with respect to its convergence.

In the next chapter, we will present two matrix generators, both starting from an imposed spectrum. In what follows, we will use matrices generated by these matrix generators only.

Chapter 5

Matrix Generators for Extreme-Scale Computing

Basically, all the scientific domains concerned by the exascale computing must have target matrices in accordance with the problem to solve and its extreme scale.

In the wide spectrum of the High Performance Computing scientific applications, the eigenvalue problem arises quite frequently.

Nevertheless, the eigenvalue problems are not the only solvers concerned by the lack of target matrices with respect to fixed mathematical properties and/or extreme-scale size. As an example, some linear systems may use the matrix eigenvalues (or a subset of its eigenvalues) to build preconditioners in order to accelerate the system convergence.

Starting from these observations, many scientific fields crave of target matrices to test both the eigensolver numerical performances and the parallel efficiency.

5.1 Extreme-Scale Eigenvalue Solvers Performances Evaluation

Many applications mentioned in the Chapter 2 require to solve an (or many) eigen problem(s) during their computation process. Based on this observation, some eigen solver (depending on their numerical/parallel characteristics) used by the simulation must be adapted to the extreme scale supercomputers and still be numerically efficient. This reasoning can be extended to the linear solver case.

As every large HPC simulation is always preceded by validation, the eigen solvers must be checked and endorsed in terms of both numerical results and parallel sturdiness. In the context of Extreme-Scale computing, two objectives must be reached:

- The exactness of the eigen solver: does the computed eigenvalues $\theta_j, \forall j \in [1, s]_{\mathbb{N}}$ correspond to the exact eigenvalues $\lambda_j, \forall j \in [1, s]_{\mathbb{N}}$[1]?
- The scalability of the eigen solver: does the solver verify the extreme-scale scalability constraints?

In order to provide a numerically robust eigen solver, we need to dispose of matrices whose spectrum is known to conclude on the numerical efficiency. Having an extreme-scale

[1] s is the number of desired eigenpairs. In this specific case, $s \in [1, n]_{\mathbb{N}}$ where $A \in \mathbb{C}^{n \times n}$

eigen solver implies to dispose of large input matrices, in accordance with the extreme-scale systems size. As a conclusion, we need target matrices with very large dimension and a known spectrum.

5.1.1 The Existing Collections

We can cite two famous and widely spread matrices providers, the Tim Davis [Davis] and the Matrix Market [Mat] collections.

They both contain many matrices coming from many scientific fields. As an non-exhaustive list, we can cite the undirected (random/weighted) (multi-)graph, optimization problem, combinatorial problem, circuit simulation problem, computational fluid dynamics problem, linear programming, structural problem, electro-magnetics problem, model reduction problem et caetera. If both collections provide many matrices with many mathematical properties, we still have issues to get matrices that match with our requirements:

- *Hypothesis 1*: The spectrum of the matrix must be known and moreover controlled: we can fix specific spectrum properties in order to evaluate numerical robustness of various algorithms,
- *Hypothesis 2*: The matrix shall not be Hermitian and shall not look like a trivial matrix,
- *Hypothesis 3*: The matrix may have very high dimension (this includes the non-zeros elements and/or the matrix size),
- *Hypothesis 4*: The matrix must not be confidential and can be shared with the scientific community.

It is burdensome to get matrices to evaluate the parallel performances and the behavior of the numerical algorithms (linear and eigen solvers) at the extreme-scale.

In this chapter, we will present two matrices generators, focus on their benefits and drawbacks regarding our extreme-scale matrices hypothesis.

5.1.2 The Spectrum as the Key Parameter

We address the following problem : Let's consider a matrix $A \in \mathbb{C}^{n \times n}, n \in \mathbb{N}^*$. We denote by $Spec(A)$ the spectrum of A defined by:

$$\text{Spec(A)}=\{\lambda \in \mathbb{K}, \exists x \in \mathbb{K}^n \setminus \{0\} \text{ such that } Ax = \lambda x\}, \tag{5.1}$$

Starting from a fixed spectrum $Spec_{in} \in \mathbb{C}^n$, we aim to create a matrix $A_{gen} \in \mathbb{C}^{n \times n}$ such that $Spec(A_{gen}) = Spec_{in}$ where A_{gen} refers to the matrix resulting from the matrix generator. During all the following chapters, we will keep these notations.

Our matrices generators use the input spectrum as the leading characteristic to build the algorithm. Starting from an imposed spectrum itself to generate a matrix allows to skirt many problems that we raised earlier:

- Some eigen solvers are more or less sensitive to "particular" spectrum. As an illustration, a scientific community may be interested in matrices with clustered eigenvalues or conjugated eigenvalues. Some others could be interested in a random distribution of the eigenvalues or eigenvalues contained in a specified interval. This list of examples is of course non exhaustive.

  Fixing the spectrum may allow many scientific communities to numerically check their algorithms according to a specific spectrum scheme and remaining close to the initial problem.

- Many industries, laboratories and academic centers are confronted to the confidentiality locks of their results. Matrices issued from real applications (as an illustration, neutronic applications) are confidential most of the time. It is out of question to let these matrices being shared on matrix collection websites.

  Generate a *"fake"* matrix, with the same spectrum but different shape (dense instead of sparse, or different sparsity schemes) erases the confidentiality lock. Starting from the spectrum allows to experiment the algorithms with no confidentiality violations while remaining close from the original application field.

- Choosing the spectrum implies choosing the matrix size. In the context of the exascale computing, very-large matrices are required (in terms of dimension and/or number of non-zeros elements). Especially, choosing the size and the spectrum allows to execute scalability tests and ensure the numerical accuracy of the results. We illustrate our proposal with an example: If we aim to study the ERAM scalability, we will execute both strong and weak scaling tests. Let's consider A whose size is $n = 2^{10}$.

  In the case of the strong scaling, we will solve the eigenvalue problem on A using successively 2^k tasks, $k \in [0, 10]_{\mathbb{N}}$. In this case, the same matrix will be used for all parallel executions.

  In the case of the weak scaling, the problem size per tasks must remain the same. In this context, we need to get A of size $n = 2^k, \forall k \in [0, 10]_{\mathbb{N}}$. The current matrices collections do not necessarily provide such matrices, having the same eigenvalues but different sizes. This is of course theoretically impossible as spectrum size is the matrix size, therefore, for $n_1 \neq n_2$, the associated spectrum of A^{n_2} and A^{n_1} will not remain the same. As the ERAM computes the dominant eigenpairs, we will consider this particularity to build an extended spectrum. If we consider $n_1 > n_2$, we start from A^{n_2} spectrum and add some eigenpairs whose values are close to the smallest eigenpairs of A^{n_2}. This aims to extend A^{n_2} spectrum until n_1 size, without disrupting the original spectrum (or at least, limit their influence). The opposite method could be used to preserve the smallest eigenvalues.

5.1.3 The Matrix Generator Computational Hypothesis

One major component has not been clearly identified in the previous matrix generator hypothesis. The hypothesis *"The matrix generator must conserve $Spec_{in}$"* may be divided in two hypothesis:

- $Spec_{in}$ must be mathematically conserved,
- The computer arithmetic must be controllable.

The first hypothesis depends on the method itself as we may find an algorithm that mathematically checks the exact equality of $Spec_{in}$ and $Spec(A_{gen})$.

Regarding the second hypothesis, we must consider that the floating point operations and rounding will induce that $Spec_{in} \neq Spec(A_{gen})$. This point will be detailed later in this chapter.

In the context of the *Framework Programming for Post-Petascale Computing (FP3C)* Project [2], two matrix generators matching with the hypothesis listed above have been designed by Hervé Galicher in CEA Saclay (DEN/DANS/DM2S). More Informations on the matrices generators ca be found in [Galicher 2014].

These two matrix generators have been widely used during this thesis. I contributed to the improvement in terms of parallelism performances of the methods presented bellow, their numerical validation and parallel performances.

Secondly, I contributed to their validation and use to check the numerical efficiency of our ERAM results. As a major contribution, I could identify some numerical trends of the matrices generators thanks to the ERAM results that will be presented in the following chapter.

These results justify the *Hypothesis 4 :The matrix must not be confidential and can be shared with the scientific community.* We will detail later how the matrix generators could provide target matrices whose behavior remain close to the initial problem and validate their use, efficiency and great interest for the all HPC community.

5.2 The Dense Matrix Generator

This method is radically different from the diagonal-band matrix generator, it is based on orthogonal permutations and Bartlett's formula.

We consider the system matrix $A \in \mathcal{M}_n(\mathbb{K})$ and divide it into blocks such that:

[2] The FP3C project is a French-Japanese project started in 2010 funded by ANR and JST. During four years, French and Japanese researcher have shared their knowledge and collaborated altogether on the software technologies, languages and programming models to explore extreme performance computing beyond petascale computing. This project gathered the CNRS (IRIT, PRISM), INRIA (Bordeaux, Rennes, Saclay) and CEA/DEN (Saclay) on French side and the Kyoto University, Tokyo Institute of Technology, University of Tokyo and University of Tsukuba on the Japanese side. More informations can be found in http://jfli.nii.ac.jp

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}$$

This block shape can be used efficiently to compute A^{-1}. To simplify the notations, we denote by A_{block} the matrix resulting of $(A_{2,2} - A_{2,1}{A_{1,1}}^{-1}A_{1,2})^{-1}$. It is assumed that computing A^{-1} is equivalent to compute ([Bierens 2013] and [Diciunas 1998]):

$$\begin{pmatrix} {A_{1,1}}^{-1}(I_d + A_{1,2}A_{block}A_{2,1}{A_{1,1}}^{-1}) & -{A_{1,1}}^{-1}A_{1,2}A_{block} \\ -A_{blocks}A_{2,1}{A_{1,1}}^{-1} & A_{block} \end{pmatrix}$$

As a consequence, computing A^{-1} requires to compute $A_{1,1}$ and $(A_{2,2} - A_{2,1}{A_{1,1}}^{-1}A_{1,2})$ inverts. To simplify the problem, we impose $A_{1,1}$ orthogonal. Then, we must focus on computing $(A_{2,2} - A_{2,1}{A_{1,1}}^{-1}A_{1,2})^{-1}$. We define by k the size of $A_{1,1}$ and by l the size of $A_{2,2}$.

According to [Diciunas 1998] computing $A_{2,2} - A_{2,1}{A_{1,1}}^{-1}A_{1,2}$ remains the same as computing :

$$A_{2,2} - A_{2,1}{A_{1,1}}^{-1}A_{1,2} = [\cdots[[A_{2,2} + \lambda_1 C_1 R_1] + \lambda_2 C_2 R_2] + \cdots + \lambda_{k^2} C_{k^2} R_{k^2}] \tag{5.2}$$

Each block has a form such as $B_j + \lambda_j C_j R_j, j \in [1, k^2]_{\mathbb{N}}$. Computing the invert of $(A_{2,2} - A_{2,1}{A_{1,1}}^{-1}A_{1,2})$ requires to compute the successive inverts of $B_j + \lambda_j C_j R_j$. The Bartlett's formula [Bartlett 1951] matrix provide us a simplified version of $B_j + \lambda_j C_j R_j$ invert (more details can be found in [Diciunas 1998]):

$$(B_j + \lambda_j C_j . R_j)^{-1} = {B_j}^{-1} - \frac{1}{\frac{1}{\lambda_j} + R_j {B_j}^{-1} C_j} {B_j}^{-1} C_j R_j {B_j}^{-1} \tag{5.3}$$

It results from 5.3 that computing $B_j + \lambda_j C_j R_j, j \in [1, k^2]_{\mathbb{N}}$ invert requires to compute only B_j invert. Finally, computing $(A_{2,2} - A_{2,1}{A_{1,1}}^{-1}A_{1,2})^{-1}$ implies to compute the successive B_j and $A_{2,2}$ inverts. We impose the orthogonality property to $A_{2,2}$ matrix to simplify the inverts computations.

The dense matrix generator method imposes to build a matrix such that:

- $A_{1,1}$ and $A_{2,2}$ are orthogonal matrices,
- $A_{1,2}$ and $A_{2,1}$ contains random values.

We then apply k^2 Bartlett's formula using the chosen eigenvalues to build the final dense matrix. We present the associated Algorithm 11:

Algorithm 11 The Dense Matrix Generator

Input: $Spec_{in} \in \mathbb{C}^n$, $\varepsilon \in]0,1]$, $nnul \in [1,n]$

$nnz = n - nnul$
for $i = 1, n$ **do**
$A_p[i] = i$
$colRow_p[i] = i$
end for
make a random n-permutation of A_p and $colRow_p$
Insert nnz random elements in $\{col, row\} \in \{\mathbb{C}^n\}^2$
for $k = 1, n$ **do**
if $colRow_p[k] \geq nnz$ **then**
$ind_{row} = 0$
else
$ind_{row} = row[colRow_p[k]]$
end if
if $colRow_p[A_p[k]] \geq nnz$ **then**
$ind_{col} = 0$
else
$ind_{col} = col[colRow_p[A_p[k]]]$
end if
$RinvAC = RinvAC + ind_{row} \times ind_{col}$
$RinvADC = RinvADC + Spec_{in}[A_p[k]] \times ind_{row} \times ind_{col}$
end for
$\lambda = \varepsilon - RinvAC$
for $i = 1, n$ **do**
$A_{out}[i,i] = Spec_{in}[A_p[i]]$
for $j = 1, n$ **do**
if $colRow_p[k] \geq nnz$ **then**
$ind_{row} = 0$
else
$ind_{row} = row[colRow_p[k]]$
end if
if $colRow_p[A_p[k]] \geq nnz$ **then**
$ind_{col} = 0$
else
$ind_{col} = col[colRow_p[A_p[k]]]$
end if
$element = (ind_{row} \times ind_{col})(\frac{1}{\lambda} \times Spec_{in}[A_p[i]] - \frac{1}{\varepsilon} \times Spec_{in}[A_p[j]] - \frac{RinvADC}{\lambda \times \varepsilon})$
$A_{out}[i,j] = A_{out}[i,j] + element$
end for
end for

Output: $A_{out} \in \mathbb{C}^{n \times n}$, such that $Spec(A_{out}) \sim Spec_{in}$

5.2.1 The Dense Matrix Generator Results

The current implementation is sequential, due to the poor parallelism potential of the method. We summarized the target matrices to test the dense matrix generator accuracy in the Table 5.2.1. We add a new matrix to this table, issued from CEA neutronic applications:

Name	Size	Field	Source
Fission	10,000	neutronic	CEA applications
Ex11	16614	computational fluid dynamics problem	UF SMC[Davis]
Mixtank_new	29,957	computational fluid dynamics problem	UF SMC[Davis]
Rim	22,560	computational fluid dynamics problem	UF SMC[Davis]

Table 5.1: Target Matrices to Test the Dense Matrix Generator

For each target matrices, we dispose of the original spectrum. We will use these spectrums to obtain A_{gen} that will correspond to the *"dense"* version for each presented matrices above.

The poor parallelization potential of the Algorithm 11 does not match with the hypothesis *"The algorithm must be tractable in very high dimension"*. Theoretically, there is no limit regarding the A_{gen} size. However, the sequentiality of the dense matrix generator implementation limits the memory size of A_{gen}, therefore its dimension. The A_{gen} matrices obtained from the Algorithm 11 are interesting for the eigenvalue problems, this is why we present the following results.

5.2.1.1 The Fission Matrix

The Fission matrix is issued from CEA neutronic simulations. Originally, this matrix is very sparse. As we dispose of its complete spectrum, we will generate from it two dense matrices:

- *Dense Fission*: Obtained by applying the Algorithm 11 with $Spec(Fission)$ as input parameter.
- *Dense Fission*$_{2^p}$: We aim to extend the Fission matrix size until $2^p, p \in \mathbb{N}$. Due to the memory limits, we could generate *Dense Fission*$_{2^{15}}$. Increasing the matrix size implies to increase the spectrum size therefore add eigenvalues to the spectrum. We choose to add 6384 eigenvalues that we will denote by $\delta_j, j \in [1, 6384]_{\mathbb{N}}$ to distinct them from the original spectrum. We choose δ_j such that $\forall j \in [1, 6384]_{\mathbb{N}}, \lambda_{8,000} \geq$

$\delta_j \geq \lambda_{10,000}$. We disrupt the lowest part of the spectrum as the Restarted Arnoldi Method favors the convergence to the dominant eigenpairs.

On the following figures, we present the spectrum of Fission and $\text{Fission}_{2^{15}}$ matrices (real modulus eigenvalues) and the respective distance between each real modulus eigenvalues.

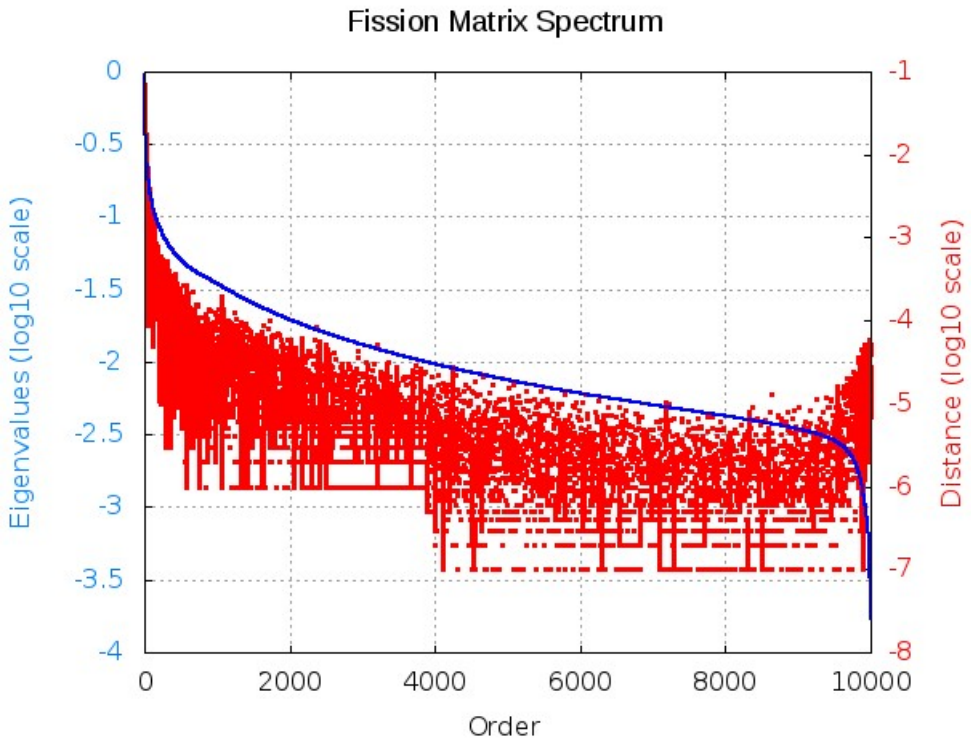

Figure 5.1: The Fission Matrix Spectrum, Size is $10,000$

This figure shows the original Fission matrix real modulus eigenvalues (blue line) without any modification. The red points present the distance between each successive real modulus eigenvalue. The Eigenvalues and Distance metrics use both log_{10} scale to enhance the eigenvalues distribution visibility.

The generated matrix resulting from the Algorithm 11 is *Dense Fission* with $99,980,003$ non-zeros elements. We could generate *Dense Fission* on a fat node (4 octo-cores processors per node) of the PRACE CURIE supercomputer. The matrix generation (including the Input/Output operations) execution time is about 35 minutes.

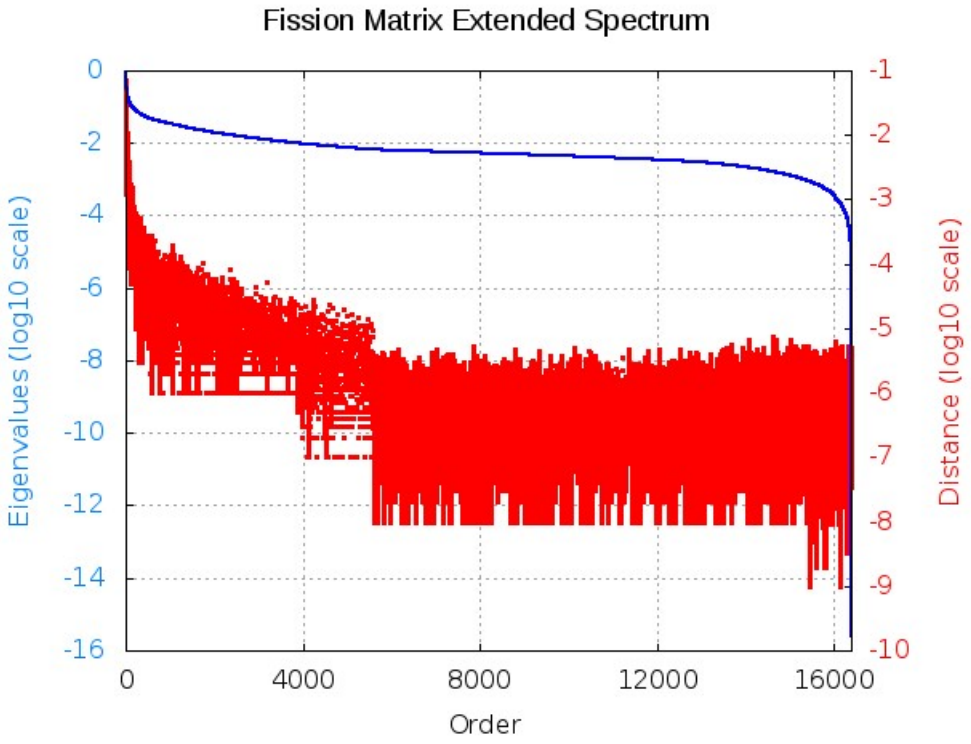

Figure 5.2: The $\text{Fission}_{2^{15}}$ Matrix Spectrum, Size is 2^{15}

The presented spectrum is an extension of the original Fission matrix spectrum. We added 6384 eigenvalues, starting from the 8000^{th} eigenvalue so as we do not disrupt the dominant part of the spectrum. The added eigenvalues are chosen with respect to the lowest part of the original spectrum. The blue line represents the real modulus of eigenvalues while the red points represent the distance between each real modulus of eigenvalues (this aims to show the clustered eigenvalues if they exist). The Eigenvalues and Distance metrics use both log_{10} scale to enhance the eigenvalues distribution visibility.

The generated matrix resulting from the Algorithm 11 is $\textit{Dense Fission}_{2^{15}}$. This matrix has $286,402,691$ non-zeros elements. We could generate this matrix on a fat node of the PRACE CURIE supercomputer. The matrix generation (including the Input/Output operations) execution time is about 4h37.

The next step is to compute the dominant (largest real modulus) eigenvalues and compare the results with the spectrums presented above. We used the SLEPc Arnoldi and Krylov-Schur method. For each solver, we will compare the relative error (blue line) of the obtained eigenvalues and present the associated computed residuals (red line) resulting from the SLEPc eigen solvers.

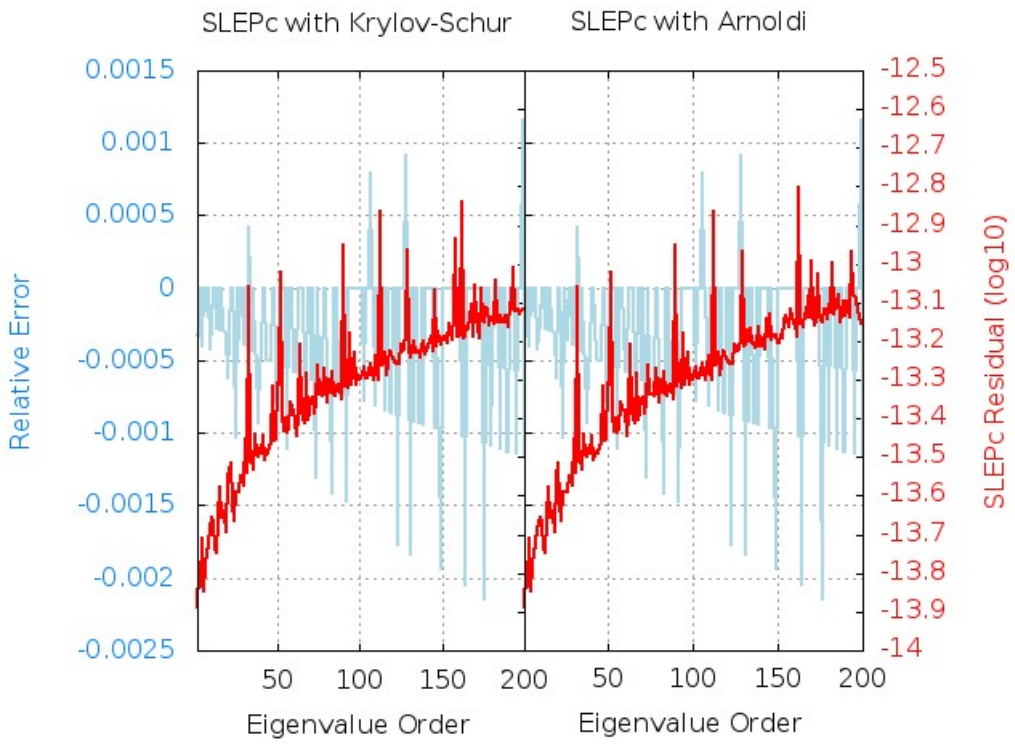

Figure 5.3: Dense Fission Matrix Spectrum, SLEPc Eigen Solvers Numerical Comparison
The blue line refers to the residual error while the red line refers to the SLEPc computed eigenpairs residual. The SLEPc eigen solvers have been executed using 50 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. We fixed $s = 200$ and $m = 400$ (the default SLEPc value). The SLEPc Arnoldi (respectively Krylov-Schur) eigen solver stopped after 5 (respectively 3) restarts.

The Figure 5.3 shows the relative error between $Spec(Fission)$ and $Spec(Dense\ Fission)$ (blue line) for the 200 largest real modulus eigenvalues. We computed the *Dense Fission* eigenvalues using the Arnoldi and Krylov-Schur eigen solver of the SLEPc library. On the red line, we present the residual of the 200 largest real modulus eigenvalues obtained with SLEPc library. Both SLEPc eigen solver use the Rayleigh-Ritz method to extract the eigenvalues ([Hernandez 2006] and [Hernandez 2007a]).

The relative error regarding the computed spectrum is very satisfying, as we expected. The SLEPc eigen solvers may compute at maximum 20% of the spectrum (id est 2000 dominant eigenvalues). As we are interested in the dominant part of the spectrum, we choose to compute 2% of the spectrum. The precision is similar for both eigen solvers. The SLEPc residuals order shows that *Dense Fission* computed eigenvalues are relatively accurate, leading to the fact that the relative error will provide a good metric to evaluate the spectrum conservation through the Algorithm 11. Based on the Figure 5.3, we conclude on the spectrum conservation, *Dense Fission* sprectum is comparable to $Spec(Fission)$.

We executed the same tests with the *Dense Fission*$_{2^{15}}$ spectrum, with exactly the same parameters as mentioned above excepted for the number of MPI tasks. The following tests have been executed on 64 MPI tasks, using the thin nodes of the PRACE Curie supercomputer.

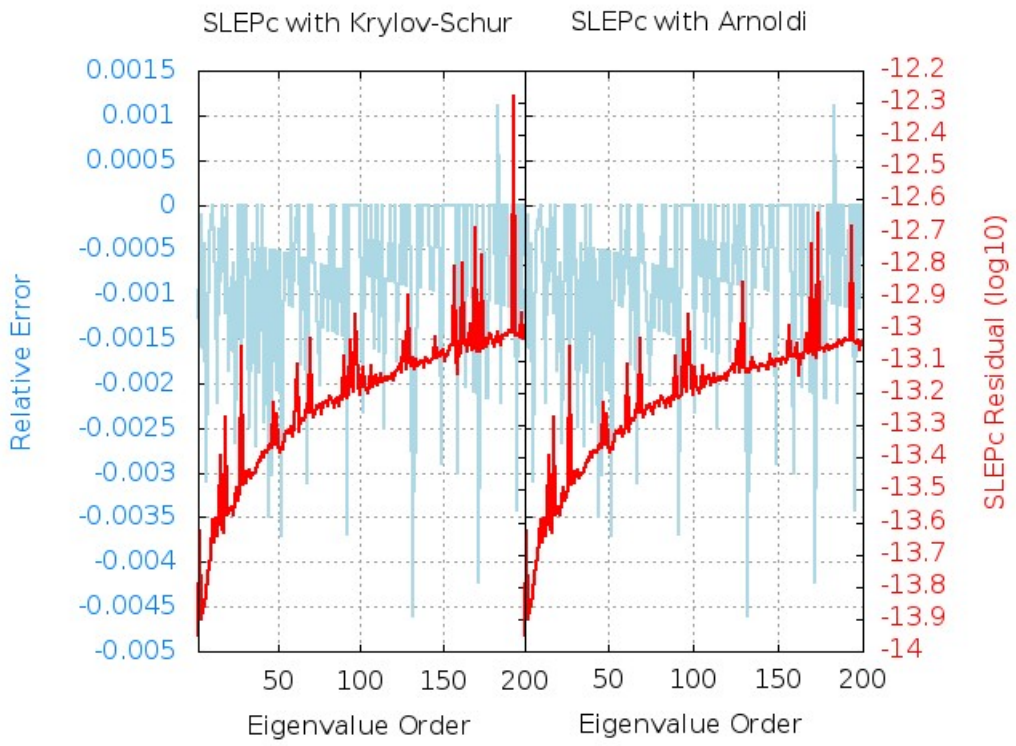

Figure 5.4: Dense Fission$_{2^{15}}$ Matrix Extended Spectrum, SLEPc Eigen Solvers Numerical Comparison

The blue line refers to the residual error while the red line refers to the SLEPc computed eigenpairs residual. The SLEPc eigen solvers have been executed using 64 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. The SLEPc Arnoldi (respectively Krylov-Schur) eigen solver stopped after 6 (respectively 3) restarts.

The Figure 5.4 shows that the 200 dominant eigenvalues have been conserved by the algorithm, as the relative error (blue line) and the residuals (red line) are comparable to the previous results. Both solvers provide accurate residuals, leading to the same conclusion as the *Dense Fission* matrix: the relative error of *Dense Fission*$_{2^{15}}$ spectrum shows the spectrum conservation through the Algorithm 11.

Obviously, the dominant part of the spectrum of *Dense Fission*$_{2^{15}}$ matrix is not disrupted by the injection of *"fake"* eigenvalues ot the lowest part of the spectrum.

The same tests have been executed on the target matrices listed in the Table 5.2.1. As they lead to the same conclusion, we present all these results in the Appendix A. We invite the reader to consult this appendix for more results.

5.3 The Diagonal-Band Matrices Generator

Starting from a spectrum, this algorithm generates A_{gen} with a diagonal-band shape. The diagonal-band matrix generator method is based on the Ordinary Differential Equations and the nilpotent semi-group generators. The method is premised on the following theorem:

Theorem 1 *:*
Let's consider the matrices $A \in \mathbb{C}^{n\times n}, M \in \mathbb{C}^{n\times n}, M_0 \in \mathbb{C}^{n\times n}, n \in \mathbb{N}^*$. *If* M *verifies :*

$$\begin{cases} \frac{dM(t)}{dt} & = AM(t) - M(t)A, \\ M(t=0) & = M_0, \end{cases}$$

Then the matrices M_t *and* M_0 *are similar,* $\forall A \in \mathbb{C}^{n\times n}$.

The idea is to impose the desired spectrum to M_0 and obtain a M_t matrix that verifies the Theorem 1 and our hypothesis. The M_t spectrum is the same as M_0 however we recall that the M_t eigenvectors are not the same as M_0. The idea may seem very simple but many parameters need to be fixed to achieve our objective.

Firstly, we define the linear operator $\tilde{A}$ such that:

$$\begin{array}{rccl} \tilde{A}_A: & \mathcal{M}_{n\times n} & \rightarrow & \mathcal{M}_{n\times n} \\ & M & \rightarrow & A.M - M.A, \end{array} \tag{5.4}$$

At the present time, we did not imposed any conditions on the matrix A. The $\tilde{A}$ operator verifies that:

$$\tilde{A}_A(I_d) = 0, \forall A \in \mathcal{M}_{n\times n}, \tag{5.5}$$

Based on the Theorem 1 and the linear operator $\tilde{A}_A$ definition, we can rewritte the ordinary differential equation such that:

$$\begin{cases} \frac{dM(t)}{dt} & = \tilde{A}_A(M(t)), \\ M(t=0) & = M_0, \end{cases} \tag{5.6}$$

Starting out the equation 5.6, we apply the exponential operator (which is possible as $\tilde{A}_A$ has no time dependency). This leads to the fact that the solution of the equation 5.6 can be expressed as follows:

$$\begin{cases} M(t) & = e^{(\tilde{A}_A t)}(M_0), \\ M(t) & = \sum_{n=0}^{\infty} \frac{t^n}{n!}(\tilde{A}_A)^n(M_0), \end{cases} \tag{5.7}$$

The equation 5.7 does not allow to compute the M_t matrix, as its sum is infinite. We add the hypothesis that $\tilde{A}_A$ is nilpotent to obtain a finite sum: If $\tilde{A}_A$ is nilpotent, then $\exists p \in \mathbb{N}^*$ such that $\tilde{A}^p = 0$. Therefore, the equation 5.7 can be rewritten as:

$$M(t) = M_0 + t\tilde{A}_A(M_0) + \frac{t^2}{2!}(\tilde{A}_A)^2(M_0) + ... + \frac{t^{p-1}}{(p-1)!}(\tilde{A}_A)^{p-1}(M_0), \tag{5.8}$$

The equation 5.8 exact arithmetic can be conserved if computed with integers values **and if the integers divisions by the successive (k)! are avoided**. The exact arithmetic version of the equation 5.8 is then:

$$(p-1)!M(t) \quad = (p-1)! \sum_{k=0}^{p-1} \frac{t^k}{(k)!} (\tilde{A}_A)^k (M_0), \tag{5.9}$$

The equation 5.8 provides a satisfiable form of M_t but we still need to define the A matrix associated to $\tilde{A}_A$ to compute the final matrix M_t. The $\tilde{A}_A$ nilpotency degree is related to A one's. We denote by $d \in \mathbb{N}^*$ the A nipoltency degree then $p = 2d$. As A is nilpotent of degree d, $\exists B \in \mathcal{M}_{n\times n}$ an invertible matrix such that :

$$A = B^{-1} \begin{pmatrix} P_1 & \cdots & \cdots & \cdots \\ \cdots & P_2 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & P_k \end{pmatrix} B \tag{5.10}$$

where $P_k \in \mathbb{R}^{k\times k}, k \in \mathbb{N}$ has the following form:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \cdots & \cdots & 1 \\ 0 & 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix}$$

We can easily express the A nilpotency degree as $d = \max_{i\in[1,k]_{\mathbb{N}}} (size(P_i))$.

Fixing $B = Id_n$ avoids to compute B^{-1} and simplifies the A form. Considering this hypothesis leads to a simple formula of the equation 5.9 to compute M_t with the input spectrum.

This method verifies most of the hypothesis fixed for the matrix generator:

- *Hypothesis 1*: $Spec_{in}$ is mathematically conserved by the method.
- *Hypothesis 2*: $A_{gen} = M_t$ is non-Hermitian.
- *Hypothesis 3*: $A_{gen} = M_t$ size and/or number of non-zeros elements must not be limited by the algorithm. This implicitly means that the algorithm must be parallelizable. This hypothesis is partially verified: The parallelism of the method is undeniable, leading to the possibility to fix very large dimension for A_{gen}. However, the number of non-zeros elements is limited by the number of diagonals we can add.

- *Hypothesis 4*: $A_{gen} = M_t$ shape is diagonal-band and the final matrix M_t depends on M_0 matrix. Executing the algorithm with M_0 matrix whose elements are significantly different from the original matrix ensures that M_t will have a different shape and elements from the confidential matrix.

Our current version and implementation of the diagonal-band matrix generator ensures the exact arithmetic computation by using integers and the equation 5.9 formula, leading to the point that M_t has the exact spectrum $Spec_{in}$.

We present the diagonal-band matrix generator Algorithm 12, taking into account the specificity of integer values. The following Algorithm 12 is available for $M_0 \in \mathcal{M}_n(\mathbb{N})$ and can be extended to the complex matrices by executing the algorithm with $\Re(M_t)$ and $\Im(M_t)$ separately (due to the $\tilde{A}_A$ linear property).

Algorithm 12 The Diagonal-Band Matrix Generator

Input: $Spec_{in} \in \mathbb{N}^n$

Randomly choose $k \in [0, n-1]_{\mathbb{N}}$
Insert random elements in k lower diagonals of $M_0 \in \mathbb{N}^{n \times n}$
$diag(M_0) = Spec_{in}$
Randomly insert 1 on $A \in \mathbb{N}^{n \times n}$ sub-diagonal
Find the nilpotency degree d of A
for $i = 0, 2d-2$ **do**
  $M_{i+1} = M_i + (\tilde{A}_A)^k(M_0)$
end for
$M_t = \frac{1}{(2d-2)!} M_{2d-2}$

Output: $M_t \in \mathbb{N}^{n \times n}$, such that $Spec(M_t) = Spec_{in}$

5.3.1 Diagonal-Band Matrix Generator Results

We computed 5 spectrums with different size (from 2^{15} until 2^{19}) and different integers values. We then generated 3 matrices for each spectrum and computed 200 eigenvalues of the generated matrices that we denote by A_{gen}.

For each generated matrix, we will compare $Spec(A_{gen})$ with respect to $Spec_{in}$. We will detail our approach with the spectrum $Spec_{2^{15}}$. The same tests have been executed with higher dimensions and lead to the same conclusion. These results will be summarized in what follows. We present on the Figure 5.5 the input spectrum $Spec_{2^{15}}$. The blue points represent the real modulus of the eigenvalues, the red bars represent the distance between each successive real modulus eigenvalue. Such metric is important as clustered eigenvalues as an example deeply influence the ERAM convergence.

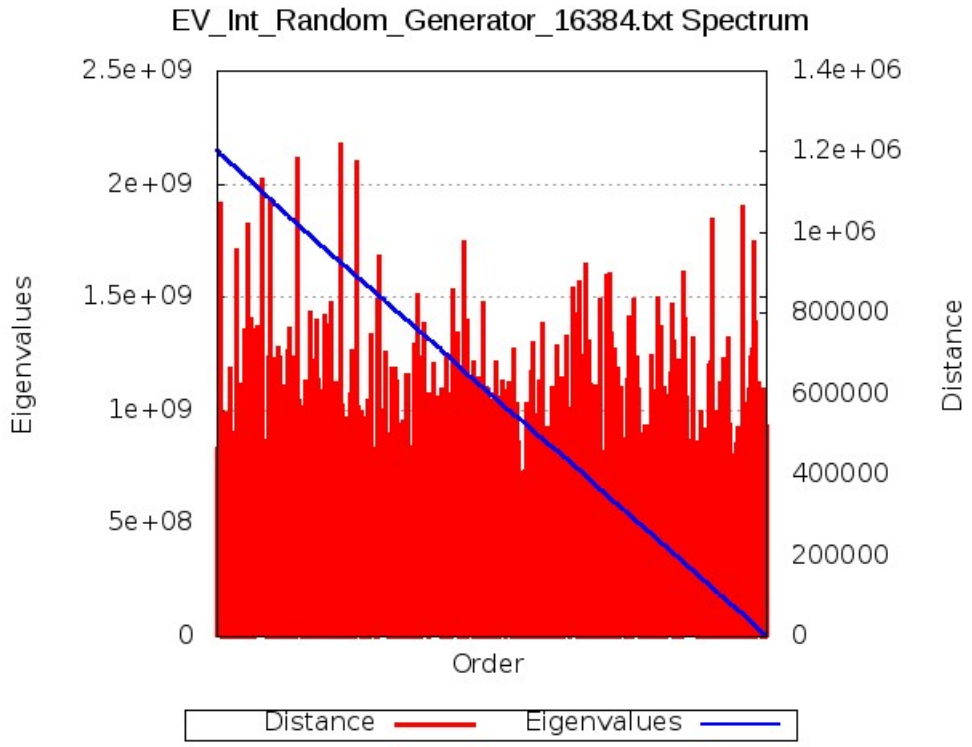

Figure 5.5: Spectrum $Spec_{2^{15}}$

$Spec_{2^{15}}$ contains exclusively integer eigenvalues to preserve the exact arithmetic of the Algorithm 12. Its size is 2^{15}, its highest eigenvalue is $2,147,328,638$ and its lowest is $8,893$. This spectrum has been randomly generated.

We executed the Algorithm 12 with 1, 2 and 4 MPI tasks respectively, using the same input $Spec_{2^{15}}$ presented on the Figure 5.5. We summarize in the Table 5.2 some characteristics of M_t and M_0 matrices for each execution of the Algorithm 12. In what follows, the matrix ${M_t}^{np}$ (respectively ${M_0}^{np}$) refers to the output matrix of the Algorithm 12 using $Spec_{2^{15}}$ as input and executed with np MPI tasks.

$Spec_{2^{15}}$	nnz	max(nnz per row)	$\max_{(i,j)\in[1,2^{15}]_{\mathbb{N}}^2} m_{i,j}$	$\min_{(i,j)\in[1,2^{15}]_{\mathbb{N}}^2} m_{i,j}$	d	MPI
${M_0}^1$	180,169	11	7,792,226,161,574,400	0	5	1
${M_t}^1$	181,435	16	7,792,226,150,688,000	−1,422,738,172,800	5	1
${M_0}^2$	114,667	7	1,546,076,619,360	0	3	2
${M_t}^2$	115,770	10	1,546,076,619,360	−180,652,320	3	2
${M_0}^4$	180,169	11	7,792,226,161,574,400	0	5	4
${M_t}^4$	181,403	16	7,792,226,161,574,400	−1,354,407,868,800	5	4

Table 5.2:

For each execution of the Algorithm 12 on 1, 2 and 4 MPI tasks respectively, we summarized some properties of M_0 and M_t matrices. All matrices presented in this table verify that their spectrum is $Spec_{2^{15}}$ presented on the Figure 5.5. We used the fat nodes of PRACE CURIE Supercomputer (4 octo-cores processors per node).

Based on the Table 5.2 results, the Algorithm 12 does not verify the reproducibility even though this characteristic remains important in the context of exascale computing.

We compute for ${M_t}^1, {M_t}^2$, and ${M_t}^4$ presented in the Table 5.2 the 200 dominant eigenvalues and their associated eigenvectors (1,22% of the complete spectrum).

We use the Scientific Library SLEPc [SLE], with respectively the Arnoldi [Hernandez 2006] and the Krylov-Schur [Hernandez 2007a] eigen solvers. We choose to compute a subset of the dominant spectrum only to evaluate the spectrum conservation accuracy. We recall that in our study, we are interested only in the dominant eigenvalues.

For each matrices ${M_t}^1, {M_t}^2$, and ${M_t}^4$, we will present the eigenvalues relative error (blue line) and the residuals associated to the computed eigenvalues (red line).

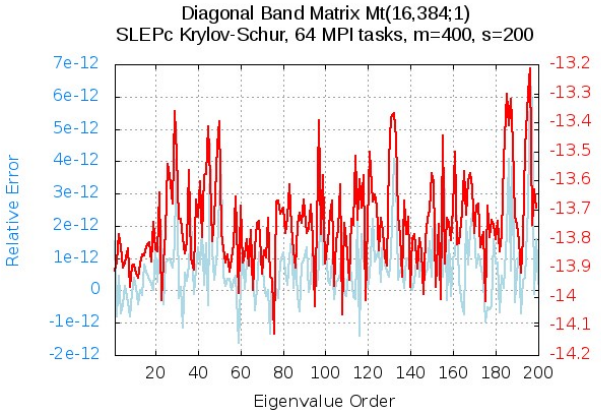

Figure 5.6: $Spec_{2^{15}}$ $M_t{}^1$, SLEPc Krylov-Schur

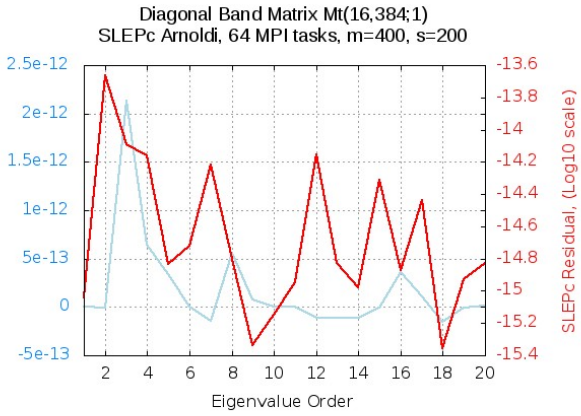

Figure 5.7: $Spec_{2^{15}}$ $M_t{}^1$, SLEPc Arnoldi

The Figure 5.7 (respectively 5.6) shows the $M_t{}^1$ dominant eigenvalues relative error (blue line) using the Arnoldi (respectively Krylov-Schur) SLEPc eigen solver. The red line shows the residual associated to each computed eigenpair. Both Solvers are executed with 64 MPI tasks using the fat nodes of PRACE CURIE Supercomputer (4 octo-cores processors per node). We fixed $s = 200$ and $m = 400$. The SLEPc Krylov-Schur (respectively Arnoldi) eigen solver reached the convergence after 22 (respectively 100) restarts.

The residuals according to the relative error associated to each eigenvalue are every satisfying, therefore we consider that the computed eigenvalues accuracy is correct. We do not consider the eigenvectors in this work, we only focus on the eigenvalues. We conclude on the Figures 5.6 and 5.7 that the 200 dominant eigenvalues are conserved through the Algorithm 12.

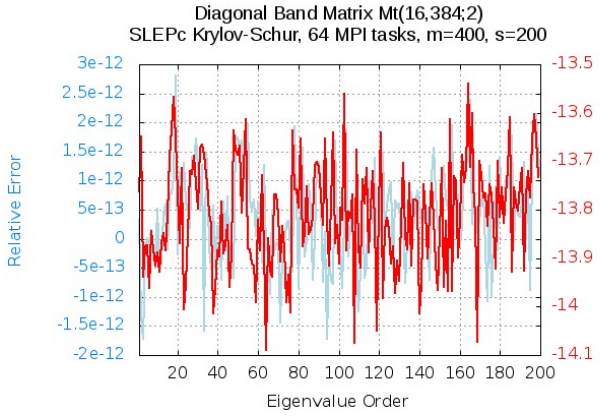

Figure 5.8: $Spec_{2^{15}}$ ${M_t}^2$, SLEPc Krylov-Schur

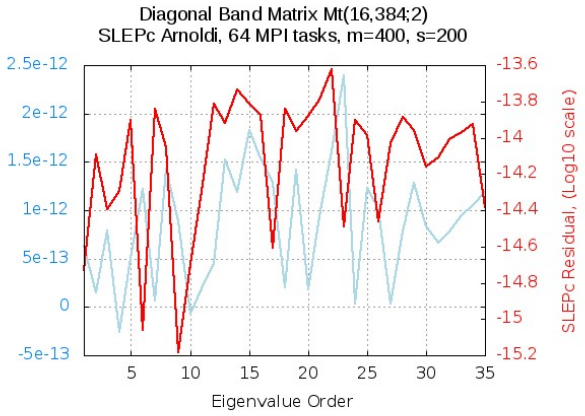

Figure 5.9: $Spec_{2^{15}}$ ${M_t}^2$, SLEPc Arnoldi

The Figures 5.9 (respectively 5.8) shows the ${M_t}^2$ dominant eigenvalues relative error (blue line) using the Arnoldi (respectively Krylov-Schur) SLEPc eigen solver. The red line shows the residual associated to each computed eigenpair. Both Solvers are executed with 64 MPI tasks using the fat nodes of PRACE CURIE Supercomputer (4 octo-cores processors per node). We fixed $s = 200$ and $m = 400$. The SLEPc Krylov-Schur (respectively Arnoldi) eigen solver reached the convergence after 16 (respectively 100) restarts.

Figures 5.8 and 5.9 show the same results for the matrix ${M_t}^2$. Spectrums of ${M_t}^2$ and ${M_t}^1$ are the same, however their eigenvectors may be different. This explains the better convergence of ${M_t}^2$ eigenvalues using the SLEPc Arnoldi eigen solver compared to ${M_t}^1$. Both SLEPc residuals have the same order as 5.6 and 5.7 Figures. The relative errors remains very small, therefore we conclude on the ${M_t}^2$ spectrum conservation through the Algorithm 12.

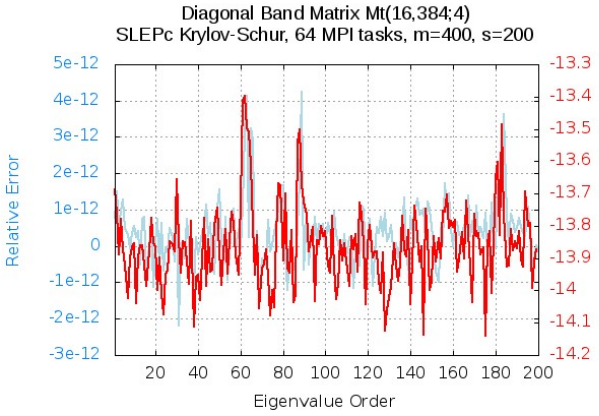

Figure 5.10: $Spec_{2^{15}}$ ${M_t}^4$, SLEPc Krylov-Schur

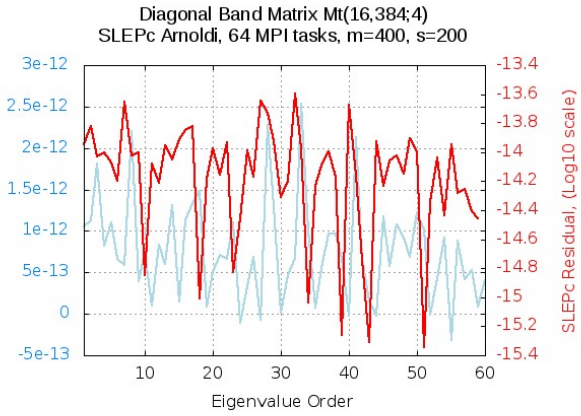

Figure 5.11: $Spec_{2^{15}}$ ${M_t}^4$, SLEPc Arnoldi

Figures 5.11 (respectively 5.10) shows the ${M_t}^4$ dominant eigenvalues relative error (blue line) using the Arnoldi (respectively Krylov-Schur) SLEPc eigen solver. The red line shows the residual associated to each computed eigenpair. Both Solvers are executed with 64 MPI tasks using the fat nodes of PRACE CURIE Supercomputer (4 octo-cores processors per node). We fixed $s = 200$ and $m = 400$. The SLEPc Krylov-Schur (respectively Arnoldi) eigen solver reached the convergence after 11 (respectively 100) restarts.

On figures 5.10 and 5.11, we present the residuals (red line) and the relative errors obtained for the matrix ${M_t}^4$. SLEPc Krylov-Schur solver computes efficiently 200 desired eigenvalues, with a very satisfying residual while SLEPc Arnoldi solver efficiency is not uniform for each matrices presented above: We could obtain 60 dominant eigenvalues for the matrix ${M_t}^4$, 35 for ${M_t}^2$ and finally 20 for ${M_t}^1$. We conclude on the ${M_t}^4$ spectrum conservation through the Algorithm 12 based on the residuals and relative errors orders presented on the Figures 5.11, 5.8 and 5.10.

We realized the same tests for the spectrum $Spec_{2^{16}}$, $Spec_{2^{17}}$ and $Spec_{2^{18}}$. We summarize in the Table 5.3.1 some properties of the matrices M_t generated from spectrums $Spec_{2^{16}}$, $Spec_{2^{17}}$ the $Spec_{2^{18}}$ using the Algorithm 12.

	Size	nnz	max(nnz per row)	$\max_{(i,j)\in[1,n]_{\mathbb{N}}^2} m_{i,j}$	$\min_{(i,j)\in[1,n]_{\mathbb{N}}^2} m_{i,j}$	d	MPI
M_0	2^{16}	294,876	9	86,579,235,711,360	0	4	1
M_t	2^{16}	296,405	13	86,579,235,711,360	−5,681,225,760	4	1
M_0	2^{16}	294,876	9	86,579,235,711,360	0	4	2
M_t	2^{16}	296,434	13	86,579,235,711,360	−7,799,561,280	4	2
M_0	2^{16}	294,876	9	86,579,235,711,360	0	4	4
M_t	2^{16}	296,327	13	86,579,235,711,360	−6,566,071,680	4	4
M_0	2^{17}	720,841	11	7,792,741,480,204,800	0	5	1
M_t	2^{17}	722,635	16	7,792,741,480,204,800	−322,996,766,400	5	1
M_0	2^{17}	720,841	11	7,792,741,480,204,800	0	5	2
M_t	2^{17}	722,517	16	7,792,741,480,204,800	−363,600,316,800	5	2
M_0	2^{17}	589,788	9	86,586,016,446,720	0	4	4
M_t	2^{17}	592,781	13	86,586,016,446,720	−5,839,021,440	4	4
M_0	2^{18}	1,441,737	11	7,792,713,269,913,600	0	5	1
M_t	2^{18}	1,445,221	16	7,792,713,269,913,600	−217,291,636,800	5	1
M_0	2^{18}	1,441,737	11	7,792,713,269,913,600	0	5	2
M_t	2^{18}	1,445,065	16	7,792,713,269,913,600	−272,584,569,600	5	2
M_0	2^{18}	1,441,737	11	7,792,713,269,913,600	0	5	4
M_t	2^{18}	1,445,081	16	7,792,713,269,913,600	−144,145,008,000	5	4

Table 5.3: Diagonal-Band Matrices Characteristics

This Table summarizes some properties of matrices generated using the Diagonal-Band Matrix Generator Algorithm 12. The matrices have been generated using the fat nodes of PRACE CURIE Supercomputer (4 octo-cores processors per node).

For each input spectrum, the reader may note that each output matrix have different characteristics. The non-reproducibility of our diagonal-band matrix generator is due to the random functions used in the Algorithm 12 (that could be avoided depending on the user needs).

We computed the 200 dominant eigenpairs using both SLEPc Arnoldi and Krylov-Schur eigen solvers to evaluate the spectrum conservations through the Algorithm 12 for each matrices listed in the Table 5.3.1.

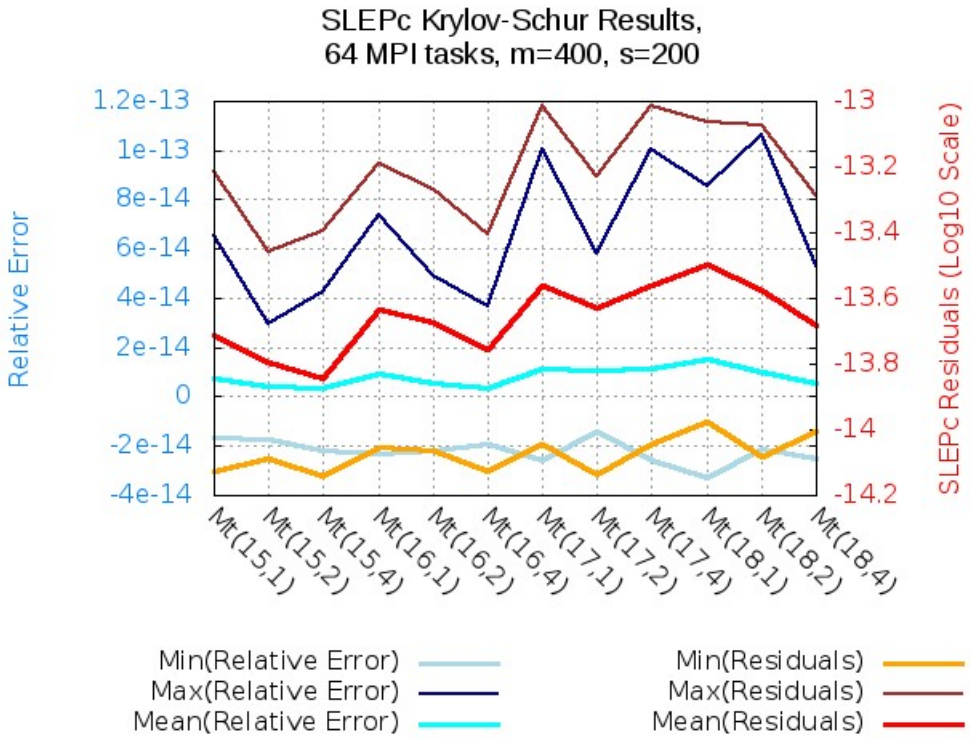

Figure 5.12: Diagonal Band Matrices, SLEPc Krylov-Schur Accuracy

This Figure shows the SLEPc residuals and relative errors for all diagonal-band matrices eigenvalues computed with SLEPc Krylov-Schur. We denote by $Mt(q, np)$ the diagonal band matrix generated by the Algorithm 12 using the spectrum $2^q, q \in [15, 18]_{\mathbb{N}}$ and 2^{np} tasks, $np \in [0, 2]_{\mathbb{N}}$. The Figure shows for both metric (Eigenvalues SLEPc Residuals and Eigenvalues Relative Error) the minimum, maximum and mean respectively. All solvers have been executed with 64 MPI tasks using the fat nodes of PRACE CURIE Supercomputer (4 octo-cores processors per node).

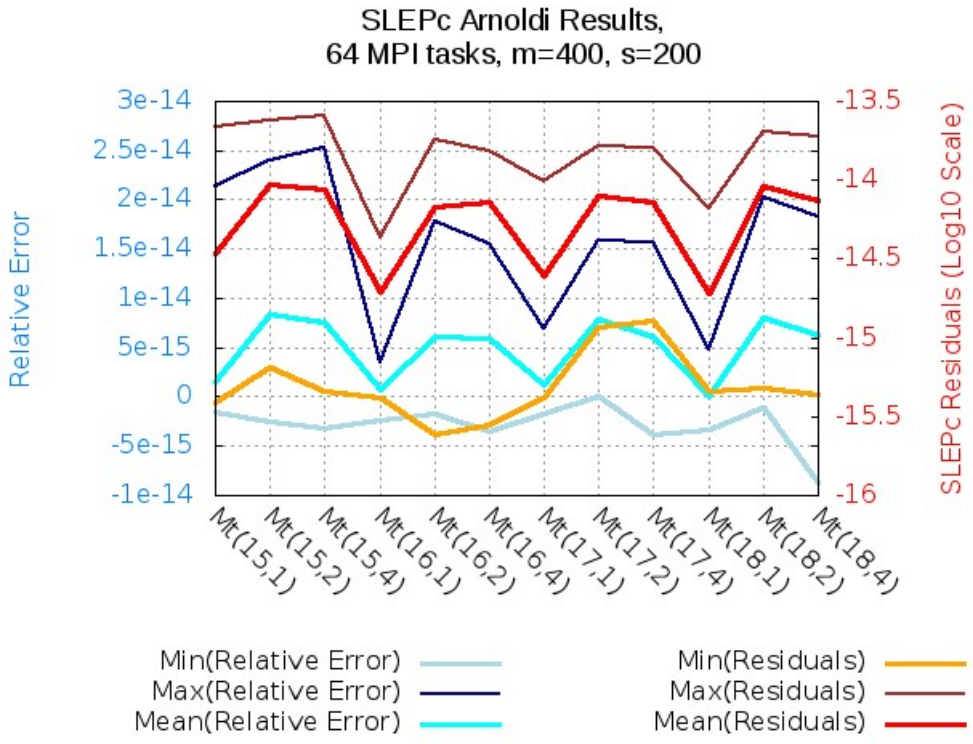

Figure 5.13: Diagonal Band Matrices, SLEPc Arnoldi Accuracy

This Figure shows the SLEPc residuals and relative errors for all diagonal-band matrices eigenvalues computed with SLEPc Arnoldi. We denote by $Mt(q, np)$ the diagonal band matrix generated by the Algorithm 12 using the spectrum $2^q, q \in [15, 18]_{\mathbb{N}}$ and 2^{np} tasks, $np \in [0, 2]_{\mathbb{N}}$. The Figure shows for both metric (Eigenvalues SLEPc Residuals and Eigenvalues Relative Error) the minimum, maximum and mean respectively. All solvers have been executed with 64 MPI tasks using the fat nodes of PRACE CURIE Supercomputer (4 octo-cores processors per node).

Based on the results presented on the Figures 5.12 and 5.13 we conclude on the spectrum conservation through the Algorithm 12 for each computed matrix. We recall that Algorithm 12 does not conserve the eigenvectors, only the spectrum. The SLEPc residuals variations between $Mt(q, np) \forall np \in [0, 2]_{\mathbb{N}}$ with q fixed are explained by the non conservation of the eigenvectors. Nevertheless, their small values indicate that the computed eigenvalues are reliable. For q fixed, all $Mt(q, np) \forall np \in [0, 2]_{\mathbb{N}}$ have the same spectrum (conserved by the Algorithm 12), as shown on the previous results.

The Diagonal-Band Matrix Generator matches with all of the required hypothesis. The spectrum is conserved, however, this is not the case regarding the eigenvectors. For this work, we considered that the eigenvectors conservation was not the priority.

We may temper the hypothesis "The matrix shall not look like a trivial matrix", as the resulting matrices have a diagonal band shape and the sparsity is relatively high (cf Table 5.3.1, maximum non-zeros elements per row). The Algorithm 12 presents many interesting properties, especially regarding the output matrix mathematical properties.

Conclusion

Validation is a key point in many scientific frameworks. We highlighted the needs of open-source matrix generators or collection that would provide huge dimension matrices and their spectrum.

As a remedy, two matrix generators have been implemented. We choose different spectrums to enhance that both Algorithms 12 and 11 are not dependent of its distribution.

The diagonal-band matrix generator presents many benefits, in terms of parallelism and numerical accuracy. Its mathematical properties are more interesting than the dense matrix generator and its parallelism is definitely better.

Nevertheless, the dense matrix generator is also very interesting as the provided matrices have a large non-zeros elements and their repartition is completely random. The reader will notice in the next section that the dense matrix generator tends to conserve the convergence behavior for the ERAM solver: we mean that an ERAM, using exactly the same parameters applied on a matrix and its dense version issued from the dense matrix generator tends to converge similarly.

This point is important insofar as it confirms the ability of the dense generator to provide good candidates to tests the numerical efficiency of ultra-scale eigen solvers.

All the matrices presented in this section will be used to illustrate further results in this thesis.

CHAPTER 6

Restarting Strategies Influence on the ERAM Convergence

As presented in Chapter 3, choosing a pertinent combination to compute the restarting vector v_1 may considerably accelerate the convergence. In this chapter we present several restarting strategies that we will use during all this thesis.

6.1 ERAM Restarting Strategies

The ERAM (cf Algorithm 5) restarts its process with a restarting vector computed using the equation 6.1 :

$$v_1^{(i+1)} = \sum_{j=1}^{\gamma} \Re(u_j^{(i)}), m \geq \gamma \geq s \geq 1 \tag{6.1}$$

This linear combination uses a uniform weight for each desired approximated eigenvectors. In the scientific literature, this weight scheme is definitely the most classic and widely spread for the existing ERAM implementations.

6.1.1 The Restarting Strategies

Some other weight schemes were discussed, especially in [Sedrakian 2005]. As an illustration, Yousef Saad suggested in [Saad 2011] to use the following equation to compute the restarting vector:

$${v_1}^{(i+1)} = \sum_{j=1}^{\gamma} resTr_j^{(i)} \Re(u_j^{(i)}), \tag{6.2}$$

Where $\gamma \in [1, m]_{\mathbb{N}}$.

This weight scheme favors the slowest convergence eigenpair(s), as their associated residual(s) will be higher. Such weight scheme may be pertinent as it is well known that the Restarted Arnoldi Method (see the section 3.2) favors the outer eigenvalues of the spectrum, therefore it is not surprising to observe that $\{\theta_s^{(i)}, u_s^{(i)}\}$ converges faster than $\{\theta_{s-1}^{(i)}, u_{s-1}^{(i)}\}$[1]. Nevertheless, we moderate this purpose as this observation varies from one ERAM configuration to another (depending on the matrix itself, the subspace size value, the number of desired eigenpairs etc ...).

[1] This is an illustration, not a generality.

In what follows, we aim to be very generic to compute the restarting vector. We introduce $\alpha_j^{(i)}$, the restarting coefficient associated to the j^{th} Ritz eigenpair used to compute the restarting vector $v_1^{(i+1)}$. Then, the equation 3.8 is rewritten as follows 8.5:

$$v_1{}^{(i+1)} = \sum_{j=1}^{\gamma} \alpha_j^{(i)} \Re(u_j^{(i)}), \tag{6.3}$$

Where $\gamma \in [1, m]_{\mathbb{N}}$.

The $\alpha_j^{(i)}$ coefficients may weight differently the Ritz eigenvectors, depending on their values. The restarting eigenvector may use three different weight schemes: whether we use a uniformly weight, a randomly weight or an ordered (increasing or decreasing) weight.

In this Thesis, we will study these three schemes to compute the restarting vector. Additionally to the weight schemes, we will study two restarting coefficients profiles:

- The restarting coefficients uses Ritz eigenpairs information (including the residuals),
- The restarting coefficients do not contain any Ritz eigenpairs information, but they are computed relatively to their order (in terms of real modulus eigenvalue).

We summarize in the Table 6.1.1 the restarting strategies used in our ERAM framework:

Restarting Strategy	Abreviation	$\alpha_j^{(i)}$	Weight Scheme*
Default	α_{Def}	1	U
Residual	α_{Res}	$\lvert 1 - resTr_j^{(i)} \rvert$	O or R or U
Linear	α_{Li}	$(s - j)$	O
Linear Residual	α_{LiRes}	$(s - j) \times \lvert 1 - resTr_j^{(i)} \rvert$	O or R
Lambda	α_{La}	$\lvert \theta_j^{(i)} \rvert$	O or U
Lambda Residual	α_{LaRes}	$\lvert \theta_j^{(i)} \rvert \times \lvert 1 - resTr_j^{(i)} \rvert$	O or R or U

Table 6.1: We summarize in this table the values of restarting coefficients and their weight schemes: O refers to an Ordered weight, R to a Randomly weight and U to a Uniformly weight. *Relatively to the real modulus of the desired eigenvalues.

The reader may note that many other restarting strategies can be derived from the Table 6.1.1, the restarting strategies listed above do not constitute an exhaustive list.

6.2 The ERAM Restarting Strategies Convergence

In this section, we will emphasize the restarting strategies influence on the ERAM convergence. In what follows, we will execute many ERAMs, with the same parameters excepted the restarting strategy.

We define by $EP = \{m, s, \gamma, orthogonalization, \alpha_X\}$ the ERAM Parameters set (we will detail each component below). We will consider in this section that these parameters are fixed for the complete ERAM execution.

- $m \in [1, n]_{\mathbb{N}}$ the Krylov subspace size,
- $s \in [1, m]_{\mathbb{N}}$ the number of desired eigenpairs,
- $\gamma \in [1, m]_{\mathbb{N}}$ the number of Ritz eigenpairs used to compute the restarting vector ${v_1}^{(i+1)}$,
- *orthogonalization* the orthogonalization process used by the Arnoldi Method. *orthogonalization* may be CGS, CGSR ... As a first step, we will fix *orthogonalization* = $CGSR$: This choice is influenced by the previous work executed in [Dubois 2011d],
- α_X the restarting strategy. α_X value may be α_{Def}, α_{Res}, α_{Li}, α_{LiRes}, α_{La} or α_{LaRes}.

To highlight the restarting strategies behavior, we will execute, for each target matrix (that will be listed later), the following test:

We fix a constant ERAM Parameter Set whose components, except that restarting strategy is fixed for one target matrix. We will vary he restarting strategy parameters among the one presented in the Table 6.1.1 and compare the results.

We apply the test presented above to the target matrices summarized in the Table 6.2:

	Size	nnz	Source
Ex11	16,614	1,096,948	UF SMC[Davis]
$Ex11_{Dense}$	16,614	275,991,771	Issued from A.0.2.1
$Fission_{Dense}$	10,000	99,980,003	Issued from 5.2.1.1 and CEA neutronic applications
$Fission_{2^{15}}$	16,384	286,402,691	Issued from 5.2.1.1 and CEA neutronic applications
Mixtank_new	29,957	1,995,041	UF SMC[Davis]
$Mixtank\_new_{Dense}$	29,957	897,361,938	Issued from A.0.2.2
Rim	22,560	1,014,951	UF SMC[Davis]
Rim_{Dense}	22,560	508,908,483	Issued from A.0.2.3
$Spec_{2^{16}}, {M_t}^4$	32,768	296,327	Issued from 5.3.1

Table 6.2: Target Matrices to Test the Restarting Strategies.

We mostly used matrices provided from the famous Tim Davis matrices collection [Davis]. We choose different spectrums, in terms of the real modulus eigenvalues and their distribution. We apply the ERAM restarting strategy behavior study to the matrices issued from our matrix generators (please, see the Chapter 5). We choose to focus to one diagonal band matrix, but the results presented in this section can be extended to the other diagonal-band matrices.

For each figure presented in this section, one color will be associated to one restarting strategy as follows: α_{Def}, α_{Res}, α_{Li}, α_{LiRes}, α_{La} and α_{LaRes}. Such colors "code" remain available in this section only.

We recall that the restarting strategy has no impact on the parallel execution time, meaning that using α_{Def} or α_{LaRes} will **not change the execution time per restart but it will influence the number of restarts to reach the convergence, therefore the global parallel execution time**.

6.2.0.1 Ex11 & $Ex11_{Dense}$

The Figure 6.1 (respectively 6.2) presents the restarting strategy convergence behavior of Ex11 and $Ex11_{Dense}$ matrices respectively. Both ERAM presented on figures 6.1 and 6.2 have the following ERAM Parameter set: $EP = \{m = 20, s = 4, \gamma = 4, CGSR, \alpha_X\}$.

The number of restarts until the convergence is from our point of view the best performance metric for this study. As the parallel execution time remains identical whatever the restarting strategy is, it measures simultaneously the algorithm performance and the numerical quality of the ERAM with the fixed parameter set. This would have not been the case if the restarting strategies would have had an impact on the parallel execution time per restart.

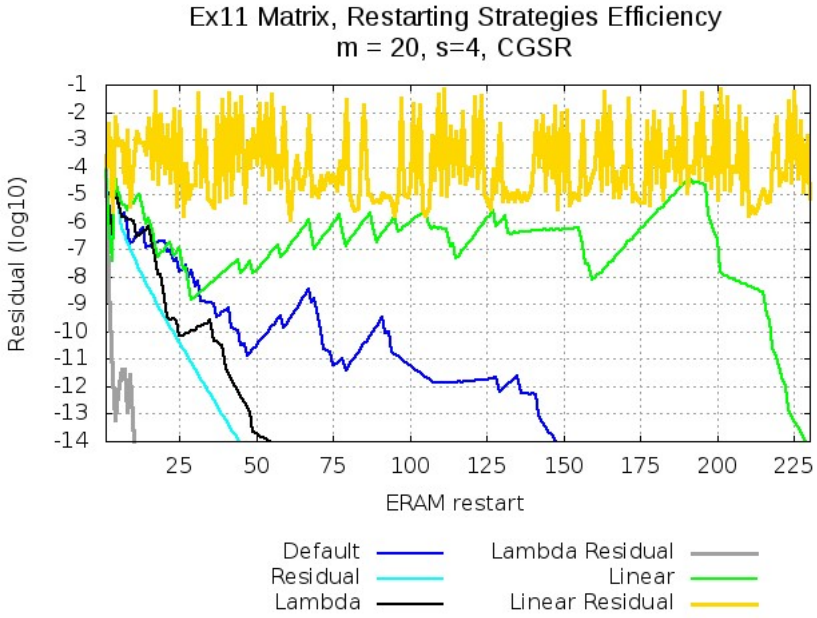

Figure 6.1: $Ex11$ Matrix, ERAM Restarting Strategies Convergence.

The ERAM has $m = 20$, $s = \gamma = 4$ and a CGSR orthogonalization process for the Arnoldi Method (s is the number of desired eigenpairs while γ is the number of eigenpairs we use to compute the restarting vector). We present the dominant eigenvalue residual evolution. We used a single thin node (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. The ERAMs reach the convergence in the following order (according to the restarting strategies): α_{LaRes}, α_{Res}, α_{La}, α_{Def} and α_{Li}. α_{LiRes} did not reach the convergence.

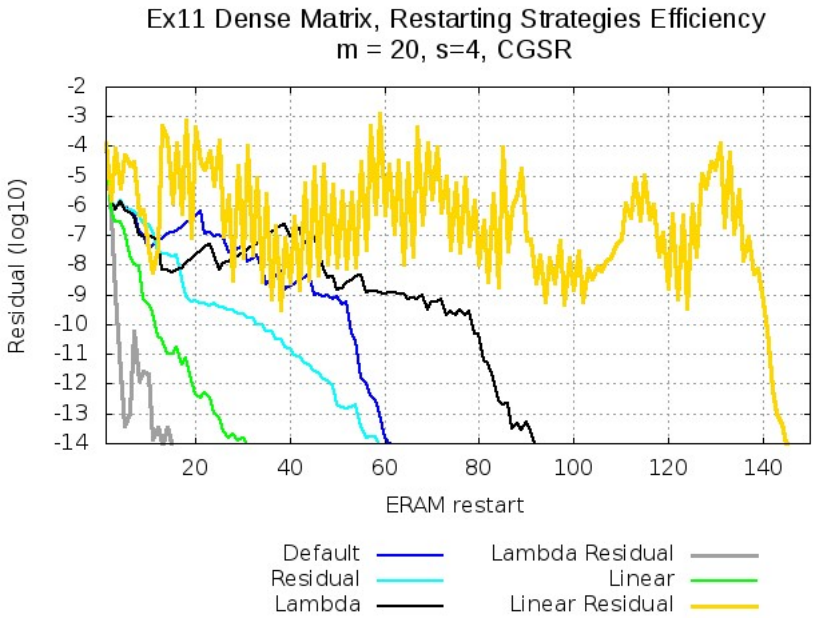

Figure 6.2: $Ex11$ Dense Matrix, ERAM Restarting Strategies Convergence with respect to the number of restarts.
The ERAM has $m = 20$, $s = \gamma = 4$ and a CGSR orthogonalization process for the Arnoldi Method (s is the number of desired eigenpairs while γ is the number of eigenpairs we use to compute the restarting vector). We present the dominant eigenvalue residual evolution. We used 234 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores Intel® Xeon®) of the PRACE Curie supercomputer. The ERAMs reach the convergence in the following odrer (according to the restarting strategies): α_{LaRes}, α_{Li}, α_{Res}, α_{Def}, α_{La} and α_{LiRes}.

On the Figure 6.1, the most efficient restarting strategy is without any contest the α_{LaRes} one. We reach the convergence in only 10 restarts. Then, the α_{Res} and α_{La} reach the convergence at 47 and 52 restarts respectively. The other restarting strategies and especially α_{Def} are clearly less efficient.

The α_{LaRes} restarting strategy efficiency is still very impressive for the $Ex11_{Dense}$ matrix (Figure 6.2).

The performance of the α_{Def} compared to the best restarting strategy is still relatively poor for both matrices.

The reader may notice that the convergence performances between Ex11 (6.2.0.1) and its Dense version (issued from A.0.2.1) are similar. This confirms the ability of the dense matrix generator to provide pertinent test matrices for eigenvalue solvers. This observation will be emphasized with the matrices that will be presented below.

Note that for both $Ex11$ and $Ex11_{Dense}$, the α_{LaRes} convergence presents a *"peak"* during its convergence. This peak is due to a spectrum shift: Due to the Ex11 eigenvalue repartition (please see Figure A.1), θ_4 tends to converge to λ_5 instead of λ_4. Until the peak, θ_4 has reach a relatively accurate approximation of λ_5, the next restart θ_4 starts to converge to λ_4, which explains the sudden convergence peak. Same behavior is observed for the other restarts, but the *"shift peaks"* are less accentuated.

This observation is not necessarily true anymore with an another ERAM parameter set, which emphasizes the complexity of fixing a pertinent ERAM parameter. As an illustration, with an other m and or s value, the shift peak convergence would not have necessarily be observed anymore. This strengthen the necessity to converge to an ERAM whose parameters would be dynamically adapted to the numerical evolution of the ERAM. In other terms, we need to build a Smart-Tuning heuristic that would optimize the numerical parameters of the ERAM according to its convergence.

6.2.0.2 Fission$_{Dense}$&$Fission_{2^{15}}$

We executed the same study with the Fission matrices presented in section 5.2.1.1. This study has two objectives: evaluate the restarting strategies impact on one hand and in a second hand, show the convergence similarities between the two generated matrices. Indeed, we recall that $Fission_{2^{15}}$ is an *"extension"* of the $Fission_{Dense}$ matrix (cf section 5.2.1.1). Therefore we aim to show that extend the lowest part of the matrix spectrum has a limited impact on the ERAM restarting strategies convergence.

The Figure 6.3 (respectively 6.4) presents the ERAM using $EP = \{m = 15, s = 4, \gamma = 4, CGSR, \alpha_X\}$ (respectively $EP = \{m = 10, s = 4, \gamma = 4, CGSR, \alpha_X\}$).

The restarting strategies efficiency are still pretty different. We note that on Figure 6.4 (respectively Figure 6.3), α_{Res} (respectively α_{Def}) and α_{La} have a very similar behavior, while their coefficients are very different.

On Figure 6.4, the α_{La} restarting strategy converges at the restart 45, which is 2.3 times faster than α_{Def}. We recall that using α_{La} or any other restarting strategy than α_{Def} does not imply neither parallel communications nor additional computation costs. The convergence is pretty smooth for the α_{Def} and α_{La} restarting strategies. The α_{LiRes}

restarting strategy almost reached the convergence: the reader may have noticed that this restarting strategy particularly has a *"chaotic"* behavior, for almost all the matrices presented in this thesis. We shall shortly return to this subject later.

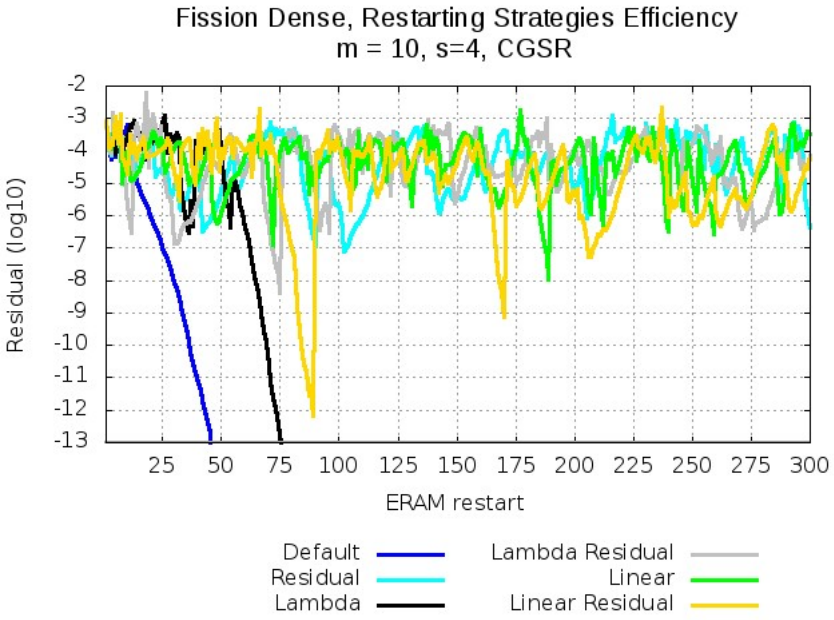

Figure 6.3: *Fission* Dense Matrix, Restarting Strategies Convergence.
The ERAM has $m = 10$, $s = 4$ and a CGSR orthogonalization process for the Arnoldi Method. We present the dominant[2] eigenvalue residual evolution. We used 400 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. ERAMs reach the convergence in the following order (according to the restarting strategies): α_{Def} then α_{La}. α_{Li}, α_{LiRes}, α_{LaRes} and α_{Res} did not reach the convergence.

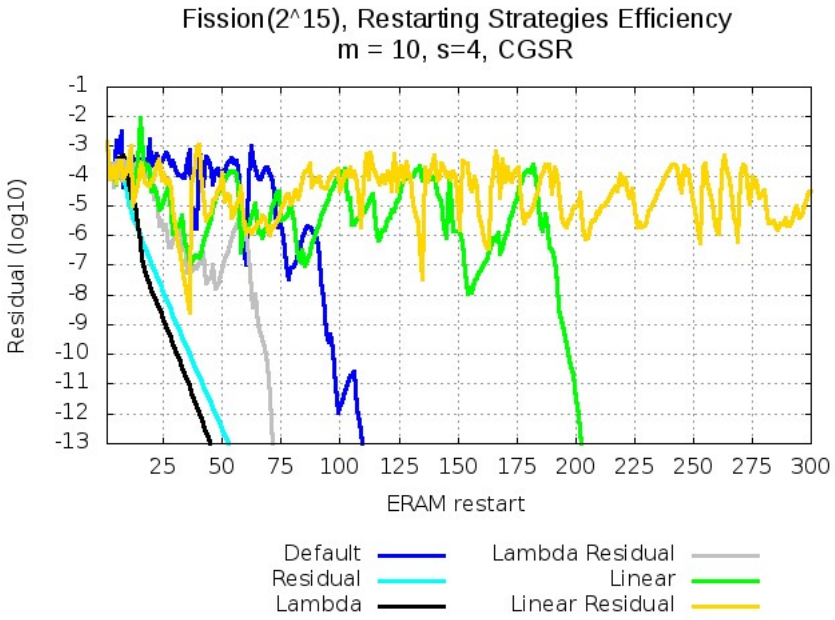

Figure 6.4: $Fission_{2^{15}}$ Dense Matrix, ERAM Restarting Strategies Convergence.
The ERAM has $m = 10$, $s = 4$ and a CGSR orthogonalization process for the Arnoldi Method. We present the dominant[3] eigenvalue residual evolution. We used 400 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. ERAMs reach the convergence in the following order (according to the restarting strategies): α_{La}, α_{Res}, α_{LaRes}, α_{Def} and α_{Li}. α_{LiRes} did not reach the convergence.

6.2.0.3 Mixtank_new & Mixtank_new$_{Dense}$

The $Mixtank\_new$ matrix (therefore $Mixtank\_new_{Dense}$) has a complete different spectrum from the matrices studied above. We present on the Figures 6.5 and 6.6 the ERAM convergence by using the following parameter set $EP = \{m = 15, s = 4, \gamma = 4, CGSR, \alpha_X\}$.

Both matrices have a convergence that we can characterize as a *"plateau"* convergence, meaning that we alternate between a stagnation part and then a convergence. On the Figure 6.5, the convergence is pretty slow and the reader may notice that for this matrix, the α_{LiRes} restarting strategy is relatively efficient.

We executed the same test with the $Mixtank\_new_{Dense}$ matrix, the results are presented on the Figure 6.6. None of the restarting strategies could provide the convergence at 10^{-14} threshold. Nevertheless, the reader may notice the similarities of ERAM convergence with the original matrix $Mixtank\_new$. In this configuration, the α_{Def} restarting strategy offers the best performance.

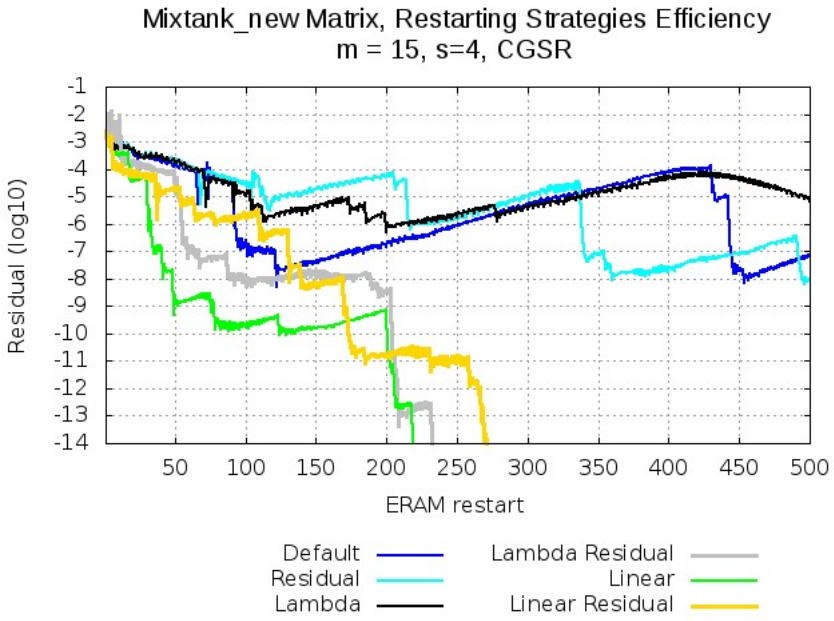

Figure 6.5: *Mixtank_new* Matrix, Restarting Strategies Convergence.
The ERAM has $m = 15$, $s = 4$ and a CGSR orthogonalization process for the Arnoldi Method. We present the dominant eigenvalue residual evolution. We used a single thin node (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. The ERAMs reach the convergence in the following order (according to the restarting strategies): α_{Li}, α_{LaRes} and α_{LiRes}. α_{La}, α_{Res} and α_{Def} did not converge.

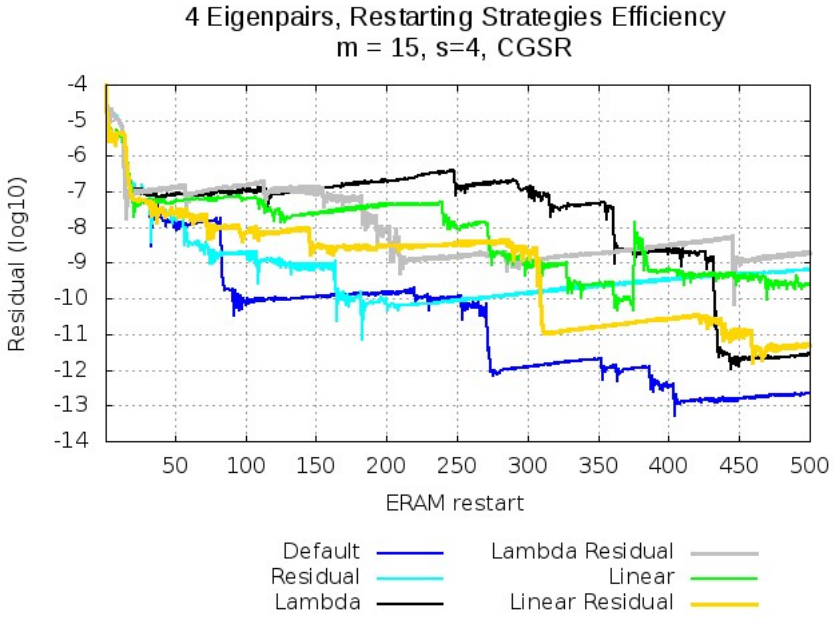

Figure 6.6: *Mixtank_new* Dense Matrix, Restarting Strategies Convergence. The ERAM has $m = 15$, $s = 4$ and a CGSR orthogonalization process for the Arnoldi Method. We present the dominant eigenvalue residual evolution. We used 29 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer.

6.2.0.4 Rim & Rim_{Dense}

We present on the Figure 6.7 (respectively 6.8) the Rim (respectively Rim_{Dense}) matrix convergence with ERAMs using as parameter set $EP = \{m = 20, s = 4, \gamma = 4, CGSR, \alpha_X\}$ (respectively $EP = \{m = 15, s = 4, \gamma = 4, CGSR, \alpha_X\}$).

For both matrices, the best convergence scheme is obtained with the α_{Li} restarting strategy, wile α_{Def} restarting strategy has poor performances. The α_{LiRes} restarting strategy has still a chaotic behavior.

As a primary observation, the convergence behavior of Rim_{Dense} is very similar to the original matrix. The reader may notice that the convergence efficiency is the same, meaning that the restarting strategies reach the convergence in almost the same order: α_{Res} and α_{Li}, then α_{La}, α_{LaRes} and finally α_{Def}. For both matrices, α_{LiRes} has a very chaotic behavior.

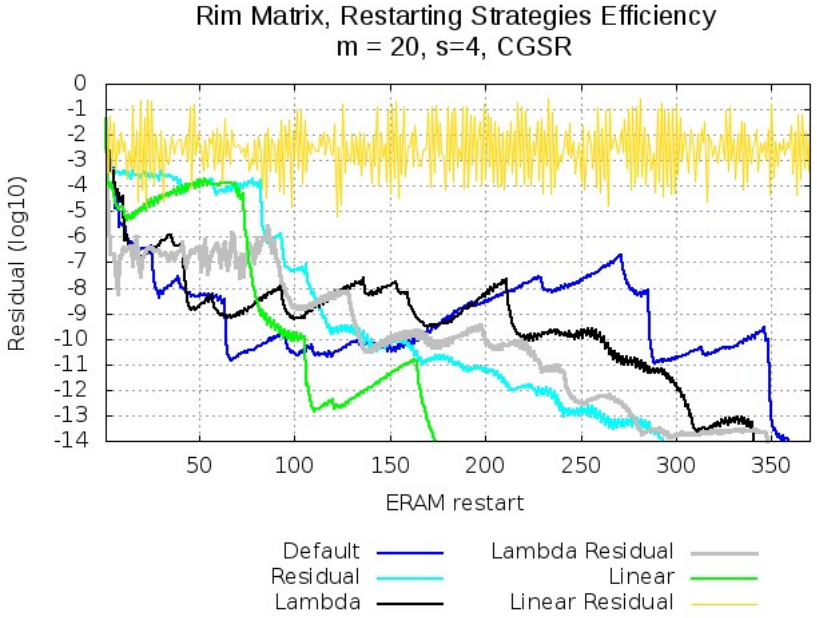

Figure 6.7: *Rim* Matrix, Restarting Strategies Convergence.
The ERAM has $m = 20$, $s = 4$ and a CGSR orthogonalization process for the Arnoldi Method. We present the dominant[4] eigenvalue residual evolution. We used a single thin node (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. ERAMs reach the convergence in the following order (according to the restarting strategies): α_{Li}, α_{LaRes}, α_{Res}, α_{La} and α_{Def}. α_{LiRes} did not converge.

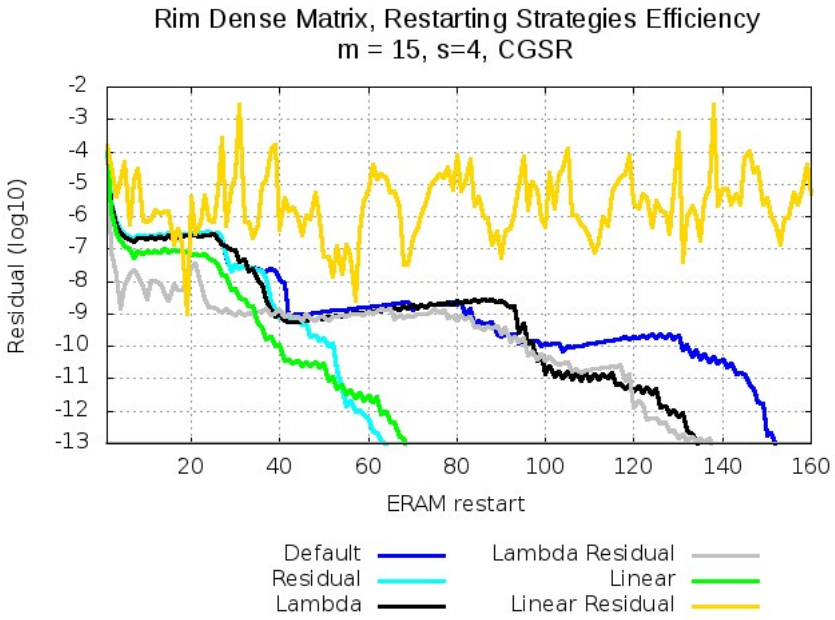

Figure 6.8: *Rim* Dense Matrix, Restarting Strategies Convergence.

The ERAM has $m = 15$, $s = 4$ and a CGSR orthogonalization process for the Arnoldi Method. We present the dominant eigenvalue residual evolution. We used 188 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. ERAMs reach the convergence in the following order (according to the restarting strategies): α_{Res}, α_{Li}, α_{LaRes}, α_{La} and α_{Def}. α_{LiRes} did not converge.

6.2.0.5 $\mathbf{Spec}_{2^{16}}, M_t^4$

The last but not the least, we present the ERAM restarting strategy behavior by using one of the diagonal band matrices, issued from the Algorithm presented in section 5.3.1. The observation listed in this section can be extended to all the diagonal band matrices presented in this thesis. For each results presented below, we computed the 3 dominant eigenpairs and used a CGSR orthogonalization process for the Arnoldi Method.

Subspace Size	α_{Def}	α_{Res}	α_{Li}	α_{LiRes}	α_{La}	α_{LaRes}
100	#	#	#	#	#	27
105	#	#	#	#	#	31
110	**212**	**106**	**345**	#	**242**	42
115	**89**	**97**	**83**	#	**72**	43
120	**74**	**77**	**85**	#	**63**	54
125	**59**	**44**	**48**	#	**52**	50

Table 6.3: We present the dominant eigenvalue residual evolution. We used 256 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer for each tests presented above. This Table summarizes the number of restart to reach he convergence for each restarting strategy. # means that ERAM did not converge.

The α_{LaRes} efficiency is definitely the best restarting strategy. Its behavior is very similar to the $Ex11$ and $Ex11_{Dense}$ matrices. In fact, their sprectums have similarities in terms of eigenvalue distribution and real modulus. We note that α_{LaRes} is very efficient to converge to the exterior eigenvalues, while other restarting strategies such as α_{Res} will favors the dominant (in terms of real modulus) eigenvalues and have more trouble to reach the convergence to the lowest eigenvalue.

6.2.1 The Restarting Strategy Efficiency

Matrix	EP	α_{Def}	α_{Res}	α_{Li}	α_{LiRes}	α_{La}	α_{LaRes}
Ex11	$\{m = 20, s = 4, \gamma = 4, CGSR, \alpha_X\}$	1	3,29	0,65	#	2,69	14,8
$Ex11_{Dense}$	$\{m = 20, s = 4, \gamma = 4, CGSR, \alpha_X\}$	1	1,05	2	0,42	0,67	3,88
$Fission_{Dense}$	$\{m = 15, s = 4, \gamma = 4, CGSR, \alpha_X\}$	#	#	0,15	#	0,58	#
$Fission_{2^{16}}$	$\{m = 10, s = 4, \gamma = 4, CGSR, \alpha_X\}$	1	1,98	0,55	#	2,3	1,49
Mixtank_new	$\{m = 15, s = 4, \gamma = 4, CGSR, \alpha_X\}$	#	#	2,27	1,84	#	2,15
Rim	$\{m = 20, s = 4, \gamma = 4, CGSR, \alpha_X\}$	#	#	#	#	#	#
Rim_{Dense}	$\{m = 15, s = 4, \gamma = 4, CGSR, \alpha_X\}$	1	2,45	2,2	0,49	1,13	1,1
$Spec_{2^{16}}$	$\{m = 110, s = 3, \gamma = 3, CGSR, \alpha_X\}$	1	2	0,61	#	0,88	5,05
	$\{m = 115, s = 3, \gamma = 3, CGSR, \alpha_X\}$	1	0.92	1,07	#	1,24	2,07
	$\{m = 120, s = 3, \gamma = 3, CGSR, \alpha_X\}$	1	0,96	0,87	#	1,17	1,37
	$\{m = 125, s = 3, \gamma = 3, CGSR, \alpha_X\}$	1	1,34	1,23	#	1,13	1,18

Table 6.4: We compare the efficiency of restarting strategies compared to α_{Def}. Each restarting strategy efficiency is computed by using $\frac{\alpha_{Def}}{\alpha_X}$. If α_{Def} does not converge, we take as a reference the maximum of ERAM restarts, which is 500.

We presented in this section some restarting strategies used to compute restarting vector. The idea is to use different profiles, in terms of weight schemes but also in terms of *"contents"*, by using -or not- Ritz eigenpair information(s).

It results from our observations that α_{LaRes} and α_{La} are pretty efficient with eigenvalues whose spectrum is large (in terms of real modulus). Nevertheless, these restarting strategies have more difficulties to reach the convergence to the inner desired eigen subspace, while uniform weight restarting strategies (such as α_{Def} or α_{Res}) are more sensitive to the inner desired eigen subspace.

With a large subspace size (compared to the number of desired eigenpairs s), the restarting strategy behavior is blurred. As restarting strategies do not imply neither parallel communications nor additional computations, their use improves the ERAM con-

vergence by using small subspace sizes: this aims to reduce the number of restarts to reach the convergence without increasing the subspace size and therefore the number of global/blocking communications and memory storage.

6.3 The Twin Restarting Strategies

The restarting coefficients formula (cf Table 6.1.1) may be similar to other restarting strategies coefficients, depending on the Ritz eigenpairs accuracy. In the case that $resTr_j^{(i)}$ has a small value, $|1 - resTr_j^{(i)}|$ is very close from 1. Starting from this observation, we have the following equations:

$$\begin{cases} \forall j \in [1,\gamma]_{\mathbb{N}}, & \lim\limits_{resTr_j^{(i)} \to 0} \textbf{Residual}(\alpha_j^{(i)}) & \to & \textbf{Default}(\alpha_j^{(i)}), \\ \forall j \in [1,\gamma]_{\mathbb{N}}, & \lim\limits_{resTr_j^{(i)} \to 0} \textbf{Linear Residual}(\alpha_j^{(i)}) & \to & \textbf{Linear}(\alpha_j^{(i)}), \\ \forall j \in [1,\gamma]_{\mathbb{N}}, & \lim\limits_{resTr_j^{(i)} \to 0} \textbf{Lambda Residual}(\alpha_j^{(i)}) & \to & \textbf{Lambda}(\alpha_j^{(i)}), \end{cases} \tag{6.4}$$

After a satisfiable threshold, each restarting coefficients pair verifies the equation 6.4. We qualify this behavior as the *"Twin Restarting Strategies"*. Nevertheless, twin restarting strategies may induce very different convergence behavior, as presented in the previous figures. The matrices with large eigenvalues such as $Ex11$, $Ex11_{Dense}$ and the generated Diagonal Band matrices tend to have twin restarting strategies whose convergence behavior is really different. The matrices such as $Mixtank\_new$ or Rim (and their dense versions) respect the twin restarting strategies principle, meaning that the convergence evolutions are similar. This is illustrated on the Figure 6.9, Figure 6.10 and Figure 6.11. These restarting strategies convergence have been presented before, we zoomed the twin restarting strategies to emphasize the similarity behavior.

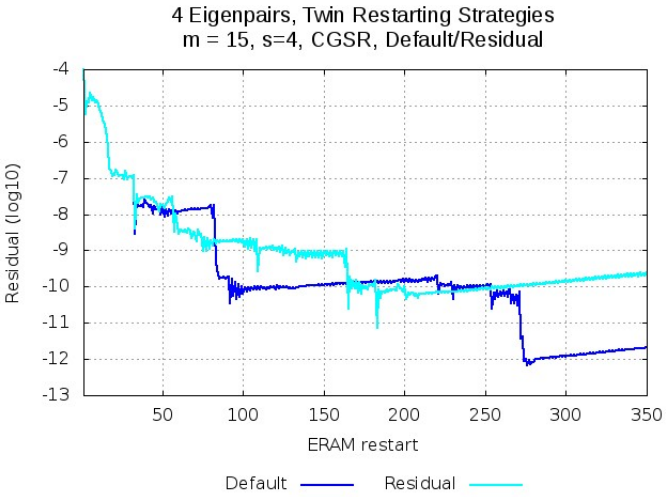

Figure 6.9: *Mixtank_new* Dense, α_{Res} & α_{Def}.

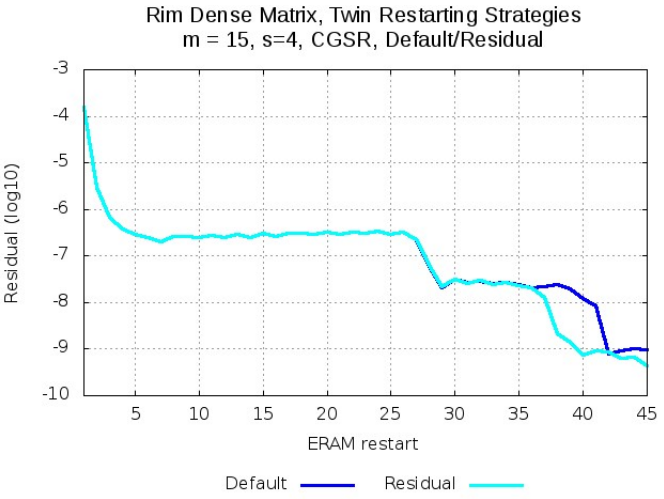

Figure 6.10: *Rim* Dense, α_{Res} & α_{Def}.

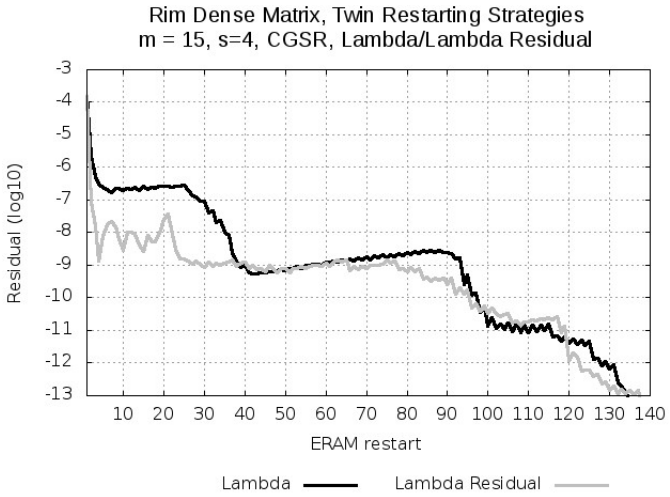

Figure 6.11: *Rim* Dense, α_{La} & α_{LaRes}.

The twin restarting strategy behavior is interesting in the case we change the restarting strategy during the runtime execution. With this in mind, we know that switching a restarting strategy by its twin for the Mixtank_new or the Rim matrix may have a limited impact. The opposite behavior will be observed for the matrices that are not sensitive to the twin restarting strategies phenomena, such as the Ex11 one.

Conclusion

In this Chapter, we presented the restarting strategies and their associated weight schemes. We propose two "profiles" as restarting strategies, one using the Ritz eigenpairs information and one independent from it.

We proposed to emphasize the restarting strategy influence by using many target matrices, issued from the University of Florida Matrix Collection [?] and from our matrix generators presented in the Chapter 5.

This chapter emphasized the ERAM restarting strategies behavior and the possible gain we may have (compared to a uniform scheme) in terms of number of restarts to reach the convergence. Using the restarting coefficients presented in the Table 6.1.1 implies neither parallel communications nor operations, we recall that we just reuse the available Ritz eigen information. As presented in the Table 6.2.1, choose the right restarting strategy may be from 2 to 14 times faster in terms of number of restarts compared to the uniform weight scheme Default.

We emphasized that some sprectrums are more sensitive to the *"Twin restarting strategy"* behavior: by twin, this means that two restarting strategies such as Default and Residual may have almost the same restarting coefficients values. For some matrices such as Mixtank_new or Rim (and their *"Dense"* versions issued from the dense matrix generator) the twin restarting strategy behaves similarly, then we observe that the two respective Krylov subspaces have a different convergence direction. For the matrices whose spectrum is very large such as $Ex11$ or the diagonal band matrices, the twin restarting strategy behavior is annihilated: The Krylov subspace generated with restarts Default and Residual have a complete different convergence direction. A part of the results presented in this chapter have been executed under the expert advisement of M. Leroy Anthony Drummond at the Computer Science Department of the Lawrence Berkeley National Laboratory.

As a conclusion, finding the optimum parameter set among all the possibilities remain very tricky. Nevertheless, we emphasized the impact of the restarting strategy in this chapter, how this parameter can influence the ERAM convergence at no *"computation costs"*. The results presented in this chapter shown promising ERAM numerical convergence acceleration but the results are heterogeneous, leading to the need of a dynamic modification of the ERAM restarting strategy.

We aim to dynamically adapt the restarting strategies so as to reduce the number of restarts until convergence. Such smart-tuning implies to study the mixed restarting strategy and ensure that mixing several restarting strategy can provide the convergence. The next chapter focuses on the mixed restarting strategies.

Chapter 7
ERAM with Mixed-Restarting Strategies

The previous chapter presented the restarting strategies behavior and their impact on the ERAM convergence. In this chapter, we focus on the mixed restarting strategy, id est the investigation phase to outcome to a restarting strategy smart tuning ERAM.

Starting from the ERAM restarting strategies behavior (Chapter 6), we observed that there is a need to dynamically change the restarting strategy to accelerate the convergence to the desired eigen subspace.

All the following tests use the target matrices generated at the Chapter 5.

7.1 The ERAM Restarting Vector with Mixed Restarting Strategies

As a first step, we need to precise how we use the restarting strategies. For each ERAM configuration presented in the previous chapter, we applied the Convergence Analysis Algorithm 10. For each target matrix (summarized in the Table 7.1), we retain the restarts reported by the Algorithm 10 and identify the divergence and stagnation status. We recall that the aim of this study is to avoid such convergence behavior and lead to an ERAM using with a smooth convergence.

Let's consider an ERAM using as parameter set $EP = \{m, s, \gamma, orthogonalization, \alpha_X\}$. If the Algorithm 10 has detected at the i^{th} restart a divergence or stagnation behavior, we execute a new instance of the ERAM with the same $EP = \{m, s, \gamma, orthogonalization, \alpha_X\}$ and change its restarting strategy at the i^{th} restart by another one.

In this thesis, we used 6 restarting strategies, leading to 36 configurations for one target matrix at a given restart i. We will limit our study to the Default restarting strategy in this section, but the conclusion presented in this chapter can be extended to the other restarting strategies.

This study has two main objectives: the first is to check if switching the restarting strategy during the ERAM execution has an interest. If the answer is no, then there is absolutely no interest to build an ERAM smart tuning based on the restarting strategies modifications. If the answer is yes, then we are proving that an ERAM with restarting strategies smart tuning can be developed. Then comes the second objective, id est is there any combination or ERAM behavior that emerges of the mixed restarting strategies.

	Size	nnz	Source
$Ex11_{Dense}$	16,614	275,991,771	Issued from A.0.2.1
$Fission_{Dense}$	10,000	99,980,003	Issued from 5.2.1.1 and CEA neutronic applications
$Mixtank\_new_{Dense}$	29,957	897,361,938	Issued from A.0.2.2
Rim_{Dense}	22,560	508,908,483	Issued from A.0.2.3
$Spec_{2^{15}}, {M_t}^4$	16,384	181,403	Issued from 5.3.1

Table 7.1: Target Matrices to Test the Mixed Restarting Strategies.

All the following tests have been executed on the PRACE CURIE[1] Supercomputer. The Curie machine is located at the Très Grand Centre de Calcul (TGCC) and is the 26^{th} most powerful supercomputer (according to [Meuer]).

Curie is composed of 3 different partitions each based on the X86-64 architecture. The overall supercomputer peak (respectively theoretical peak) performance is about 1,359 (respectively 1,6672) PFlops.

In what follows, we use the Curie thin nodes partition. Each node is composed of 2 Intel® Sandy-Bridge EP (E5-2680) processors and each processor has eight cores Intel® Xeon®.

For each target matrix presented in the Table 7.1, we propose to execute an ERAM with the α_{Def} restarting strategy, as this restarting strategy is the *reference*, in the sense that this restarting strategy is the most commonly used. During its execution, we apply the convergence analysis Algorithm 10. Based on its results, we remain the ERAM restarts where divergence or stagnation appears. The aim is to change the restarting strategy at these restarts, and observe the ERAM with mixed restarting strategies behavior.

To illustrate our purpose, we consider that at the i^{th} ERAM restart, the Algorithm 10 returned the *"divergence"* or *"stagnation"* status and was using so far the restarting strategy α_{Def}. At the restart $i+1$, we modify α_{Def} by another one, getting to $\alpha_X^{(i+1)}$ with the condition ${\alpha_{Def}}^{(i)} \neq \alpha_X^{(i+1)}$.

The restarting strategy gives a condition on restarting coefficients weight scheme, but we did not explain which Ritz eigen-information we use to compute the restarting vector with the mixed restarting strategy. Indeed, we have two possibilities to compute $v_1^{(i+1)}$:

- The first possibility is to compute $v_1^{(i+1)}$ using $\{\theta_j^{(i)}, u_j^{(i)}, resTr_j^{(i)}\}$ data.
- The second possibility is to compute $v_1^{(i+1)}$ using $\{\theta_j^{(k)}, u_j^{(k)}, resTr_j^{(k)}\}$ data such that:

$$\frac{||Au_j^{(k)} - u_j^{(k)}\theta_j^{(k)}||}{|\theta_j^{(k)}|} = \min_{k\in[1,i]_{\mathbb{N}}} (\max_{j\in[1,s]_{\mathbb{N}}} (\frac{||Au_j^{(k)} - u_j^{(k)}\theta_j^{(k)}||}{|\theta_j^{(k)}|})), \tag{7.1}$$

[1] A complete description of the Curie supercomputer can be found at http://www-hpc.cea.fr/fr/complexe/tgcc-curie.htm

In other terms, we use the Ritz eigen information coming from the k^{th} restart, where the convergence metric had the smallest value computed since the ERAM execution, therefore the best convergence scheme.

In what follows, we successively tested every restarting strategies such that it differs from α_{Def} and use the equation 7.1 to compute the new restarting vector. This choice is motivated by the fact that we aim to reduce the divergence and stagnation ERAM status, therefore using the last most accurate eigenpairs is more relevant.

We will compare the results with the ERAM using α_{Def} restarting strategy during its complete execution. This reference will be represented by a blue line during all this section.

7.2 The ERAM with Mixed Restarting Strategies Experimentations

7.2.0.1 $\mathbf{Ex11}_{Dense}$

We summarize the Algorithm 10 results in the Table 7.2.0.1. We indicate for both status *stagnation* and *divergence* their detection interval(s) and the restarts that provided (until the divergence or stagnation status) the most accurate Ritz eigenpairs.

	Stagnation			Divergence	
Interval	[21,24]	[28,31]	{35}	{10}	{13}
Lowest Residual	6	25	25	6	6

Table 7.2: The ERAM has $m = 15$, $s = 4$ and a CGSR orthogonalization process for the Arnoldi Method. We used 234 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. We summarize in this Table the results of the Algorithm 10.

We listed the Algorithm 10 results for the first 30 restarts, not for the complete ERAM execution. We recall that we want to intervene at the beginning of a divergence or stagnation status.

In what follows, we will switch α_{Def} by the other restarting strategies (presented in the Table 6.1.1) at the restarts presented in the table above. The Algorithm 10 results lead us to modify the restarting strategy at restarts 6 and 25 respectively.

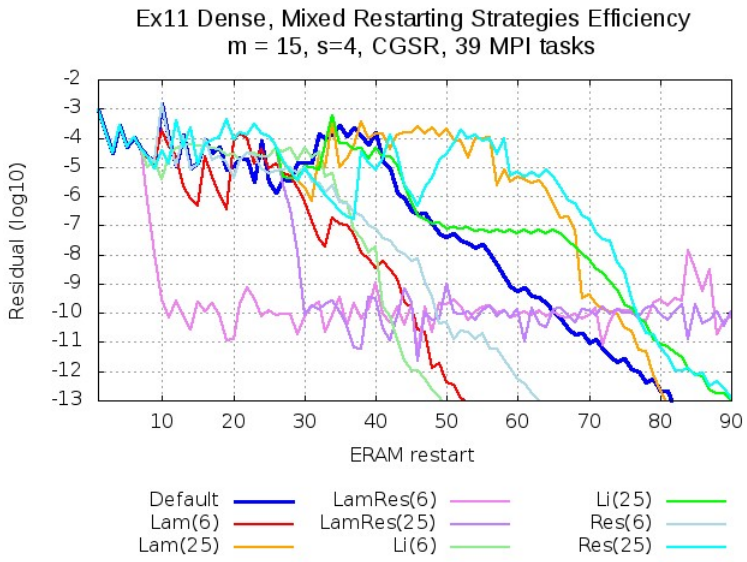

Figure 7.1: Ex11 Dense Matrix, α_{Def} Mixed Restarting Strategies.
The ERAM has $m = 15$, $s = \gamma = 4$, a CGSR orthogonalization process (s is the number of desired eigenpairs while γ is the number of eigenpairs used to compute the restarting vector). All lines on this figure are the dominant eigenvalue residuals associated to diverse ERAM executions. The blue line refers to the ERAM using α_{Def} as a single restarting strategy during its complete execution: its number of restart to reach the convergence (82) is the reference metric to estimate the ERAM with mixed restarting strategies efficiency. All other ERAMs start by using the restarting strategy α_{Def} as initial restarting strategy. We replaced α_{Def} by other restarting strategies at the restarts listed in the Table 7.2.0.1 above. We used 234 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. As a comparison, the SLEPc Krylov-Schur (respectively Arnoldi) solver using $m = 15$ converges at the 8^{th} restart and $m = 20$ converges at the 6^{th} restart (respectively 39 and 21 restarts).

We note that the mixed restarting strategies using α_{LaRes} present a considerable convergence but not sufficient to provide the Ritz eigenpairs with a satisfiable threshold: The ERAM with mixed restarting strategies stagnates then.

The second observation is that mixing α_{Def} with α_{La} (at restart 6) accelerates the original ERAM convergence compared to its original configuration: We recall that on Figure 6.2, the α_{La} had the worst convergence (converges at the 92^{th} restart), meaning that an inefficient restarting strategy -used alone- can be turned to an efficient one -if mixed-.

The third observation is that some restarting strategies *"force"* the ERAM behavior: we mean that the α_{Def} restarting strategy behavior is then completely annihilated by the new restarting strategy α_{NEW}, getting a behavior similarly of an ERAM using α_{NEW} during its complete execution. If we compare the Figure 7.1 with the Figure 6.2, the mixed restarting strategies $\alpha_{LaRes}(6)$, $\alpha_{La}(25)$, $\alpha_{Li}(6)$ and $\alpha_{Res}(25)$ have a similar convergence as the ERAM using respectively α_{LaRes}, α_{La}, α_{Li} and α_{Res}. However, this behavior is not the general case, as illustrated by the ERAM convergence using $\alpha_{Li}(25)$ (Figure 7.1).

From all the ERAM using a single restarting strategy (Figure 6.2) and mixed restarting strategies (Figure 7.1), it Results that the best scheme remains the ERAM using $EP = \{m = 20, s = 4, \gamma = 4, CGSR, \alpha_{LaRes}\}$.

The mixed restarting strategy have a considerable impact compared to the reference restarting strategy α_{Def}: note that the three mixed restarting strategies summarized in the Table 7.2.0.3 above saves restarts for both configurations using $m = 15$ and $m = 20$.

Mix. RS/Single RS	α_{Def} $m = 15$	α_{Def} $m = 20$	α_{Res} $m = 20$	α_{Li}, $m = 20$	α_{LiRes}, $m = 20$	α_{La} $m = 20$	α_{LaRes} $m = 20$
$\{m = 15, \alpha_{Li}, it = 6\}$	**1,63**	**1,27**	**1,2**	**0,63**	**2,98**	**1,88**	**0,33**
$\{m = 15, \alpha_{La}, it = 6\}$	**1,54**	**1,19**	**1,13**	**0,6**	**2,81**	**1,77**	**0,31**
$\{m = 15, \alpha_{Res}, it = 6\}$	**1,27**	**0,98**	**0,94**	**0,49**	**2,32**	**1,46**	**0,25**

Table 7.3: We compare the gain (in terms of number of restarts until convergence) of the ERAM with mixed restarting strategies with the ERAM using a single restarting strategy during its complete execution. Each column corresponds to one ERAM using a single restarting strategy. Each line corresponds to one execution of the ERAM using mixed restarting strategies (presented in Figure 7.1). This Table presents the gains (respectively losses) in terms of number of restarts until the convergence for the ERAMs with mixed restarting strategies versus the ERAMs using a single restarting strategy during their complete execution.

The total flop used for the Arnoldi method using the CGSR orthogonalization scheme is $m(2nnz(A) + 4nm + 4n)$ ([Dubois 2011a]) where $nnz(A)$ is the non-zeros elements of the matrix. Project the s $H_{m,m}$ eigenvectors onto the Krylov basis requires $2mns$ floating point operations and finally computing the residual associated to each Ritz eigenpair necessitates $4ns$ floating point operations. Therefore one ERAM restarts computes $m(2nnz(A) + 4nm + 4n + 2ns) + 4ns$ floating point operations. In this case, the matrix size is 10,000 and its number of non zeros elements is 275,991,771.

Mix. RS/Single RS	$\alpha_{Def}, m = 15$		$\alpha_{Def}, m = 20$	
	GFlop	Time	GFlop	Time
$\{m = 15, \alpha_{Li}, it = 6\}$	480	25,6 sec	505	2 min 57 sec
$\{m = 15, \alpha_{La}, it = 6\}$	450	24 sec	475	2 min 56 sec
$\{m = 15, \alpha_{Res}, it = 6\}$	285	15,2 sec	310	2 min 47 sec

Table 7.4: We compare the efficiency of the ERAM with mixed restarting strategies with the ERAM using a single restarting strategy during its complete execution. Each column corresponds to one ERAM using a single restarting strategy. Each line corresponds to one execution of the ERAM using mixed restarting strategies (presented in Figure 7.1). We summarized in this Table the GFlop and execution time saved by the mixed restarting strategies compared to the ERAM using α_{Def} for the global ERAM execution. We recall that the ERAM executions presented in this Table are executed with 234 MPI tasks on the PRACE Curie supercomputer. As a reference, the execution time of the ERAM using $\alpha_{Def}, m = 20$ (respectively $\alpha_{Def}, m = 15$) is 3 min 42 sec (respectively 1 min 8 sec).

The ERAM with mixed restarting strategies presented in the Table 7.2.0.1 gains are proportional to the number of restarts gained, if and only if we compare the ERAM execution using the same subspace size. We recall that the ERAM execution time per restart is constant whatever the restarting strategy for a fixed subspace size.

However, with different subspace size, the gain in terms of execution time is not linear: it depends on the data movement and the parallel communications, such parameters do not evolve linearly compared to the number of ERAM restarts and/or the number of MPI tasks used. This can be observed on the Table above, but it will be highlighted with the Diagonal-Band matrices in what follows. The ERAM execution time with respect to the subspace size depends on the parallel data distribution and the data movements (inside a node and across the global MPI communicator).

7.2.0.2 $Fission_{Dense}$

We execute the exact same tests with the $Fission_{Dense}$ matrix. We summarize in the following Table 7.2.0.2 the results obtained by the Algorithm 10 with an ERAM using the α_{Def} restarting strategy.

	Divergence	Stagnation
Interval	{2}	{6}
Lowest Residual	1	1

Table 7.5: The ERAM has $m = 10$, $s = \gamma = 4$, a CGSR orthogonalization process and the restarting strategy α_{Def}. We used 400 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. We summarize in this Table the results of the Algorithm 10.

We summarize on the Figure 7.2.0.2 the results obtained with the mixed restarting strategies:

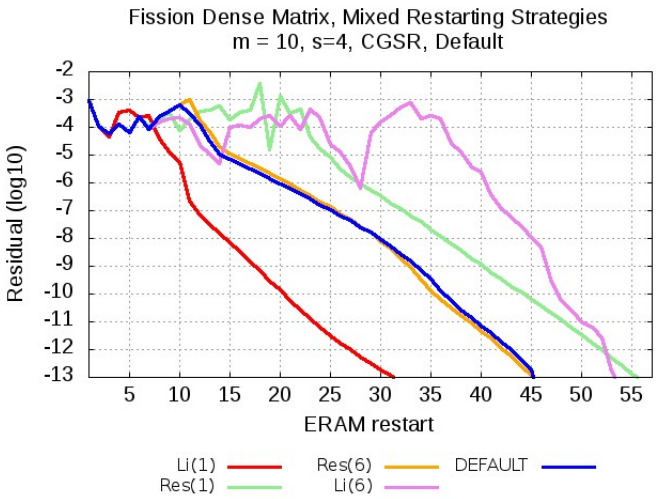

Figure 7.2: *Fission* Dense Matrix, Mixed Restarting Strategies Convergence with respect to the number of restarts.
The ERAM has $m = 10$, $s = \gamma = 4$, a CGSR orthogonalization process and the restarting strategy α_{Def}. We present the dominant eigenvalue residuals. We used 400 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. As a comparison, the SLEPc Krylov-Schur (respectively Arnoldi) solver converges at the 9^{th} (respectively 28^{th}) restart. The blue line refers to the ERAM using α_{Def} as a single restarting strategy during its complete execution: its number of restart to reach the convergence is the reference metric to estimate the ERAM with mixed restarting strategies efficiency. All ERAMs with mixed restarting strategies started with the α_{Def} restarting strategy. The ERAM with the mixed restarting strategy $\alpha_{Li}(6)$ provides similar results as the SLEPc-Arnoldi eigensolver.

The results obtained with the mixed restarting strategy are variable for this matrix. As an illustration, mixing α_{Def} with α_{Li} saves 15 restarts compared to the original ERAM. The complete execution of the ERAM with the mixed restarting strategy $\alpha_{Li}(6)$ saves 30 GFlop operations (for the complete ERAM execution) compared to the original ERAM execution.

One may note that the ERAM using the α_{Li} restarting strategy during its complete execution (using the same ERAM parameters $CGSR, m = 10$ and $s = \gamma = 4$, cf Figure 6.3) did not provide the convergence. In fact, its numerical performance was poor. Indeed, we make the same observation for the α_{Res} restarting strategy. We observed the same behavior on the previous matrix $Ex11_{Dense}$ whose configuration and properties are very different from the $Fission_{Dense}$ matrix, leading to the conclusion that this observation is not *"matrix-dependant"*.

7.2.0.3 $\mathbf{Rim}_{Dense}$

We experimented the same tests on the Rim Dense matrix. We summarize in the Table 7.2.0.3 the Algorithm 10 results:

	Stagnation					
Interval	[9,15]	[28,32]	[43,52]	[57,66]	[69,78]	[103,126]
Lowest Residual	2	23	40	40	40	102

Table 7.6: The ERAM has $m = 15$, $s = \gamma = 4$, a CGSR orthogonalization process and the restarting strategy α_{Def}. We used 480 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. We summarize in this Table the results of the Algorithm 10.

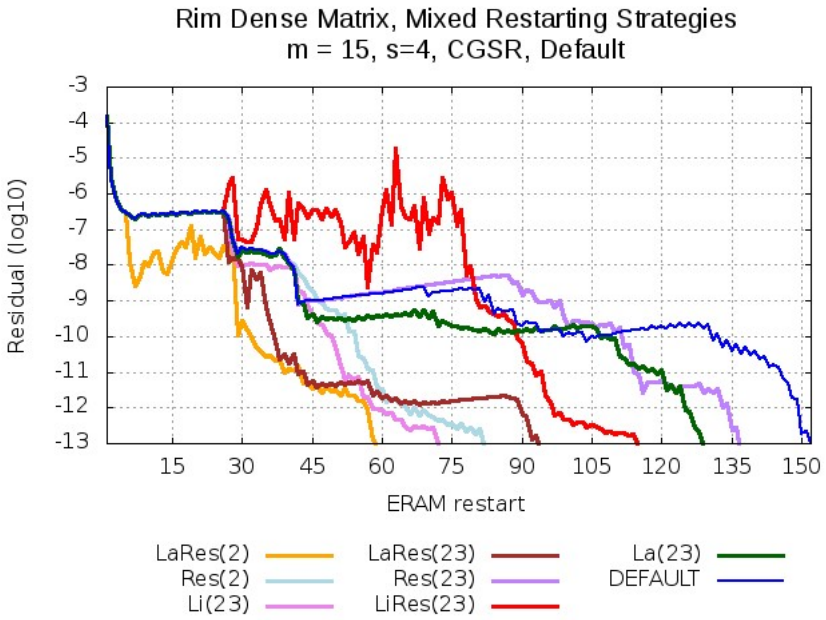

Figure 7.3: *Rim* Dense Matrix, Mixed Restarting Strategies Convergence.
The ERAM has $m = 15$, $s = \gamma = 4$ (s is the number of desired eigenpairs while γ is the number of eigenpairs used to compute the restarting vector), a CGSR orthogonalization process and the restarting strategy α_{Def}. We present the dominant eigenvalue residuals. We replaced α_{Def} by other restarting strategies at the restarts listed in the Table 7.2.0.1. We used 480 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. The blue line refers to the ERAM using α_{Def} as a single restarting strategy during its complete execution: its number of restart to reach the convergence is the reference metric to estimate the ERAM with mixed restarting strategies efficiency. All ERAMs with mixed restarting strategies started with the α_{Def} restarting strategy. As a comparison, the SLEPc Krylov-Schur (respectively Arnoldi) solver converges at the 14^{th} (respectively 30^{th}) restart.

We summarize the mixed restarting strategy efficiency compared to the ERAM using a single restarting strategy (cf Figure 6.8) in the following Table.

Mix. RS/Single RS	α_{Def}	α_{Res}	α_{Li},	α_{La}	α_{LaRes}
$\{\alpha_{LaRes}, it = 2\}$	**2,59**	**1,07**	**1,17**	**2,33**	**2,33**
$\{\alpha_{Res}, it = 2\}$	**2,05**	**0,85**	**0,93**	**1,85**	**1,85**
$\{\alpha_{Li}, it = 23\}$	**1,88**	**0,78**	**0,85**	**1,69**	**1,69**
$\{\alpha_{LaRes}, it = 23\}$	**1,63**	**0,67**	**0,74**	**1,47**	**1,47**
$\{\alpha_{Res}, it = 23\}$	**1,28**	**0,53**	**0,58**	**1,15**	**1,15**
$\{\alpha_{LiRes}, it = 23\}$	**1,17**	**0,48**	**0,53**	**1,05**	**1,05**
$\{\alpha_{La}, it = 23\}$	**1,1**	**0,46**	**0,5**	**0,99**	**0,99**

Table 7.7: We compare the gains (in terms of number of restarts until convergence) of the ERAM with mixed restarting strategies with ERAM using a single restarting strategy during its complete execution. Each column corresponds to one ERAM using a single restarting strategy (presented in Figure 6.8). Each line corresponds to one execution of ERAM using mixed restarting strategies (presented in Figure 7.3). The ERAM using as a single restarting strategy α_{LiRes}, did not converged.

This matrix illustrates perfectly the mixed restarting strategies benefits: With the ERAMs using a single restarting strategy, the restarting strategies such as α_{LaRes} and its twin α_{La} were not efficient. Mixed with α_{Def}, we reach the convergence at the 58 restarts, which is 2,3 times faster than the original configuration (blue line). We recall this is *"free"* gain, in terms of parallel communications and operations costs.

In the following Figure 7.4, we summarized the ERAM with mixed restarting strategies gain (respectively lost) in terms of Floating point operations compared to the ERAM using a single restarting strategy during its complete execution. It results from this Table that the α_{LaRes} at the second restart ($\alpha_{LaRes}(2)$ on Figure 7.3) mixed with α_{Def} can provide a better ERAM convergence compared to each original configuration.

Globally, the ERAM using α_{Res} and α_{Li} as single restarting strategies are not drastically improved by mixed restarting strategies in this case. Nevertheless, we recall that the mixed restarting strategies start their process using the α_{Def} restarting strategy, therefore as a faire comparison we should use ERAM with mixed restarting strategies starting with α_{Res} and α_{Li} respectively. In this context we may have observed a gain of the ERAM with mixed restarting strategies.

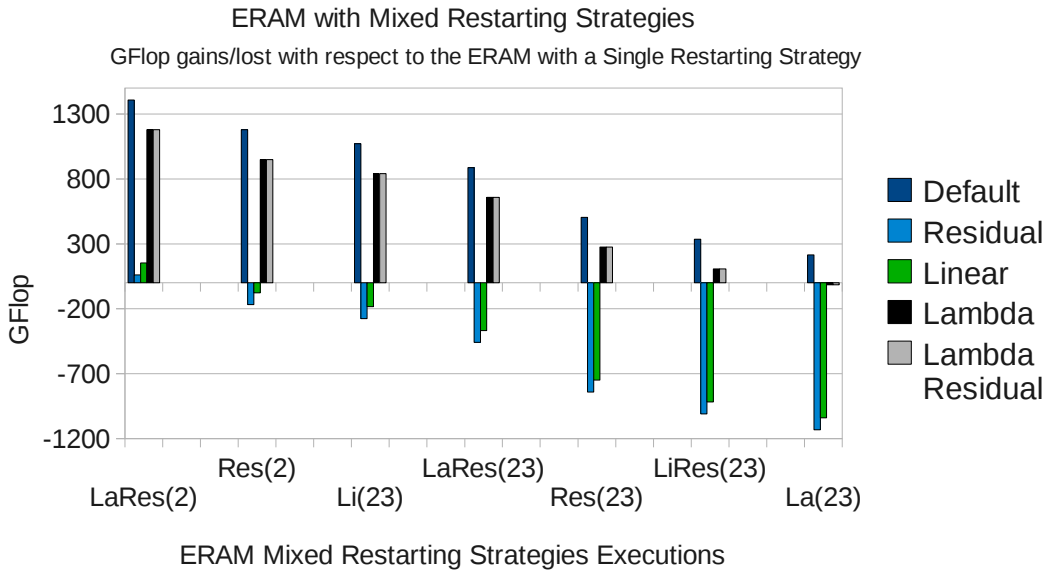

Figure 7.4: Rim_{Dense} Mixed Restarting Strategies GFlop Gain/Losts.

We present on this figure the GFlop gains and losses (for the complete ERAM execution) of the ERAM with mixed restarting strategies compared to the ERAM using a single restarting strategy during their complete execution. Abscissa refers to each ERAM execution using the mixed restarting strategy. For each ERAM with mixed restarting strategies, we compare the gains and losses in terms of GFlop operations for the complete ERAM execution with respect to an ERAM using a single restarting strategy. Each stripe refers to an ERAM using a single restarting strategy (see the legend). We recall that all the ERAMs using the mixed restarting strategies start with α_{Def}.

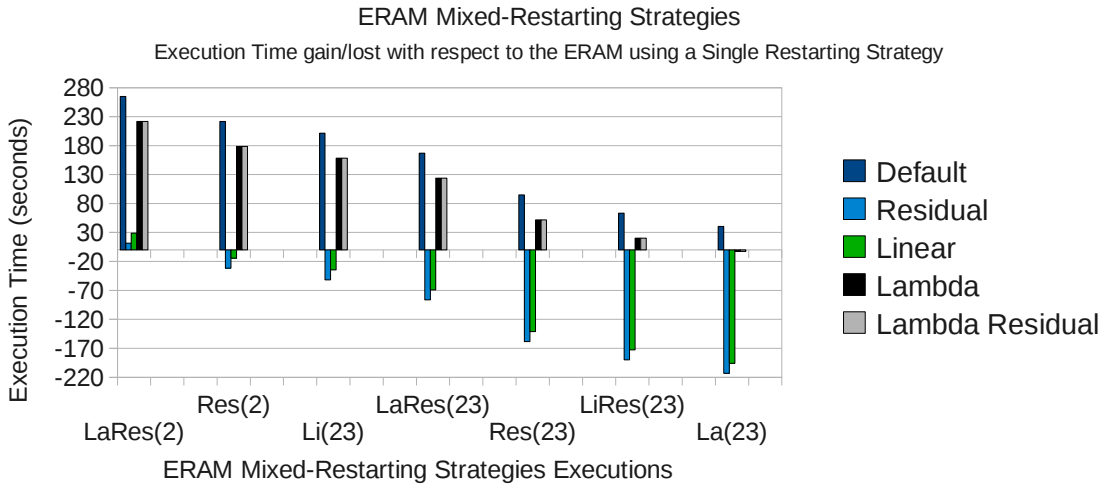

Figure 7.5: Rim_{Dense} Mixed Restarting Strategies Time (seconds) Gains/Losts.
We present on this figure the Parallel Execution Time (seconds) gains and losses (for the complete ERAM execution) of the ERAM with mixed restarting strategies compared to the ERAM using a single restarting strategy during their complete execution. Each ERAM presented on this figure has been executed using 480 MPI tasks on the thin nodes of the PRACE Curie supercomputer. Abscissa refers to each ERAM execution using the mixed restarting strategy. For each ERAM with mixed restarting strategies, we compare the gains and losses in terms of time execution (seconds) for the complete ERAM execution with respect to an ERAM using a single restarting strategy. Each stripe refers to an ERAM using a single restarting strategy (see the legend). We recall that all the ERAMs using the mixed restarting strategies start with α_{Def}. The best performance is obtained by switching α_{Def} onto α_{LaRes} at the second restart: it accelerates every original ERAM configurations.

7.2.0.4 Diagonal Band 2^{15}

The last but not the least, we used the mixed restarting strategies with the diagonal-band matrix $Spec_{2^{15}}$.

Based on the results presented in the section 6.2.0.5, here are our expectations from the mixed restarting strategies:

- The mixed restarting strategies using α_{LaRes} will considerably improve the ERAM convergence: This observation is based on the Figures presented in 6.2.0.5, on which we observed the fast convergence of the ERAM using the α_{LaRes} restarting strategy. We expect that the ERAM using the α_{LaRes} restarting strategy mixed with α_{Def} will behave as the ERAM using α_{LaRes} during its complete execution.

  These observations are strengthened by the spectrum of $Spec_{2^{15}}$, we already know thanks to the previous chapter that α_{LaRes} is the most efficient restarting strategy with very large real modulus eigenvalues.

- The mixed restarting strategies using α_{LiRes} and α_{Li} will provide poor performances. Firstly, the α_{LiRes} restarting strategy has a chaotic behavior for most of the target matrices presented in this thesis, except the Mixtank_new one. Secondly, the matrices such as Ex11 and the diagonal-band matrices generated in this thesis are more sensitive to the restarting strategies that use the Ritz eigen information to compute their restarting coefficients.

We recall that the diagonal-band matrix presented in this section **is not the same as** the one presented in section 6.2.0.5. Nevertheless, their spectrum are similar leading to the same (or very similar) behavior of the ERAM restarting strategies regarding the diagonal-band matrices generated in this thesis.

We first summarize in the Table 7.2.0.4 the ERAM (using a single restarting strategy during its complete execution) number of restart until convergence with respect to the restarting strategies. We compute the 3 dominant eigenpairs with a CGSR orthogonalization process.

This is not surprising to observe the spectacular efficiency of α_{LaRes} compared to the other restarting strategies, as its efficiency has been emphasized with the diagonal-band 2^{16} matrix in the section 6.2. We recall that the diagonal-band 2^{16} and diagonal-band 2^{15} matrices do not have the same spectrum, nevertheless the eigenvalues real modulus and their distribution are similar.

We did not expect to observe that the ERAM solving the diagonal-band matrix 2^{15} eigenproblem would be that sensitive to the subspace-size value: The performance of the ERAM using $m = 150$, $m = 190$ and $m = 200$ using the restarting strategy α_{LaRes} are comparable. The α_{Res} and α_{Li} restarting strategies are also sensitive to the subspace size: the number of restarts until convergence does not necessarily decrease with the higher m values.

m	α_{Def}	α_{Res}	α_{Li},	α_{LiRes},	α_{La}	α_{LaRes}	SLEPc Krylov-Schur	SLEPc Arnoldi
150	#	#	#	#	#	12	8	44
160	#	#	#	#	#	17	7	24
170	#	#	#	#	#	14	7	29
180	**249**	**155**	**211**	#	**215**	14	6	44
190	**144**	**237**	**88**	#	**177**	12	6	25
200	**88**	**100**	**110**	#	**98**	10	6	21

Table 7.8: We compare the number of restart until convergence of ERAMs with a single restarting strategy. Every ERAMs presented in this table has been executed using 256 MPI tasks on the thin nodes of the PRACE Curie supercomputer. The # symbols mean that the ERAM did not reached the convergence, the desired threshold is 10^{-14}. We used $s = \gamma = 3$ and a CGSR orthogonalization process.

For each ERAM execution presented in the Table 7.2.0.4 we applied the Algorithm 10 to characterize the ERAM convergence. According to the Algorithm 10 results, we will switch the restarting strategy α_{Def} by another one at the restarts presented in the Table 7.2.0.4 below. We choose to study many configurations for the diagonal-band matrix as the results presented below highlight the necessity to choose the *"most adapted"* restarting strategy.

ERAM with Mixed-Restarting Strategies

m		Stagnation					Divergence			
150	Interval	{6}	{79}	[13,14]			{10}	{20}	{37}	[16,18]
	Lowest Residual	4	61	7			7	15	25	15
160	Interval	{12}	{35}	[15,16]	[27,28]		{9}			
	Lowest Residual	8	28	8	26		8			
170	Interval	{12}	[8,9]	[18,19]	[25,26]	[34,37]	{11}	{14}		
	Lowest Residual	10	4	10	10	27	10	10		
180	Interval	[17,19]					{11}	{36}		
	Lowest Residual	7					7	35		
190	Interval	[22,25]					4	9		
	Lowest Residual	16					3	8		
200	Interval	[8,9]	29							
	Lowest Residual	6	27							

Table 7.9: The ERAM has $s = \gamma = 3$, a CGSR orthogonalization process and the restarting strategy α_{Def}. We used 256 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. We summarize in this Table the results of the Algorithm 10.

In what follows, we detailed the ERAM with mixed restarting strategies using m=150 results with respect to the ERAM using a single restarting strategy (presented in the Table 7.2.0.4).

The two tables below present the parallel execution time in second saved (respectively lost) thanks to the ERAM with mixed restarting strategies (lines) compared to the ERAM using a single restarting strategy during its complete execution (columns).

Single / Mixed	α_{Def}			α_{Res}			α_{Li}		
m=150	180	190	200	180	190	200	180	190	200
$\{\alpha_{LaRes}, it = 3\}$	**46,33**	**26,64**	**15,55**	**27,84**	**45,55**	**18,03**	**38,86**	**15,25**	**20,09**
$\{\alpha_{LaRes}, it = 6\}$	**46,18**	**26,49**	**15,4**	**27,7**	**45,4**	**17,88**	**38,71**	**15,11**	**19,95**
$\{\alpha_{LaRes}, it = 14\}$	**44,57**	**24,88**	**13,79**	**26,08**	**43,79**	**16,27**	**37,1**	**13,49**	**18,33**
$\{\alpha_{LaRes}, it = 24\}$	**42,96**	**23,27**	**12,17**	**24,47**	**42,18**	**14,65**	**35,48**	**11,88**	**16,72**
$\{\alpha_{Res}, it = 3\}$	**23,74**	**4,05**	-7,04	**5,26**	**22,96**	-4,56	**16,27**	-7,33	-2,49
$\{\alpha_{Res}, it = 14\}$	**15,82**	-3,87	-14,96	-2,66	**15,04**	-12,48	**8,35**	-15,25	-10,41

Table 7.10: Diagonal Band 2^{15} Matrix, Mixed Restarting Strategies Execution Time Gains/Losses.

We compare the parallel execution time saves and losses of the ERAMs with mixed restarting strategies versus the ERAMs using a single restarting strategy. Every ERAMs presented in this table has been executed using 256 MPI tasks on the thin nodes of the PRACE Curie supercomputer. The ERAMs execution time saves/losses is in seconds.

Single / Mixed	α_{La}			α_{LaRes}					
m=150	**180**	**190**	**200**	150	160	170	180	190	200
$\{\alpha_{LaRes}, it = 3\}$	**39,64**	**33,35**	**17,61**	-0,88	**0,25**	-0,12	**0,11**	-0,2	-0,57
$\{\alpha_{LaRes}, it = 6\}$	**39,5**	**33,2**	**17,47**	-1,03	**0,1**	-0,27	-0,03	-0,35	-0,72
$\{\alpha_{LaRes}, it = 14\}$	**37,88**	**31,59**	**15,85**	-2,64	-1,51	-1,88	-1,65	-1,96	-2,33
$\{\alpha_{LaRes}, it = 24\}$	**36,27**	**29,98**	**14,24**	-4,25	-3,12	-3,49	-3,26	-3,57	-3,95
$\{\alpha_{Res}, it = 3\}$	**17,06**	**10,76**	-4,97	-23,47	-22,34	-22,71	-22,47	-22,79	-23,16
$\{\alpha_{Res}, it = 14\}$	**9,14**	-2,84	-12,89	-31,39	-30,26	-30,63	-30,39	-30,71	-31,08

Table 7.11: Diagonal Band 2^{15} Matrix, Mixed Restarting Strategies Execution Time Gains/Losses.

We compare the parallel execution time saves and losses of ERAMs with mixed restarting strategies versus ERAMs using a single restarting strategy. Every ERAMs presented in this table has been executed using 256 MPI tasks on the thin nodes of the PRACE Curie supercomputer. ERAMs execution time saves/losses is in seconds.

Most of the original ERAM configurations are improved by the mixed restarting strategies. The ERAM using α_{LaRes} as a single restarting strategy is the only exception, as its results are already very good.

As expected, the ERAM using α_{LaRes} as a mixed restarting strategies provide the best performances. The ERAM using α_{Res} as a mixed restarting strategy results are

more heterogeneous: the best gain is 22,96 seconds while the worst performance losses 31,08 seconds.

One may note that the ERAM parallel execution time gains/losses presented in the tables 7.2.0.4 and 7.2.0.4 do not evolve linearly with the subspace size. This is all the more true due to the random behavior of the ERAM with the Diagonal Band 2^{15} Matrix. The ERAM numerical performances may be better by using a smaller subspace (such as presented in the Table 7.2.0.4). Additionally to the parallel execution time, we recall that the memory storage and floating point operations are also reduced thank to a smaller subspace size. The comparison of such gains regarding the Diagonal Band 2^{15} Matrix are summarized in the Table B.0.2.5 and can be consulted in the appendix.

We executed many other tests with the subspace size 150, 160, 170, 180, 190 and 200 respectively. More results can be found in the annexe B.0.2.5 where we summarize, for each ERAM configuration, the ERAM with mixed restarting strategies efficiency (in terms of number of restarts until convergence) with respect to the ERAM using a single restarting strategy presented in the Table 7.2.0.4).

Conclusion

In this chapter, we studied the mixed restarting strategies: by mixed, we mean that the ERAM may use two or more restarting strategies during its execution.

This study focus on the "Default" restarting strategy as this one remains the classic one among the scientific literature.

From each target matrix results, we could highlight some behaviors of the mixed restarting strategies. Whatever the target matrix, we observed that a *"wrong"* restarting strategy can be turned to a very efficient one thanks to the mixed restarting strategies: this point is largely positive, meaning that we increase the probability to find a good combination of mixed restarting strategies.

Secondly, some mixed restarting strategies can behave similarly to the last restarting strategy used: the last one may annihilated all the other restarting strategies impact and behave similarly as an ERAM using it as a single restarting strategy. That was especially the case with α_{LaRes} restarting strategy for the matrices $Ex11_{Dense}$ or the diagonal-band matrices.

Finally the gain obtained with the mixed restarting strategies are free in terms of parallel communications, parallel operations and memory storage.

Thanks to the convergence Algorithm and the ERAM restarting strategies behavior, we could study the ERAM with mixed restarting strategies.

This chapter linked the convergence Algorithm to the restarting strategies efficiency: altogether, we can arrive to a *"free"* amelioration of the ERAM convergence.

However, getting such gains implies to choose the right restarting strategy and at the right time: this illustrates the complexity of the mixed restarting strategies.

The results presented in this section shown the efficiency of the mixed restarting strategy, but also the random behavior according to each matrix and each ERAM configuration, therefore there is still not a better restarting strategy among all the one presented in this thesis that would afford a gain for all ERAMs. This lead us to look for restarting strategy auto-tuning so as to choose mixed restarting strategies combinations according to ERAM numerical performances.

The next step is to find a smart-tune algorithm so as to get this free efficiency and the next chapter details our approach to build a restarting strategy smart-tuning ERAM algorithm.

Chapter 8

Toward an ERAM Restarting Strategies Auto-Tuning

The previous chapter highlighted the mixed restarting strategies efficiency according to the numerical convergence evaluation of the ERAM. We are now interesting by automating the mixed restarting strategies during the iterations.

8.1 The ERAM with Auto-Tuned Restarting Strategies

We presented in the Chapter 6 the influence of the ERAM restarting strategies. Indeed, the ERAM convergence is very sensitive to the ERAM restarting strategies, leading to a considerable gain (or lost) in terms of number of restarts to reach the convergence for the desired eigenpairs at the desired threshold.

Such gains are free in terms of parallel computation costs, as using a restarting strategy has no impact neither on the parallel operations/execution time nor on the memory storage. Each restarting strategy presented in this thesis only reuses the existing Ritz eigen information, therefore *"ready-to-use data"*.

Among all the test matrices we used to study the ERAM restarting strategy influence, we could emphasized some trends of the restarting strategies behavior with respect to the dominant eigenvalues distribution. Nevertheless, this trends are really limited as on the other side, some matrices spectrums could not be correlated to the restarting strategy efficiency.

Due to the possible gains in terms of number of restarts to reach the convergence thanks to the restarting strategy, we studied the ERAM with mixed restarting strategies in the Chapter 7. The objective of this chapter was to study the efficiency of the ERAM with mixed restarting strategy, id est an ERAM starting with a restarting strategy fixed by the user (in this study, we choose the uniform weight scheme as the initial restarting strategy). The idea is to use the ERAM convergence Algorithm (presented in the Chapter 4) so as to detect the divergence and stagnation status of the ERAM during the execution time. Then, we re-executed the same ERAM (with exactly the same input parameters) and changed at the indicated restart the initial restarting strategy by a new one and observe the ERAM convergence behavior with respect to the ERAM using a single restarting strategy during its complete execution.

The results presented in the Chapter 7 lead us to study all the possibilities to get to an ERAM with auto-tuned restarting strategies. As the ERAM with mixed restarting strategies was manually changed, the possibility of mixing many restarting strategies was

limited. This emphasizes the need to auto-tune this process, leading to a auto ERAM that would change itself its restarting strategy during its execution, based on the analysis of its convergence.

This chapter explores many possibilities of auto-tuning tools to improve the ERAM convergence and focuses on the restarting strategies auto-tuning.

8.2 On the Choose of the ERAM Auto-Tuned Parameters

The previous chapter presented the mixed restarting strategy idea and its *"proof of concept"*. As we *"manually"* changed the restarting strategies, we emphasized that we could not experiment many restarting strategies combinations:

At each stagnation or divergence status, we can choose among 5 restarting strategies, therefore the combinations of multiple restarting strategies are very large and can not be tested manually.

In this chapter, we aim to find mixed restarting strategies combinations automatically, meaning that such mechanism will be executed at the runtime execution, without any intervention of the user. As a final aim, we want to provide an auto-Tuned ERAM with mixed restarting strategies.

Building an auto-Tuned ERAM with mixed restarting strategies implies to answer to the three following questions:

- *Question 1:* **"WHEN"** shall we modify the ERAM parameter(s)?
- *Question 2:* **"WHAT"** shall we modify among the ERAM parameter(s)?
- *Question 3:* **"HOW"** shall we modify the ERAM parameter(s)?

Thanks to the previous chapters, we almost solve the problem:

- *Question 1:* The convergence metric determines the moment when we shall intervene to improve the ERAM convergence. Thanks to the Algorithm 10, we answered this question at the Chapter 4.
- *Question 2:* There are several possible answers to this question. In fact, each ERAM parameter modification could lead to an amelioration of the convergence, but the way to succeed will be more or less costly regarding the ERAM parallel execution. Let's expand further this study. Among the ERAM parameters, we listed the following ones: $EP = \{m, s, \gamma, orthogonalization, \alpha_X\}$.
  - As a first observation, s can not be modified as this parameter is fixed by the user. The only possibility to modify it would be by considering a MRAM such that each ERAM member would compute a part of the s desired eigenpairs. In this case, each RAM would have its own s_k ($\forall k \in [1, \mu]$) such that $\sum_{k=1}^{\mu} s_k \geq s$.

- Modifying the subspace size is probably the most straightforward option. Indeed, we know that the m value leads the ERAM convergence (according to [Saad 2011]): therefore the most basic idea would be to increase the subspace size in the case of a bad convergence detection. Nevertheless we recall that m also leads, in the opposite direction as the numerical convergence, the parallel execution time. The bigger m is, the more we will be faced to global and blocking communications, default-cache operations etc... We emphasized the parallel costs and energy consumption issue of the Krylov subspace size value in the section 3.6.1.

  Such auto-tuning methods are widely known for the GMRES solver, whose subspace size value is dynamically fixed according to the GMRES convergence. Many research has been done on this topic, we invite the reader to consult [Baker 2005], [Baker 2009], [Aquilanti 2011b] and [Katagiri 2012] as an illustration.

  The same concept could be applied to the ERAM, if and only if the convergence metric is adapted. However, we recall that due to the GMRES algorithm, only the subspace size m value can be modified (among the GMRES input parameters, otherwise we could modify the preconditioner as an example, but such possibilities will not be detailed in this thesis). The restarting vector is fixed by the method, therefore, regarding the GMRES parameters, the *"freedom degree"* is very limited compared to the ERAM. As the ERAM offers many other potential parameters whose modification(s) has no impact on the parallel execution time, we will not consider the subspace size auto-tuning in this thesis.

- α_X is the parameter we have chosen to focus on. Among the restarting strategies presented in the Table 6.1.1, each of them have an impact on the ERAM restarting strategy **but absolutely not on the parallel execution time**. Even better, the restarting strategies we presented in this thesis re-use (or not) the ERAM Ritz eigen information, therefore there is absolutely neither additional communications nor operations to use a (new) restarting strategy. Thanks to the chapters 6 and 7 we emphasized the ERAM convergence acceleration with respect to the (mixed) restarting strategies, leading to a gain at no *parallel computing costs*.

- The γ parameter can be fixed according to the ERAM and the Ritz eigenpairs accuracy. We may observe that computing more than s eigenpairs can increase the s desired Ritz eigenpairs convergence. We recall that such scheme is not surprising as the Restarted Arnoldi Method tends to favor the outer eigenpairs convergence. Nevertheless, it requires to project $\gamma - s$ additional eigenvectors onto the Krylov basis and especially to compute their associated residuals (which is quite costly as it requires the Euclidean norm-2 computation).

- The orthogonalization scheme is one interesting possibility. Modifying the ERAM orthogonalization with respect to the ERAM convergence is one option

among others and is particularly interesting if we switch the CGSR by the CGS, as it saves many parallel blocking and global communications. Such auto-tuning scheme has the same benefits and drawbacks as the subspace size value: Getting more numerical accuracy will require more parallel operations that will slow down the ERAM parallel execution, but on the other side, remaining in this configuration with a very slow or critical convergence is not satisfiable. Finally, we must add to the previous purpose that the opposite behavior **may** be observed, as an illustration, switching the CGSR by a CGS may be more efficient.

Many auto-tuning options exist regarding the orthogonalization scheme, but those are applied -again- in the case of the GMRES. Nevertheless, such as the subspace size auto-tuning, these orthogonalization auto-tuning schemes could be applied to the ERAM.

Among all the ERAM parameters listed above, the one that optimizes the ratio *convergence amelioration versus parallel performances downgrading* is definitely the restarting strategy α_X.

➢ *Question 3:* That's the question we'll answer in this Chapter.

8.2.1 The ERAM with Mixed Restarting Strategies Auto-Tuning Prerequisite

We previously presented the mixed restarting strategy using firstly α_{Def}. The conclusion presented in the previous chapter for mixed restarting strategies starting from α_{Def} remain available for every restarting strategies presented in the Table 6.1.1.

The following Figure 8.1 illustrates this purpose:

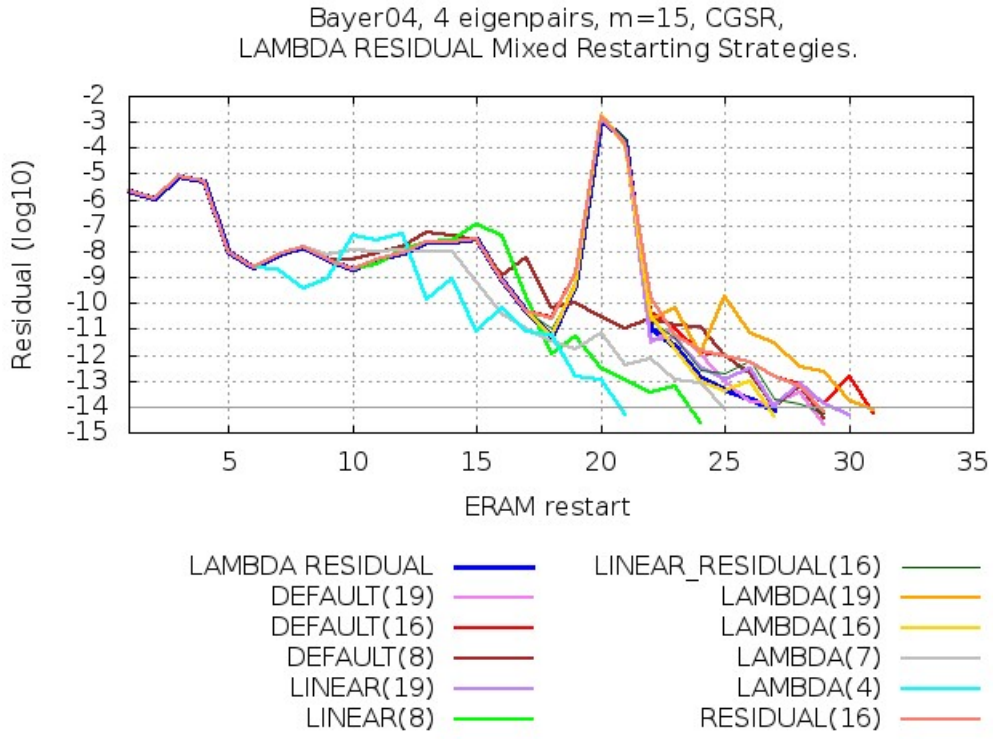

Figure 8.1: Bayer04 Matrix, α_{LaRes} Mixed Restarting Strategies.

The Bayer04 matrix is issued from [Davis]. The reference is the blue line denoted by α_{LaRes}. The ERAM has $m = 15$, $s = \gamma = 4$, a CGSR orthogonalization process and the restarting strategy α_{LaRes} as initial restarting strategy. We present the dominant eigenvalue residuals. We replaced α_{LaRes} at the restarts pointed out by the Algorithm 10. These ERAMs are executed sequentially. For information purposes, the ERAM using α_{LaRes} (blue line) as a single restarting strategy was the best scheme among all the ERAMs members using a single restarting strategy. This recall that the ERAM with mixed restarting strategy can be more efficient than the best ERAM using a single restarting strategy.

Such precision remains very important: this means that getting an ERAM with restarting strategies auto-tuning can be efficient whatever the input restarting strategy is.

Secondly, this implicitly implies that a MERAM can be composed of auto tuned ERAM, all with different restarting strategy: then, all ERAM can be improved by the restarting strategies auto tuning.

This prerequisite is very important as a restarting strategy auto-tuning efficient with the α_{Def} restarting strategy only would seriously limit the possibilities of the ERAM with mixed restarting strategies.

8.2.2 The ERAM with Mixed Restarting Strategies Auto-Tuning Choice

As illustrated in the previous chapters, there are several possibilities to dynamically change the restarting strategy:

- *Choice 1*: We recall that during the ERAM execution, we still measure the ERAM convergence thanks to the Algorithm 10. One way to select efficiently a new restarting strategy would be to pick a new restarting strategy among the one that were used during the *"convergence"* or *"high convergence"* status.

  Therefore, picking a new restarting strategy implies first to test one by one each restarting strategy, secondly to create an *"historic"* of the efficient restarting strategies. This algorithm will be detailed in what follows.

- *Choice 2*: We randomly choose the restarting strategy among the available ones. This scheme may seem surprising but it is far from that. The principal motivation is that a restarting strategy used during a complete ERAM execution may be inefficient and nevertheless very efficient if mixed with other restarting strategies. This random behavior can not be predicted, therefore choosing randomly the restarting strategies allows to test more mixed restarting strategies combinations and a higher probability to find a good mixed restarting strategies profile. We shall shortly return to this point later.

- *Choice 3*: We modify the restarting strategies of an ERAM thanks to the MERAM. We will illustrate this purpose with a simple example. Let's consider a MERAM with μ ERAMs components, each of them has different ERAM parameters. In the case that ERAM_k (where $k \in [1, \mu]$) receives more accurate results from the ERAM_l (where $l \in [1, \mu]$ and $l \neq k$), we could easily imagine that ERAM_k picks the restarting strategy of the ERAM_l. This choice presents one advantage but a major drawback from our perspective.

  The MERAM has the benefit to have many ERAMs components, therefore the independent study of the ERAMs convergence allows to elect with precision the best ERAM among the panel members. It is then intuitive to fix the ERAMs to the "best restarting strategy scheme". Then comes the drawback: Such scheme implies that all the ERAMs members will have in the short term the same restarting strategy, leading to a MERAM whose *"freedom degree"* is only the subspace size. From our perspective, this is not the optimum scheme, as optimum combinations can be found if ERAM are independent but cooperative components, meaning that each of them is focusing -thanks to pertinent ERAMs parameters- on a part of the desired spectrum, and not altogether on the complete desired spectrum.

8.3 A Basic Mixed Restarting Strategy Tuning Addressed to the ERAM

According to the previous results presented in this Thesis, we aim to dynamically change the ERAM restarting strategy at the restarts pointed out by the Algorithm 10. Basically, if a stagnation or divergence status is detected, we modify the ERAM restarting strategy with respect to the choices mentioned above.

We present the most straightforward version of the ERAM with the Mixed Restarting Strategies Tuning Algorithm 13:

Algorithm 13 Basic Restarting Strategy Tuning ERAM

Input: $A \in \mathbb{C}^{n\times n}$, $v_1 \in \mathbb{C}^n$, $\varepsilon_{Arnoldi} > 0$, $\varepsilon > 0$, $m \in [1,n]_{\mathbb{N}}$, $s \in [1,m]_{\mathbb{N}}$, $max_{ERAM} \in \mathbb{N}$, $\gamma \in [s,m]_{\mathbb{N}}$, $(f_{inf}, f_{sup}) \in]0,1[^2$, $max_{count} \in \mathbb{N}^*$, α_X, $orthogonalization$

1: $v_1 = \frac{1}{||v_1||} v_1$

2: **while** $(\varepsilon \geq \max_{j\in[1,s]_{\mathbb{N}}} \{resTr_j\})$ or $(max_{ERAM} > i)$ **do**

3: Execute m-step Arnoldi Method using $\{A,\ v_1,\ \varepsilon_{Arnoldi},\ m,\ orthogonalization\}$

4: Solve the eigen problem $H_m Y_m = \Theta_m Y_m$

5: $resTh_j = |h_{m+1,m} y_{m,j}|, \forall j \in [1,\gamma]_{\mathbb{N}}$

6: $U_{n,\gamma} = V_{n,m} Y_{m,\gamma}$

7: $resTr_j = ||Au_j - u_j\theta_j||, \forall j \in [1,\gamma]_{\mathbb{N}}$

8: $Conv_{status}$ =Multi-Levels Convergence Algorithm$(res_{CV}^{(i)}, res_{CV}^{(i-1)}, f_{inf}, f_{sup}, max_{count})$

9: **if** $Conv_{status}$ Diverges or $Conv_{status}$ Stagnates **then**

10: switch $\alpha_X^{(i)}$ by $\alpha_{new}^{(i+1)}$ such that $\alpha_X^{(i)} \neq \alpha_{new}^{(i+1)}$

11: **end if**

12: $v_1 = \sum_{j=1}^{\gamma} \alpha_{new,j}^{(i+1)} u_j^{(i)}$

13: **end while**

Output: $U_{n,s} \in \mathbb{C}^{n\times s}, \Theta_s \in \mathbb{C}^s, ResTr_s \in (\mathbb{R}^+)^s$

Respectively to the previous comments, the step 10 of the Algorithm 13 can be detailed as follows:

- *Choice 1*: Try successively (random order) each restarting strategy among $\{\alpha_{Def}, \alpha_{Res}, \alpha_{Li}, \alpha_{LiRes}, \alpha_{La}, \alpha_{LaRes}\}$ once, then pick α_{new} among the restarting strategies that could provide the convergence or high convergence status respectively to the Algorithm 10.

  To do so, we keep the restarting strategy and its associated convergence status.

In fact, we choose to keep the complete ERAM parameters, as the restarting strategy auto-tuning can be extended. In this thesis, we choose to intervene on the restarting strategy parameter only, but here is absolutely no limits if we aim to modify several ERAM parameters altogether at one *"critical"* ERAM restart. In this case, this is not just one restarting strategy scheme that we must conserve, but the complete ERAM scheme that could provide the convergence status.

Algorithm 14 ERAM Convergence Snapshot

1: **if** $Conv_{status}$ is Convergence or Highly Convergence **then**

2: Keep the parameters $\{Conv_{status}, m, \alpha_{new}, \gamma, orthogonalization\}$

3: **end if**

4: Order the parameters with respect to the highest Convergence

Output: ERAM Parameters Historic

- *Choice 2*: Pick randomly a new restarting strategy among $\{\alpha_{Def}, \alpha_{Res}, \alpha_{Li}, \alpha_{LiRes}, \alpha_{La}, \alpha_{LaRes}\}$ without considering the historic of the convergence.

8.3.1 Basic Mixed Restarting Strategy Tuning Results

For all the following matrices, we imposed some additional conditions to dynamically choose the restarting strategies. Such modifications intervene thanks to a previous study of Algorithm 13.

- First of all, we deleted the restarting strategy α_{LiRes}, meaning that at the step 10 of the Algorithm 13 we choose a new restarting strategy among $\{\alpha_{Def}, \alpha_{Res}, \alpha_{Li}, \alpha_{La}, \alpha_{LaRes}\}$ instead of $\{\alpha_{Def}, \alpha_{Res}, \alpha_{Li}, \alpha_{LiRes}, \alpha_{La}, \alpha_{LaRes}\}$. This choice is motivated by the chaotic behavior of α_{LiRes} as observed in the previous chapters. The α_{LiRes} restarting strategy performance is very poor if mixed with others and tends to considerably disrupt the ERAM convergence, this motivated our choice to delete it from the ERAM with the restarting strategies auto-tuning. Nevertheless we allow the ERAM to **start (and start only)** its process with α_{LiRes}.

- Secondly, we impose to the ERAM to keep a restarting strategy during 5 restarts at least. This choice is motivated by the fact that the Krylov subspace direction is changed thanks to the new restarting strategy: it takes few restarts to be stabilized and therefore determine if the new restarting strategy could provide a pertinent convergence direction. Deleting this option may lead to successive restarting strategies changes and possibly avoid a good restarting strategy that could not proved its efficiency.

- Finally, we suspend the restarting strategy change (id est the step 10 of the Algorithm 13) if and only if the ERAM convergence is close to the desired threshold. This order is relative to the ERAM execution itself. We compute the amplitude between the residual computed at the very first restart and the desired threshold:

$$amplitude = \frac{\varepsilon}{\max_{j=1,s}(resTr_j^{(1)})}, \tag{8.1}$$

  If the current residual $\max_{j=1,s}(resTr_j^{(i)})$ is inferior to $\frac{3}{4}amplitude$, then we lock the restarting strategy until the convergence. We observed thanks to previous results that in most of the case, changing the restarting strategy while we remain close to the desired threshold tends to disrupt rather than accelerate or maintain the ERAM convergence.

All the points above have been decided thanks to initial restarting strategies tests.

In what follows, we present the ERAM with Mixed Restarting Strategy Auto-Tuning (Algorithm 13) using as a first restarting strategy respectively $\{\alpha_{Def},\alpha_{Res},\alpha_{Li},\alpha_{LiRes},\alpha_{La},\alpha_{LaRes}\}$.

We first present the results of the Algorithm 13 by using the **Choice 1** (id est by considering the convergence evolution with respect to the ERAM) and explain why the **Choice 2** has not been retained.

For each figure presented in this section, we indicated thanks to colored stripes the number of restarts until convergence for each ERAM using as a single restarting strategy $\{\alpha_{Def},\alpha_{Res},\alpha_{Li},\alpha_{LiRes},\alpha_{La},\alpha_{LaRes}\}$ respectively.

The target matrices properties are summarized in the following Table:

	Size	nnz	Source
Fission_{Dense}	10,000	99,980,003	Issued from 5.2.1.1 and CEA neutronic applications
$\text{Fission}_{2^{15}Dense}$	16,384	286,402,691	Issued from 5.2.1.1 and CEA neutronic applications
Ex11_{Dense}	16,614	275,991,771	Issued from A.0.2.1
$\text{Mixtank\_new}_{Dense}$	29,957	897,361,938	Issued from A.0.2.2
Rim_{Dense}	22,560	508,908,483	Issued from A.0.2.3
$\text{Spec}_{2^{16}}, M_t{}^4$	32,768	296,327	Issued from 5.3.1

Table 8.1: The Target Matrices to Test the ERAM with Mixed Restarting Strategies Auto-Tuning.

We executed the Algorithm 13 with the target matrices listed in the Table 8.3.1 above. In what follows, we pick the new restarting strategy according to the Choice 1, id est among the restarting strategies that proved their efficiency.

As a reference metric, we compare for each executions presented below the number of restart until convergence: we recall that the ERAM execution time per restart remains the same whatever the restarting strategy is. As the Algorithm 13 is not deterministic (therefore we can not obtain an accurate execution time measure according to the numerical results presented below due to the write in file operations), the number of restart until convergence is the most pertinent metric.

For each following figure, we will present the Algorithm 13 results applied to every ERAM restarting strategies studied in this thesis. For one restarting strategy, we will execute several ERAM using the Algorithm 13 to emphasize the nondeterministic behavior of the Algorithm itself and the different convergence evolution with respect to the mixed restarting strategies fixed by the Algorithm 13.

For a better reading and understanding, we present the ERAM using Algorithm 13 and starting with a specific restarting strategy separately. As an illustration, we will present on the same graphic the ERAM starting with α_{Def} and α_{Res} and then separately α_{Li} and α_{LiRes}.

We present on the Figure 8.2 the ERAM with Mixed Restarting Strategy Auto-Tuning (Algorithm 13).

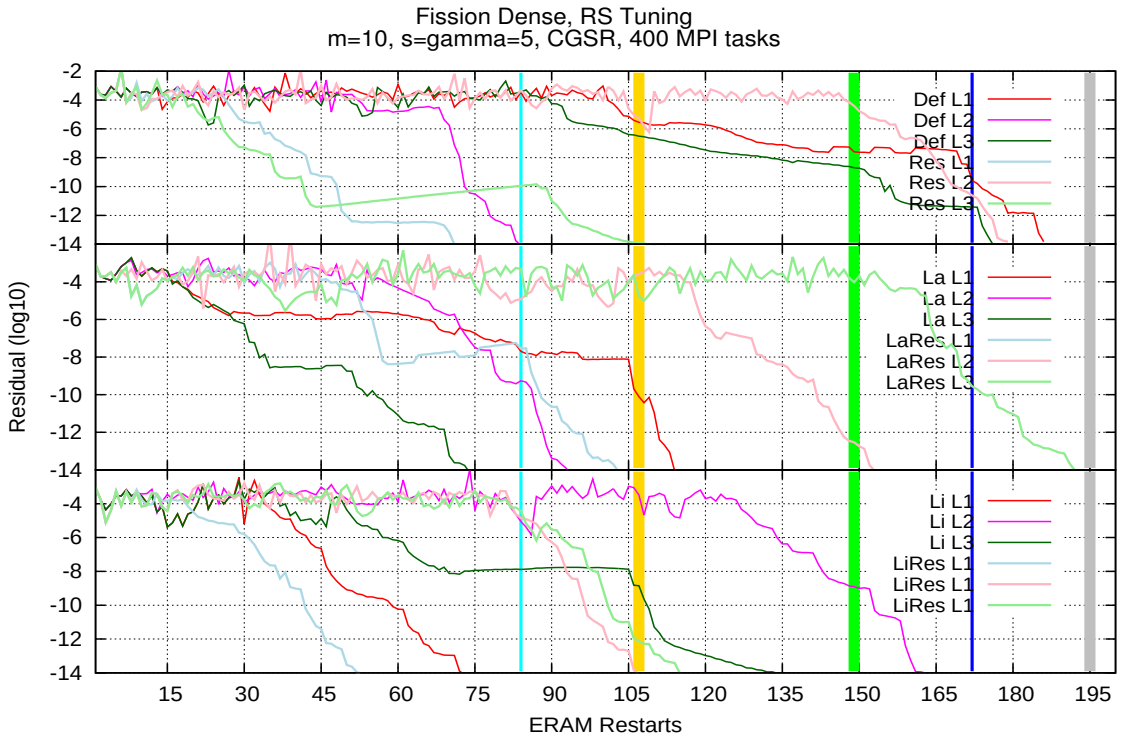

Figure 8.2: *Fission* Dense Matrix, Mixed Restarting Strategies Tuning *Choice 1*.
The ERAM has $m = 10$, $s = \gamma = 5$, a CGSR orthogonalization process. We present the lowest eigenvalue residuals. We used 400 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. Each stripe color corresponds to the ERAM using respectively $\{\alpha_{Res}, \alpha_{LiRes}, \alpha_{Li}, \alpha_{Def}, \alpha_{LaRes}\}$ restarting strategy during its complete execution (α_{La} did not converge). We presented separately the twin restarting strategies (α_{Def},α_{Res} then α_{La}, α_{LaRes} and finally α_{Li},α_{LiRes}). For each restarting strategy, we present 3 different executions of the Algorithm 13 to show its nondeterministic behavior. As an illustration, Def L1 refers to one execution of the Algorithm 13 starting with the α_{Def} restarting strategy.

- The ERAM with mixed restarting strategies starting with the α_{Res} restarting strategy is not that efficient compared to the ERAM using α_{Res} during its complete

execution. The Res L1 line (light-blue) saves 14 restarts compared to the original configuration while Res L2 (light-pink) and ResL3 (light-green) lost 70 and 15 restarts respectively. As a reminder, the ResL1 line performs better than the best ERAM using a single restarting strategy during its complete execution.

- The results are very different for the ERAM starting with the α_{LiRes} restarting strategy. Indeed, the ERAM with mixed restarting strategies LiRes L1, LiRes L2 and LiRes L3 provides whether a great acceleration (LiRes L1, light-blue line) whether comparable performances. There are no significant degradation of the initial performance, such as it was the case for the ERAM starting with the α_{Res} restarting strategy. In what follows, we will observe that the ERAM with mixed restarting strategy auto-tuning is relatively efficient when it starts with the α_{LiRes} restarting strategy. Note that in this case, the ERAM LiRes L1 offers the best performances among the one presented on the Figure 8.2: The best gain compared to the other ERAM with mixed restarting strategy (respectively using a single restarting strategy) is from 1,3 to 3,4 (respectively from 1,5 to 3,5) times faster.

- The performance of the ERAm starting with α_{Li} are very similar to the α_{LiRes} detailed above.

- The ERAM with mixed restarting strategies starting with α_{Def} performances are not very statisfiable, especially for the Def L1 (red line) and Def L3 (dark-green line): one may note that the ERAM has a chaotic behavior during 90 restarts, which is not satisfiable from our perspective. We will return to this point in what follows.

- Finally, the ERAM with mixed restarting strategy starting with α_{LaRes} ameliorates the ERAM convergence for every configurations presented on the Figure 8.2. Nevertheless, the ERAM remains chaotic during too much restarts, such behavior should be avoided.

In what follows, we will detail just a subset of the restarting strategies combination that could provide dramatic accelerations and on the opposite maintain the ERAM chaotic behavior. The chaotic behavior is marked by a completely random choice of the restarting strategy: this case appears when none of the restarting strategy has clearly been identified as a good performer. On the other side, we distinguish some restarting strategy combinations that provided the good performances:

- Def L2: α_{La},α_{Def},α_{La},
- Res L2: α_{Li},α_{Def},α_{Res},
- La L2: α_{Li},α_{La},α_{Li},α_{La}
- LiRes L1: α_{Def},α_{Li},α_{Res},α_{Li}

- Li L2: α_{Res},α_{Def},α_{Li}.

These restarting strategy combinations greatly accelerated the ERAM performances, we will compare these combinations with the following target matrices.

We recall that the execution time per restart for all the ERAM execution presented above remains similar, as the execution time per restarts is independent from the restarting strategy used. Whatever the restarting strategy is, the execution time to compute a restarting coefficient of the α_{Res} restarting strategy is the same as the α_{Li} restarting strategy one. As a reference, the average execution time per restart for the ERAMs configurations presented above is 2,11 seconds.

According to the previous paragraph, the best ERAM with auto-tuned restarting strategy (id est ERAM LiRes L1 on the Figure 8.2) saves 1 min 10 seconds (respectively 4 min 23 seconds) compared to the best (respectively the worst) ERAM using a single restarting strategy.

We executed the same tests with the $Fission_{2^{15}}$ Dense Matrix and present the results on the Figure 8.3.

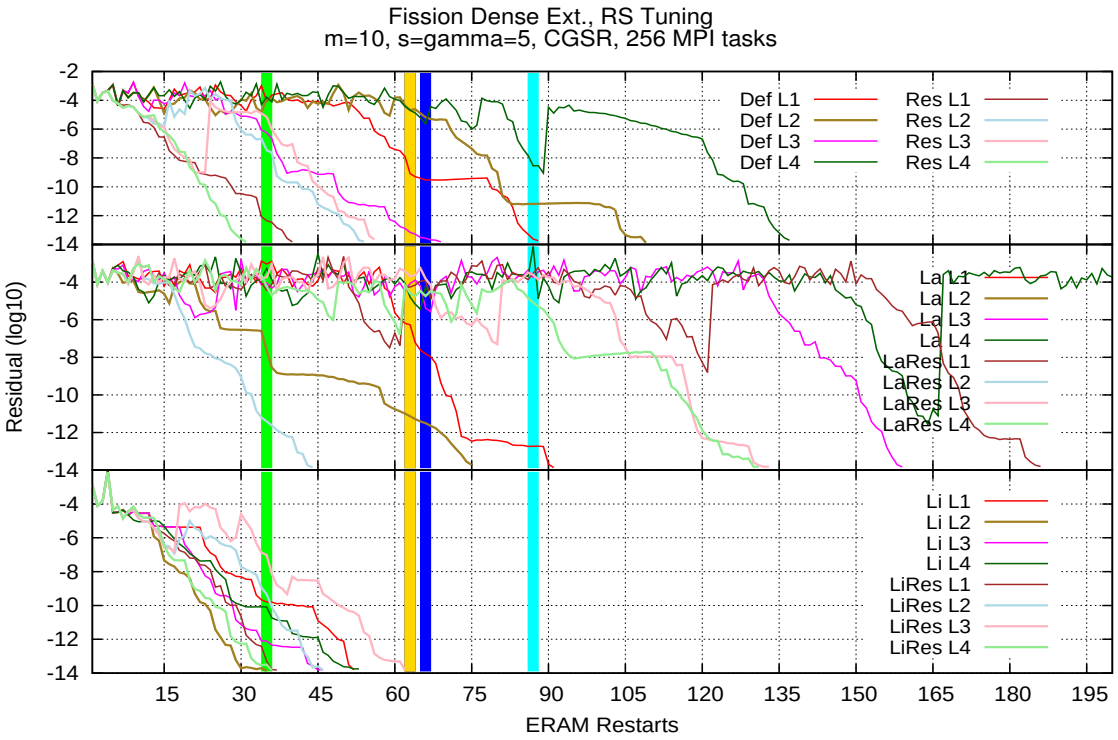

Figure 8.3: *Fission*$_{2^{15}}$ Dense Matrix, Mixed Restarting Strategies Tuning *Choice 1*. The ERAM has $m = 10$, $s = \gamma = 5$, a CGSR orthogonalization process. We present the dominant eigenvalue residuals. We used 256 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. Each stripe color corresponds to the ERAM using respectively $\{\alpha_{Li}$,α_{LiRes},α_{La},α_{Def} and $\alpha_{Res}\}$ restarting strategy during its complete execution (the α_{LaRes} did not converge). We presented separately the twin restarting strategies (α_{Def},α_{Res} then α_{La}, α_{LaRes} and finally α_{Li},α_{LiRes}). For each restarting strategy, we present 4 different executions of the Algorithm 13 to show its nondeterministic behavior. As an illustration, Li L1 refers to one execution of the Algorithm 13 starting with the α_{Li} restarting strategy.

The $Fission_{2^{15}}$ Dense Matrix convergence is more spectacular than the *Fission* Dense Matrix presented earlier. Most of the ERAM with mixed restarting strategy auto-tuning provided great accelerations and the ERAM with chaotic behavior is much more limited. However, due to the very good performance of the ERAM using α_{Li} as a single restarting strategy, it is more arduous to find better configurations with mixed restarting strategy than the best ERAM using a single restarting strategy.

Globally, the best ERAM with mixed restarting strategy (Res L4, light-green line) acceleration is from 1 to 5,6 (respectively from 1,16 to 3) compared to the other ERAM with mixed restarting strategy (respectively using a single restarting strategy).

As a comparison with the *Fission* Dense Matrix, we list some successful restarting strategy combinations:

- Res L4: α_{Def},α_{Li},
- LaRes L2: α_{Res},α_{Def},α_{Li},
- La L3: α_{Li},α_{Def},α_{La},α_{Li}
- Li L2: α_{Li},α_{La}.

We note that these combinations remain close from the *Fission* Dense Matrix, which is not surprising as the $Fission_{2^{15}}$ Dense Matrix is an extension of the *Fission* Dense Matrix.

We continue this study with the Ex11 Dense matrix, the results are presented on the Figure 8.4. As a reminder, this matrix was particularly sensitive to the restarting strategies α_{La} and α_{LaRes}. We expect for this matrix that reaching the convergence will be pretty easy and fast, as the Algorithm 13 will test α_{La} and α_{LaRes} restarting strategies, therefore we are sure that these will be identified as efficient restarting strategies and chosen by the Algorithm 13.

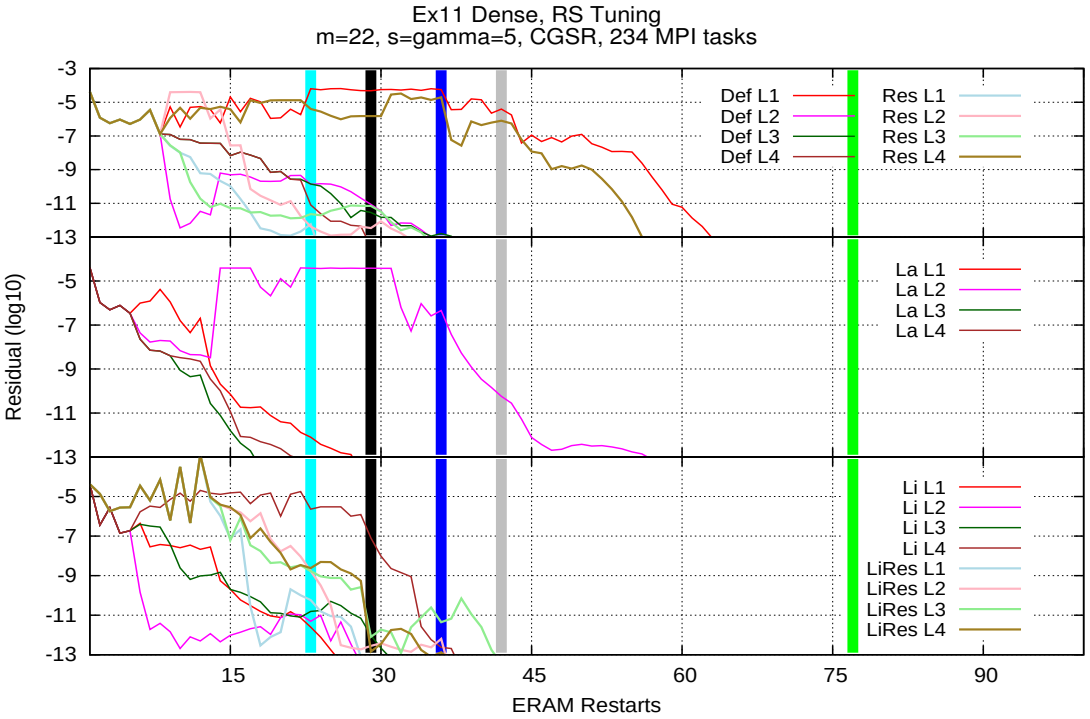

Figure 8.4: *Ex*11 Dense Matrix, Mixed Restarting Strategies Tuning *Choice 1*. The ERAM has $m = 22$, $s = \gamma = 5$, a CGSR orthogonalization process. We present the dominant eigenvalue residuals. We used 234 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. Each stripe color corresponds to the ERAM using respectively $\{\alpha_{Res}$,α_{La},α_{Def},α_{LaRes} and $\alpha_{Li}\}$ restarting strategy during its complete execution (α_{LiRes} did not converge). We presented separately the twin restarting strategies (α_{Def},α_{Res} then α_{La}, α_{LaRes} and finally α_{Li},α_{LiRes}). For each restarting strategy, we present 4 different executions of the Algorithm 13 to show its nondeterministic behavior. As an illustration, Li L1 refers to one execution of the Algorithm 13 starting with the α_{Li} restarting strategy.

The ERAM with mixed restarting strategy auto-tuning performances are very satisfying compared to the original configurations. We did not present the ERAM with α_{LaRes}

as initial restarting strategy as the Algorithm 10 does not indicate any stagnation or divergence status for this configuration, therefore, there is no need to change the restarting strategy according to the Algorithm 13.

As mentioned above, the α_{LaRes} restarting strategy provided great accelerations, most of the dramatic convergences observed on the Figure 8.4 are due to the α_{LaRes} restarting strategy. As its convergence is pretty smooth, the ERAM keeps it until it reaches the desired threshold. We recall that we imposed to the ERAM with mixed restarting strategy auto-tuned to keep the restarting strategy when the current eigenpairs threshold remains close to the desired threshold. As α_{LaRes} provides almost "immediate" convergence, it remains as the restarting strategy until the end of the process.

The gain of the best ERAM with mixed restarting strategy auto tuning La L3 (dark-green line) is from 1 to 4 (respectively 1,3 to 4,9) faster than the other ERAM with mixed restarting strategy auto (respectively using a single restarting strategy).

We present on the Figure 8.5 several ERAM executions solving the Mixtank_new Dense matrix eigenvalue problem by using the Algorithm 13 and *Choice 1* to pick the new restarting strategy (meaning that we first execute all of the restarting strategies once). We recall that this matrix was quite slow to converge, and only the α_{LiRes} restarting strategy provided the convergence.

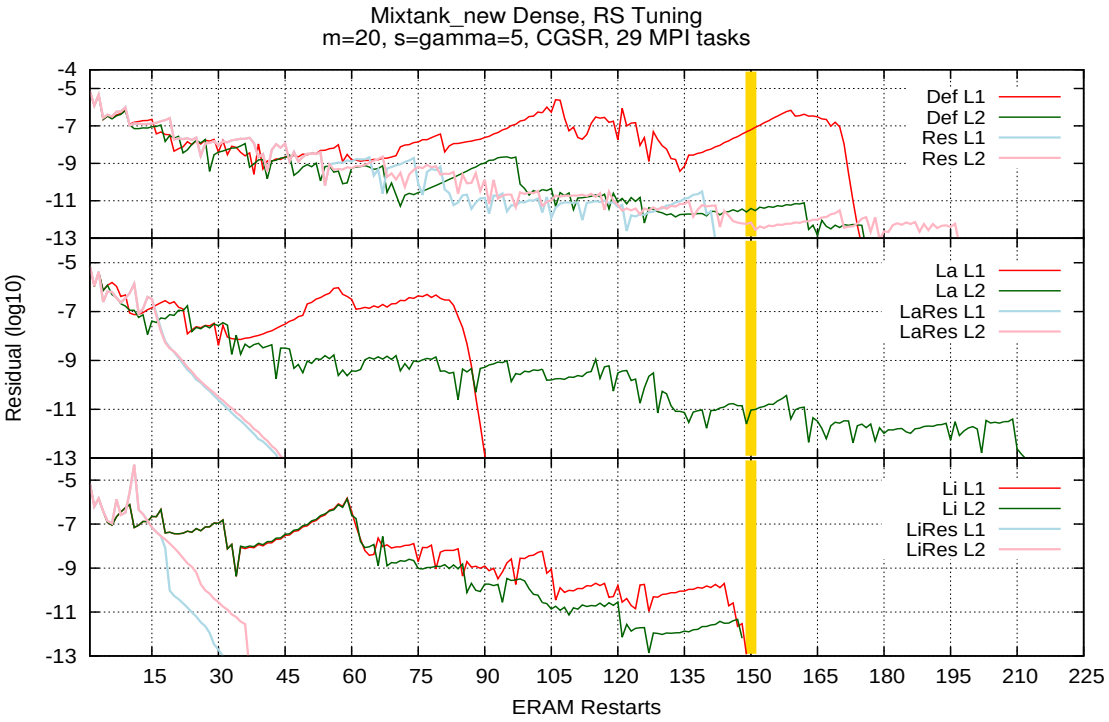

Figure 8.5: Mixtank_new Dense Matrix, Mixed Restarting Strategies Tuning *Choice 1*. The ERAM has $m = 20$, $s = \gamma = 5$, a CGSR orthogonalization process. We present the dominant eigenvalue residuals. We used 29 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. The stripe color corresponds to the ERAM using $\{\alpha_{LiRes}\}$ restarting strategy during its complete execution (this configuration is the only one that could converge). We presented separately the twin restarting strategies (α_{Def},α_{Res} then α_{La}, α_{LaRes} and finally α_{Li},α_{LiRes}). For each restarting strategy, we present 2 different executions of the Algorithm 13 to show its nondeterministic behavior. As an illustration, La L1 refers to one execution of the Algorithm 13 starting with the α_{La} restarting strategy.

The results obtained for the Mixtank_new Dense Matrix are more mixed. On one side, great accelerations can be obtained thanks to the ERAM with mixed restarting strategies, on the other side, the slow convergence scheme is still hard to avoid. The dramatic acceleration observed on the Figure 8.5 such as LaRes L1, LaRes L2, LiRes L1 and LiRes L2 are obtained thanks to the participation of many restarting strategy: in this case, there are no combinations that can be identified, such as it was the case for the Fission matrices or the Ex11 Dense one.

We present on the Figure 8.6 several ERAM executions solving the Rim Dense matrix eigenvalue problem by using the Algorithm 13. The Rim Dense matrix has a profile relatively close to the Mixtank_new matrix (however, the spectrums are different): the ERAM converges pretty slowly with many plateau and stagnation phasis. As a reminder, only two ERAM could provide the convergence (using the restarting strategies α_{Def} and α_{Li} respectively).

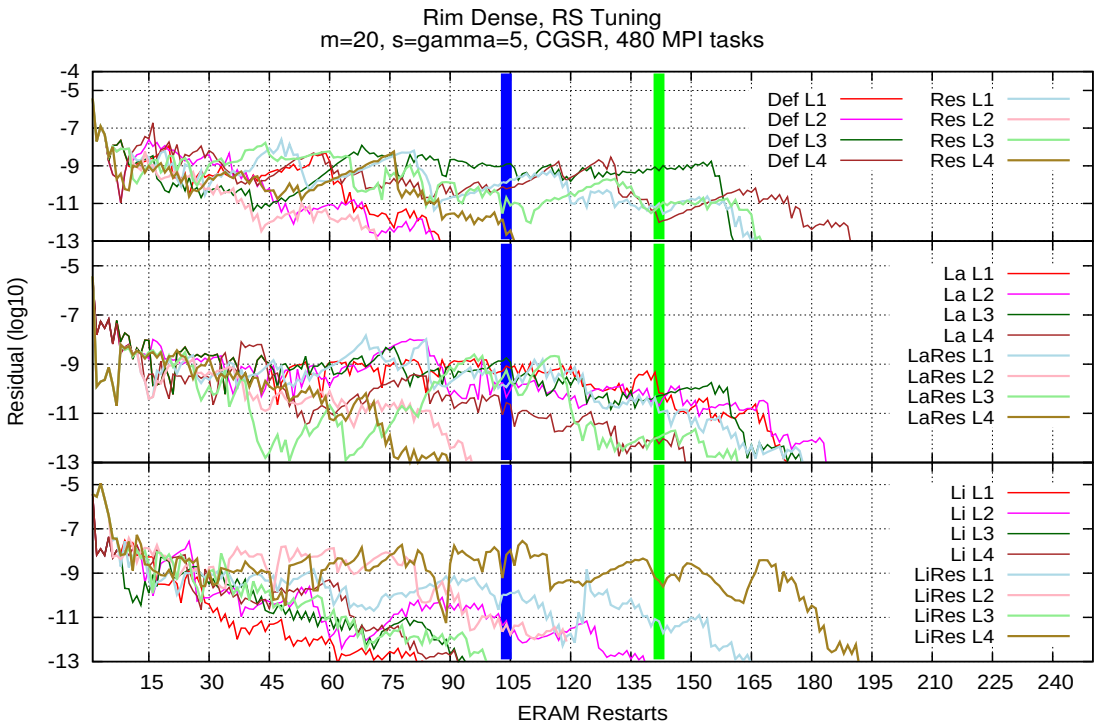

Figure 8.6: *Rim* Dense Matrix, Mixed Restarting Strategies Tuning *Choice 1*.
The ERAM has $m = 20$, $s = \gamma = 5$, a CGSR orthogonalization process. We present the dominant eigenvalue residuals. We used 480 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. Each stripe color corresponds to the ERAM using respectively $\{\alpha_{Def}$ and $\alpha_{Li}\}$ restarting strategy during its complete execution. The ERAMs using α_{LaRes}, α_{LiRes}, α_{La} and α_{Res} respectively as a single restarting strategy did not converge. We presented separately the twin restarting strategies (α_{Def},α_{Res} then α_{La}, α_{LaRes} and finally α_{Li},α_{LiRes}). For each restarting strategy, we present 4 different executions of the Algorithm 13 to show its undeterministic behavior. As an illustration, LaRes L1 refers to one execution of the Algorithm 13 starting with the α_{LaRes} restarting strategy.

Similarly to the Mixtank_new Dense Matrix, the Rim Dense eigen system is pretty hard to converge. Nevertheless, the ERAM with mixed restarting strategy provides sat-

isfiable performances, as most of the initial schemes did not converged. We do not detail the restarting strategy combinations in this case, as we do not observe drastic acceleration as it was the case for the Fission matrices or the Ex11 one for example. The convergence is accelerated but still the ERAM chaotic behavior is not fully avoided for this matrix.

The last but not the least, we executed the Algorithm 13 on the Diagonal-Band 2^{16} Matrix by using different subspace size configurations. We recall that this matrix was very sensitive to the α_{LaRes} restarting strategy behavior: therefore, such as the Ex11 matrix, we expect that most of the ERAM with mixed restarting strategy will reach the convergence thanks to the α_{LaRes} restarting strategy.

We also recall that the Diagonal-Band 2^{16} Matrix numerical convergence did not increased with the Krylov subspace size value: Indeed, we observed better performances with small subspace size in the previous section, especially for the restarting strategy α_{LaRes}. The spectrum of the Diagonal-Band 2^{16} matrix is very different from the Ex11 Dense matrix, but their sensitivity to the α_{LaRes} restarting strategy remains a common point: indeed, they both have eigenvalues whose real modulus are very large. Such as the Ex11 Dense matrix, we expect to obtain very good performances for this matrix as the Algorithm 13 will identify the restart α_{LaRes} as efficient and it will be picked up as a new restarting strategy.

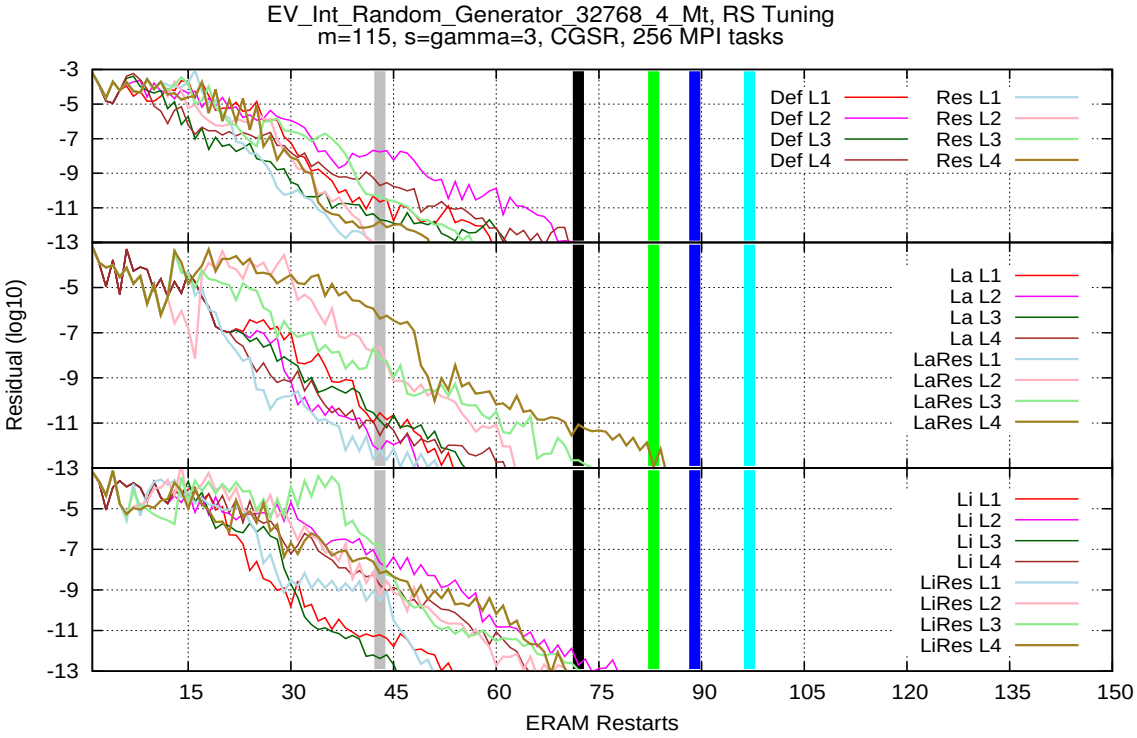

Figure 8.7: Diagonal Band 2^{16} Matrix, Mixed Restarting Strategies Tuning *Choice 1*. The ERAM has $m = 115$, $s = \gamma = 3$, a CGSR orthogonalization process. We present the dominant eigenvalue residuals. We used 256 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. Each stripe color corresponds to the ERAM using respectively $\{\alpha_{LaRes}$,α_{La}, α_{Li},α_{Def},$\alpha_{Res}\}$ restarting strategy during its complete execution. The ERAMs using α_{LiRes} as a single restarting strategy did not converge. We presented separately the twin restarting strategies (α_{Def},α_{Res} then α_{La}, α_{LaRes} and finally α_{Li},α_{LiRes}). For each restarting strategy, we present 4 different executions of the Algorithm 13 to show its undeterministic behavior. As an illustration, LiRes L2 refers to one execution of the Algorithm 13 starting with the α_{LiRes} restarting strategy.

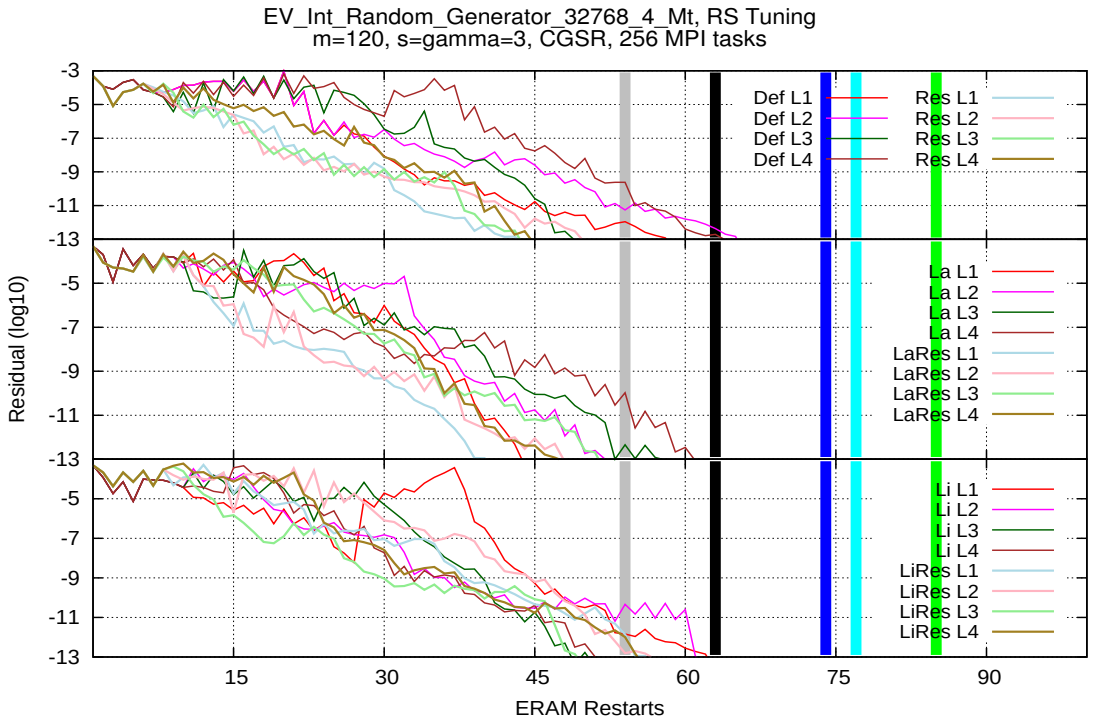

Figure 8.8: Diagonal Band 2^{16} Matrix, Mixed Restarting Strategies Tuning *Choice 1*. The ERAM has $m = 120$, $s = \gamma = 3$, a CGSR orthogonalization process. We present the dominant eigenvalue residuals. We used 256 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. Each stripe color corresponds to the ERAM using respectively $\{\alpha_{LaRes}, \alpha_{La}, \alpha_{Def}, \alpha_{Res}, \alpha_{Li}\}$ restarting strategy during its complete execution. The ERAMs using α_{LiRes} as a single restarting strategy did not converge.
We presented separately the twin restarting strategies (α_{Def},α_{Res} then α_{La}, α_{LaRes} and finally α_{Li},α_{LiRes}). For each restarting strategy, we present 4 different executions of the Algorithm 13 to show its nondeterministic behavior. As an illustration, LaRes L4 refers to one execution of the Algorithm 13 starting with the α_{LaRes} restarting strategy.

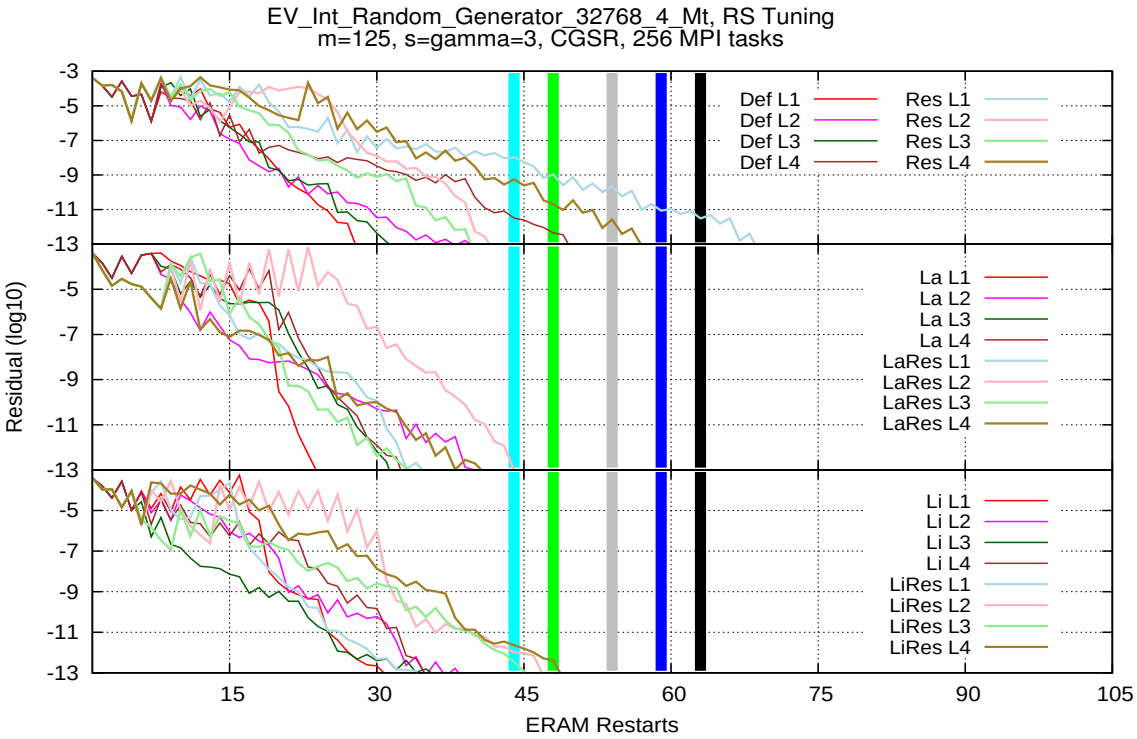

Figure 8.9: Diagonal Band 2^{16} Matrix, Mixed Restarting Strategies Tuning *Choice 1*. The ERAM has $m = 125$, $s = \gamma = 3$, a CGSR orthogonalization process. We present the dominant eigenvalue residuals. We used 256 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. Each stripe color corresponds to the ERAM using respectively $\{\alpha_{Res}, \alpha_{Li}, \alpha_{LaRes}, \alpha_{Def}$ and $\alpha_{La}\}$ restarting strategy during its complete execution. The ERAMs using α_{LiRes} and α_{La} respectively as a single restarting strategy did not converge.
We presented separately the twin restarting strategies (α_{Def},α_{Res} then α_{La}, α_{LaRes} and finally α_{Li},α_{LiRes}). For each restarting strategy, we present 4 different executions of the Algorithm 13 to show its nondeterministic behavior. As an illustration, Def L4 refers to one execution of the Algorithm 13 starting with the α_{Def} restarting strategy.

Globally, the best performances of the ERAM with mixed restarting strategy auto-tuning is obtained with the subspace size value $m = 120$ (cf Figure 8.8): Most of the ERAM auto-tuning increased the numerical convergence performances, while this is not the case by using $m = 115$ (cf Figure 8.7) for example. This is due to the fact that the ERAM performances for this matrix are not evolving according to the Krylov subspace size value as mentioned earlier. Basically, the ERAM with mixed restarting strategy auto-tuned whose subspace size is 115 performances are equivalent to the m=120 one.

On the last Figure 8.9, the ERAM with mixed restarting strategy auto-tuned performance are very satisfying, we still note that some combinations are equivalent to the case of ERAM with $m = 115$.

In this case, increasing m has not annihilated the convergence behavior, we would have to considerably increase the Krylov subspace size value to observe such phenomena. Due to the good performances with a small subspace size, such action has no interest. In terms of numerical performances versus optimizing the parallel execution time, the best scheme is obtained by using the smallest subspace size which is m=115.

Note that the large majority of the ERAMs with the configurations presented for the Diagonal Band 2^{16} matrix are improved by the mixed restarting strategies. In most of the cases, picking α_{LaRes} provides a spectacular acceleration of the ERAM convergence. Due to the sensitivity of the Diagonal Band 2^{16} matrix to the α_{LaRes} restarting strategy, this one largely contributes to the convergence accelerations presented in the figures 8.7,8.8 and 8.9.

8.3.2 Improving the ERAM with Mixed-Restarting Strategy Auto-Tuning

Thanks to the results presented above, we shown the efficiency of the ERAM with mixed restarting strategies auto tuning.

Thanks to the automation of the mixed restarting strategy, we could explore many combinations and emphasize some pertinent restarting strategies combinations. Nevertheless, we also observe that some restarting strategy combinations could not avoid the chaotic convergence of the ERAM.

We did not present the Algorithm 13 by using the Choice 2 (id est a completely random restarting strategy) for the reason that it favors the chaotic convergence behavior.

We recall that in this chapter, we use the current eigenpairs to compute the restarting vector (according to the Algorithm 13) while in the chapter 7, we restarted the ERAM at the restart where the best convergence scheme had been detected.

In what follows, we aim to go further on the mixed restarting strategy and improve their efficiency by using Ritz eigenpairs issued from different restarts.

8.4 Compute the Restarting Vector According to the ERAM Mixed Restarting Strategy

In the previous chapter, we emphasized that there are two possibilities to compute the restarting vector in the context of mixed restarting strategies.

- The most straightforward possibility is simply to use the Equation 8.2 whose restarting coefficients are adapted to the new restarting strategy.

$$v_1{}^{(i+1)} = \sum_{j=1}^{\gamma} \alpha_j^{(i)} \Re(u_j^{(i)}), \tag{8.2}$$

  Where $\gamma \in [1, m]_{\mathbb{N}}$.

- The second possibility is to use the Ritz eigen information of the k^{th} ERAM restart where the Ritz eigenpairs had the smallest residual computed so far, as expressed in the following equation:

$$\forall j \in [1, \gamma]_{\mathbb{N}}, \quad res_j^{(Best)} = \min_{i \in [1, current\_restart]_{\mathbb{N}}} \left(\frac{||Au_j^{(i)} - \theta_j^{(i)} u_j^{(i)}||}{|\theta_j^{(i)}|}\right), \tag{8.3}$$

In this case, the ERAM will use eigenpairs provided by different restarts. Note that the algorithm remains deterministic. Somehow, this restarting strategy has the same properties as the MERAM process. We recall that the MERAM asynchronously shares the Ritz eigenpairs with every ERAM component. The ERAM$_l$ uses at the $i_l + 1^{th}$ restart its own Ritz eigenpairs, the received eigenpairs or both of them. One ERAM will restart its own process by using the *"most accurate Ritz eigenpairs ever computed"* whether they come from an another ERAM or an another restart. As a difference, the MERAM has an nondeterministic behavior. The equation 8.3 does not guarantee that each *"Best eigenpairs"* will be computed at the same restart: we recall that the convergence for each desired eigenpair is not uniform. This will be detailed in what follows.

The truth is, there are plenty of possibilities to compute the restarting vector, whether we discuss about the mixed restarting strategy or a single restarting strategy.

The idea is to improve the Krylov subspace convergence without disrupting the Ritz eigenpairs whose convergence is satisfactory. We introduce a new restarting vector that uses Ritz eigenpairs issued from different restarts. This study is issued from the results of the section 7.2 essentially.

In what follows, we retain, at each restart, the *"best Ritz eigenpairs"*, id est the Ritz eigenpairs that have the smallest residual computed so far. These Ritz Eigenpairs verify the equation 8.3. According to the *Best Ritz Eigenpairs*, the ERAM with Ritz eigenpairs Selection Algorithm then becomes:

Algorithm 15 ERAM with the Best Ritz Eigenpairs Algorithm

Input: $A \in \mathbb{C}^{n \times n}$, $v_1 \in \mathbb{C}^n$, $\varepsilon_{Arnoldi} > 0$, $\varepsilon > 0$, $m \in [1, n]_{\mathbb{N}}, max_{ERAM} \in \mathbb{N}$, $\gamma \in [s, m]_{\mathbb{N}}$, α_X, *orthogonalization*
${U_{n,s}}^{(Best)} \in \mathbb{C}^{n \times s}$
${\Theta_{s,1}}^{(Best)} \in \mathbb{C}^s$
${ResTr_{s,1}}^{(Best)} \in \mathbb{R}^{+s}$

$v_1 = \frac{1}{||v_1||} v_1$

while $(\varepsilon \geq \max_{j \in [1,s]_{\mathbb{N}}} \{res_j^{(Best)}\})$ or $(max_{ERAM} > i)$ **do**

  Execute m-step Arnoldi Method using $\{A, v_1, \varepsilon_{Arnoldi}, m, orthogonalization\}$
  Solve the eigen problem $H_m Y_m = \Theta_m Y_m$
  $U_{n,\gamma} = V_{n,m} Y_{m,\gamma}$
  $resTr_j = ||Au_j - u_j \theta_j||, \forall j \in [1, \gamma]_{\mathbb{N}}$

  if $res_j^{(Best)} > resTr_j^{(i)}$ **then**
    $res_j^{(Best)} = resTr_j^{(i)}$
    $\theta_j^{(Best)} = \theta_j^{(i)}$
    $u_j^{(Best)} = u_j^{(i)}$
  end if
  $v_1^{(i+1)} = \sum_{j=1}^{\gamma} \alpha_j^{(Best)} u_j^{(Best)}$

end while

Output: $U_{n,s}^{(Best)} \in \mathbb{C}^{n \times s}, \Theta_s^{(Best)} \in \mathbb{C}^s, ResTr_s^{(Best)} \in (\mathbb{R}^+)^s$

The Algorithm 15 raises many legitimate questions:

- The ERAM stopping condition:

  The stop condition has changed compared to the original ERAM Algorithm 5.

  The Algorithm 15 output returns Ritz eigenpairs that satisfy the equation 8.3 while the Algorithm 5 returns $\{U_{n,s}^{(i)} \in \mathbb{C}^{n\times s}, \Theta_s^{(i)} \in \mathbb{C}^s, ResTr_s^{(i)} \in (\mathbb{R}^+)^s\}$ where i is the last restart executed by the ERAM (whether it has reached the convergence or the maximum number of restarts allowed by the user). This is similar to the ERAM with Deflation (respectively the MERAM), as the locked Ritz eigenpairs are not necessarily issued from the same restarts (respectively and the same ERAM component).

- Conserve an erroneous Ritz eigenpair:

  The Algorithm 15 needs additional checking regarding the *Best* Ritz eigenpairs. Indeed, during the first restarts, the Krylov subspaces are unstable, leading to the *convergence peak* (such as presented in Figures 6.1 and 6.2 for example). Such perturbations are due to the fact that the very first (in terms of ERAM restarts) Ritz eigenpairs may have a relatively small residual (by relatively, we mean compared to the previous restarts) but still not be close to the real eigenpairs.

  In this particular case, one may remain in the *Best* Ritz eigenpairs an erroneous eigenpair, forcing the successive Krylov subspaces to wrong convergence direction.

- Reproducibility: The eigen informations used to compute the restarting vector depends on the approximated eigenpair residual value and/or evolution. This operation depends on rounding and floating point operations, therefore, from one supercomputer to another one, the $\theta_j^{(Best)}$, $u_j^{(Best)}$ and $resTr_j^{(Best)}$ evolution will be different. Even though the method is still deterministic, this problem must be recalled, as we change the ERAM behavior with respect to hardware-sensitive parameters.

- Potentially Cyclic Krylov Subspaces:

  If we remain the Algorithm 15 as presented without any additional condition, we can observe the case when the Ritz eigenpairs computed after the i^{th} restart will not improve the Best Ritz eignpairs anymore ($\mathbb{K}_{m,v_1}^{(i)}$ provides less accurate eigenpairs than $\mathbb{K}_{m,v_1}^{(i-1)}, \forall i \in [2, current_{restart}]$). Starting from the restarting vector $v_1^{(i+1)}$ computed as presented in the Algorithm 15, this means that we can not compute $\{U_{n,s}^{(i)} \in \mathbb{C}^{n\times s}, \Theta_s^{(i)} \in \mathbb{C}^s\}$ more accurate than $\{U_{n,s}^{(Best)} \in \mathbb{C}^{n\times s}, \Theta_s^{(Best)} \in \mathbb{C}^s\}$. In this case, we will use the same $v_1^{(i+1)}$ vector constantly, leading to *cyclic* Krylov subspaces: $\forall (k,l) \in [i, max_{ERAM}]_{\mathbb{N}}^2, \mathbb{K}_{m,v_1}^{(k)} = \mathbb{K}_{m,v_1}^{(l)}$. We will detail in what follows our investigations to avoid such behavior. This paragraph will be detailed in what follows.

In what follows, we will present some approaches to solve a subset of issues raised by the Algorithm 15.

8.4.1 Erase the Cyclic Krylov Subspaces

We illustrate the Cyclic Krylov Subspace issue with the matrix Ex11 Dense. We will use the following Figure 8.10 to detail the phenomena.

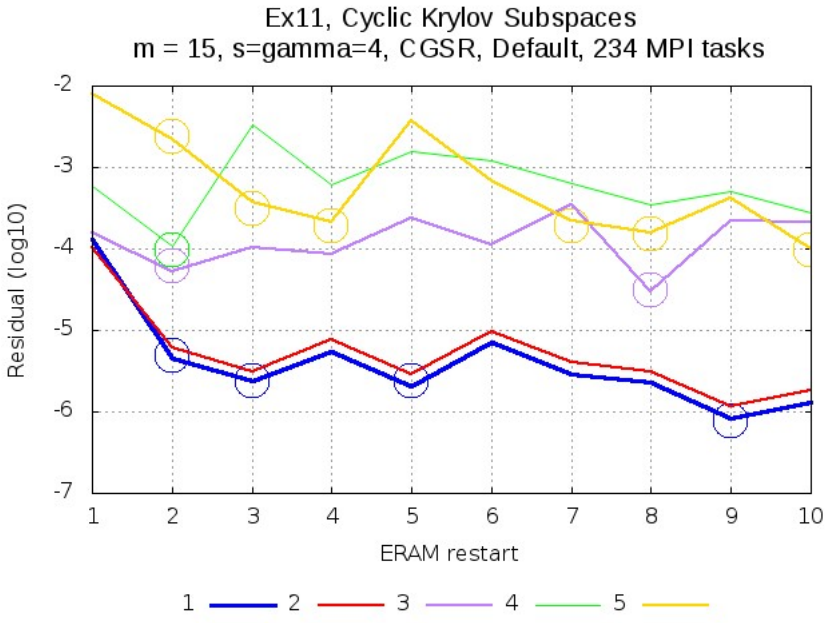

Figure 8.10: $Ex11$ Dense, Cyclic Krylov Subspace Illustration.

This Figure illustrates the cyclic Krylov subspace in the case of the Algorithm 15. The ERAM has $m = 15$, $s = \gamma = 5$, a CGSR orthogonalization process and the restarting strategy α_{Def}. The ERAM has been executed with 234 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. We present on this Figure the residual associated to each dominant Ritz eigenpair. 1 refers to the dominant real modulus eigenvalue, 2 to the second dominant eigenvalue etc until the lowest one which is the fifth (5 on the figure above). Each circle refers to a local minimum respectively to each dominant eigenpair.

Let's consider that we compute $v_1^{(i+1)}$ according to the equations 8.3 and 8.2. The explicit computation formula of $v_1^{(i+1)}$ $i \in [2, 5]$ according to the Algorithm 15 is:

$$\begin{aligned}
{v_1}^{(3)} &= \Re(u_1^{(2)}) + \Re(u_2^{(2)}) + \Re(u_3^{(2)}) + \Re(u_4^{(2)}) + \Re(u_5^{(2)}),\\
{v_1}^{(4)} &= \Re(u_1^{(3)}) + \Re(u_2^{(3)}) + \Re(u_3^{(2)}) + \Re(u_4^{(2)}) + \Re(u_5^{(3)}),\\
{v_1}^{(5)} &= \Re(u_1^{(3)}) + \Re(u_2^{(3)}) + \Re(u_3^{(2)}) + \Re(u_4^{(2)}) + \Re(u_5^{(4)}),\\
{v_1}^{(6)} &= \Re(u_1^{(5)}) + \Re(u_2^{(5)}) + \Re(u_3^{(2)}) + \Re(u_4^{(2)}) + \Re(u_5^{(4)}),
\end{aligned} \tag{8.4}$$

During the first 10 restarts, there is at least one eigenpair whose residual is improved in the Figure 8.10, therefore we do not observe the cyclic Krylov subspaces.

However, if at the 7^{th} restart the lowest modulus Ritz eigenpair (yellow line on the Figure 8.10) residual is higher than the 4^{th} restart, then the ${v_1}^{(7)}$ formula is the same as ${v_1}^{(6)}$ and therefore for all the following restarts.

As a remedy, we impose to the ERAM to change at least one of its parameter if a least one of the current eigenpair has not a better residual than the previously computed one id est:

$$\exists j \in [1, \gamma] \text{ such that } resTr_j^{(Best)} > resTr_j^{(i)}, \tag{8.5}$$

Whatever the parameter you change, there is a necessity to study at each ERAM restart if at least one of the *"Best eigenpairs"* had been changed, therefore we add to the Algorithm 15:

Algorithm 16 Erase the Cyclic Krylov Subspaces

1: **if** $\forall j \in [1, \gamma], resTr_j^{(Best)} \leq resTr_j^{(i)}$ **then**

2: Change at least one parameter of the ERAM among $\{\gamma, \alpha_X, orthogonalization, m\}$

3: **end if**

8.4.2 Mixed Restarting Strategy & Best Ritz Eigenpairs Tuning Results

According to the previous section, we converged to the following Algorithm which is the combination of the Algorithms 15 and 13:

Algorithm 17 the ERAM with Mixed Restarting Strategy and Best Ritz Eigenpairs Tuning

Input: $A \in \mathbb{C}^{n\times n}$, $v_1 \in \mathbb{C}^n$, $\varepsilon_{Arnoldi} > 0$, $\varepsilon > 0$, $m \in [1,n]_{\mathbb{N}}$, $s \in [1,m]_{\mathbb{N}}, max_{ERAM} \in \mathbb{N}$, $\gamma \in [s,m]_{\mathbb{N}}$, $(f_{inf}, f_{sup}) \in]0,1[^2$, $max_{count} \in \mathbb{N}^*$, α_X, *orthogonalization*

${U_{n,s}}^{(Best)} \in \mathbb{C}^{n\times s}$
${\Theta_{s,1}}^{(Best)} \in \mathbb{C}^s$
${ResTr_{s,1}}^{(Best)} \in \mathbb{R}^{+s}$

1: $v_1 = \frac{1}{||v_1||} v_1$

2: **while** $(\varepsilon \geq \max_{j\in[1,s]_{\mathbb{N}}} \{res_j^{(Best)}\})$ or $(max_{ERAM} > i)$ **do**

3: Execute m-step Arnoldi Method using $\{A, v_1, \varepsilon_{Arnoldi}, m, orthogonalization\}$
4: Solve the eigen problem $H_m Y_m = \Theta_m Y_m$
5: $U_{n,\gamma} = V_{n,m} Y_{m,\gamma}$
6: $resTr_j = ||Au_j - u_j\theta_j||, \forall j \in [1,\gamma]_{\mathbb{N}}$

7: $Conv_{status}$ =Multi-Levels Convergence Algorithm$(res_{CV}^{(i)}, res_{CV}^{(i-1)}, f_{inf}, f_{sup}, max_{count})$
8: **if** $res_j^{(Best)} > resTr_j^{(i)}$ **then**
9: $res_j^{(Best)} = resTr_j^{(i)}$
10: $\theta_j^{(Best)} = \theta_j^{(i)}$
11: $u_j^{(Best)} = u_j^{(i)}$
12: **end if**
13: **if** $Conv_{status}$ Diverges or $Conv_{status}$ Stagnates or $\forall j \in [1,\gamma], resTr_j^{(Best)} \leq resTr_j^{(i)}$ **then**

14: switch $\alpha_X^{(i)}$ by $\alpha_{new}^{(i+1)}$ such that $\alpha_X^{(i)} \neq \alpha_{new}^{(i+1)}$
15: **end if**
16: **if** i%5==0 **then**
17: $v_1^{(i+1)} = \sum_{j=1}^{\gamma} \alpha_j^{(Best)} u_j^{(Best)}$
18: **else**
19: $v_1^{(i+1)} = \sum_{j=1}^{\gamma} \alpha_j^{(i)} u_j^{(i)}$
20: **end if**
21: **end while**

Output: $U_{n,s} \in \mathbb{C}^{n\times s}, \Theta_s \in \mathbb{C}^s, ResTr_s \in (\mathbb{R}^+)^s$

We added the condition at step 16 of the Algorithm 17 for three reasons.

The first reason is that an ERAM **must** pursue its convergence by using its current

Ritz eigenpairs to stabilize them and possibly improve its current Ritz eigenpairs: as an illustration, one may note that the ERAM convergence is not smooth and most of the time presents some small perturbations.

Finally, this also participates to avoid the cyclic Krlov subspace behavior.

The second is to reproduce as best as we can the MERAM behavior. In the case of a MERAM, the ERAM$_k$ ($k \in [1, \mu]_{\mathbb{N}}$ where $\mu \in \mathbb{N}^*$ is the number of ERAMs components of the MERAM) will not restart its own process by using Ritz information from the other components. This is possible in practice but that means that the ERAM$_k$ has a *"eater"* behavior: it does not participate to the amelioration of the Ritz eigenpairs. We highlighted that this behavior is not satisfiable from our point.

In what follows, we present the ERAM with Mixed Restarting Strategy Auto-Tuning (Algorithm 17) using as a first restarting strategy respectively $\{\alpha_{Def}, \alpha_{Res}, \alpha_{Li}, \alpha_{LiRes}, \alpha_{La}, \alpha_{LaRes}\}$. For each figure presented in this section, we indicated thanks to colored stripes the number of restarts until convergence for each ERAM using as a single restarting strategy $\{\alpha_{Def}, \alpha_{Res}, \alpha_{Li}, \alpha_{LiRes}, \alpha_{La}, \alpha_{LaRes}\}$ respectively.

We executed the Algorithm 17 with the target matrices list in the Table 8.3.1.

We present on the Figure 8.11 the Algorithm 17 results with the Fission Dense Matrix.

The residual evolution may seem more smooth on the Figure 8.11 , which is normal as we present the residual associated to the best computed Ritz eigenpair and not to the current Ritz eigenpair.

As a comparison with the Algorithm 13, the use of the Best Ritz Eigenpairs greatly improved the ERAM convergence. The ERAM chaotic convergence is almost avoided, only one configuration remains not satisfiable, which is the LiRes 2 (light-pink): its convergence is worse than the original ERAM configuration. We recall that the ERAM with mixed restarting strategy tuning using the current Ritz eigenpairs (id est the Algorithm 13) had worst results especially for the ERAM starting with the α_{Def} restarting strategy.

Using the best Ritz eigenpairs has a direct impact on the convergence: many ERAM configurations presented in the Figure 8.3 converged many restarts after the best ERAM using a single restarting strategy (here α_{Res}). In this configuration, only LiRes L2 performs worst than the ERAM using α_{Res} only.

As a comparison, the gain between the gain between the worst ERAM using respectively the Algorithm 17 and the Algorithm 13 is about 58 restarts.

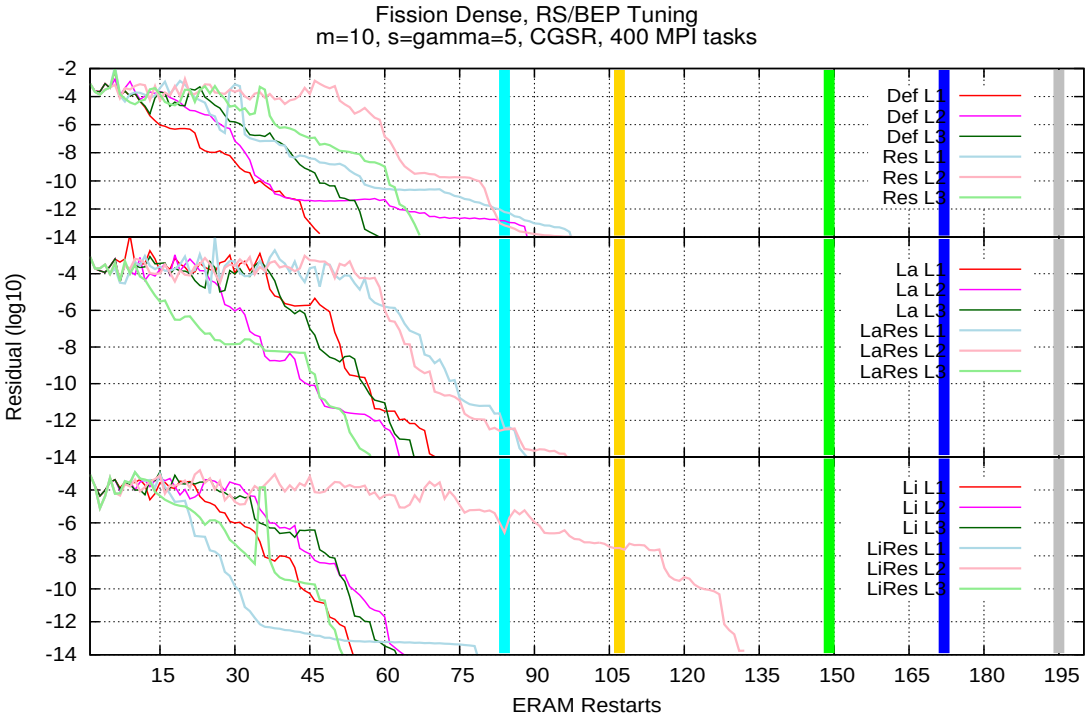

Figure 8.11: *Fission* Dense Matrix, Mixed Restarting Strategies/Best Ritz Eigenpairs Tuning.
The ERAM has $m = 10$, $s = \gamma = 5$, a CGSR orthogonalization process. We present the dominant eigenvalue residuals. We used 400 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. Each stripe color corresponds to the ERAM using respectively $\{\alpha_{Res}, \alpha_{LiRes}, \alpha_{Li}, \alpha_{Def}, \alpha_{LaRes}\}$ restarting strategy during its complete execution (α_{La} did not converge). We presented separately the twin restarting strategies (α_{Def},α_{Res} then α_{La}, α_{LaRes} and finally α_{Li},α_{LiRes}). For each restarting strategy, we present 3 different executions of the Algorithm 17 to show its nondeterministic behavior. As an illustration, Res L2 refers to one execution of the Algorithm 17 starting with the α_{Res} restarting strategy.

We summarized in the following Table 8.4.2 the gains of the ERAMs with best Ritz eigenpairs and auto-tuned restarting strategies (id est the ERAM using the Algorithm 17)

versus the ERAMs using ONLY the auto-tuned restarting strategies (id est the ERAM using the Algorithm 13). For each restarting strategy, we compare the average number of restarts until the convergence. As an illustration, for both Algorithms 17 and 13, we computed for the ERAMs starting with the

	α_{Def}	α_{Res}	α_{Li},	α_{LiRes},	α_{La}	α_{LaRes}
Algorithm 17	**65**	**91**	**60**	**86**	**66**	81
Algorithm 13	**148**	**120**	**123**	**91**	**94**	149

Table 8.2: We summarized in the Table 8.4.2 the average number of restarts until the convergence for each ERAMs using respectively the Algorithms 17 and 13. We computed the average number of restarts for each restarting strategy. We recall that for both Algorithms 17 and 13, the ERAMs start with the restarting strategies presented in this table and then the restarting strategy is changed with respect to the auto-tuning algorithm.

On the Figure 8.12 we present the Algorithm 17 results with the $Fission_{2^{15}}$ Dense Matrix. The best Ritz eigenpairs also improved the ERAM with mixed restarting strategies auto-tuning: the improvement compared to the Figure 8.3 is not as spectacular as the Fission Dense matrix but still remain very interesting. All the original configurations are improved excepted the ERAM using the α_{Li} restarting strategy: its performance is already very good, improving it more remains hard.

The major contribution is that the chaotic ERAM convergence is limited by the positive impact of the best Ritz eigenpairs.

As a comparison with the ERAM using the Algorithm 13, the configurations starting with α_{Def}, α_{Res}, α_{La} and α_{LaRes} are greatly ameliorated. Especially, the ERAM starting with α_{Def} and α_{Res} provides better results than the ERAM using only these respective restarting strategies during its complete execution, which was not the case by using the previous Algorithm 13.

The ERAM starting with α_{Res} have definitely more difficulties to converge than the other restarting strategies: we recall that the ERAM using this restarting strategy during its complete execution could not reach the convergence, therefore the ERAM using the 17 may have difficulties to stabilize the ERAM as it started to converge in a wrong direction.

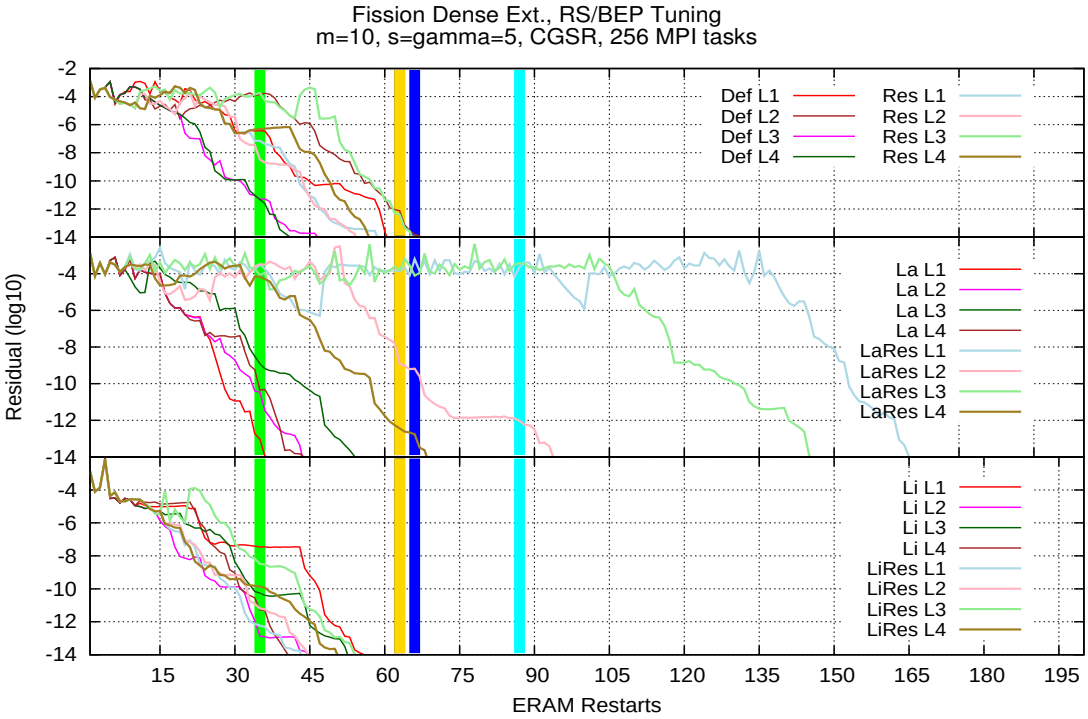

Figure 8.12: $Fission_{2^{15}}$ Dense Matrix, Mixed Restarting Strategies/Best Ritz Eigenpairs Tuning.
The ERAM has $m = 10$, $s = \gamma = 5$, a CGSR orthogonalization process. We present the dominant eigenvalue residuals. We used 256 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. Each stripe color corresponds to the ERAM using respectively $\{\alpha_{Li}$,α_{LiRes},α_{La},α_{Def} and $\alpha_{Res}\}$ restarting strategy during its complete execution (the α_{LaRes} did not converge).
We presented separately the twin restarting strategies (α_{Def},α_{Res} then α_{La}, α_{LaRes} and finally α_{Li},α_{LiRes}). For each restarting strategy, we present 4 different executions of the Algorithm 17 to show its undeterministic behavior. As an illustration, LiRes L4 refers to one execution of the Algorithm 17 starting with the α_{LiRes} restarting strategy.

We present on the Figure 8.13 the Algorithm 17 results applied to the Ex11 Dense Matrix. The integration of the best Ritz eigenpairs globally improved the ERAM convergence compared to the Algorithm 13 which uses only the mixed restarting strategies.

The best gains are obtained thanks to the ERAMs starting with α_{Li} and α_{LiRes}. The Algorithm 17 still maintains this observation, meaning that the ERAM starting with α_{LiRes} provides very good performances, while the ERAM using α_{LiRes} as a single restarting strategy does not converge most of the time. Compared to the initial Algorithm 13, the ERAM starting with α_{Li} and α_{LiRes} convergence are clearly ameliorated (from 5 to 10 restarts).

The ERAM starting with the α_{Def} and α_{Res} respectively are improved (from 7 until 30 restarts).

The configuration starting with α_{La} is already very efficient (such as α_{Res}) therefore having a better performance or improving the ERAM using these restarting strategies during their complete executions remains difficult.

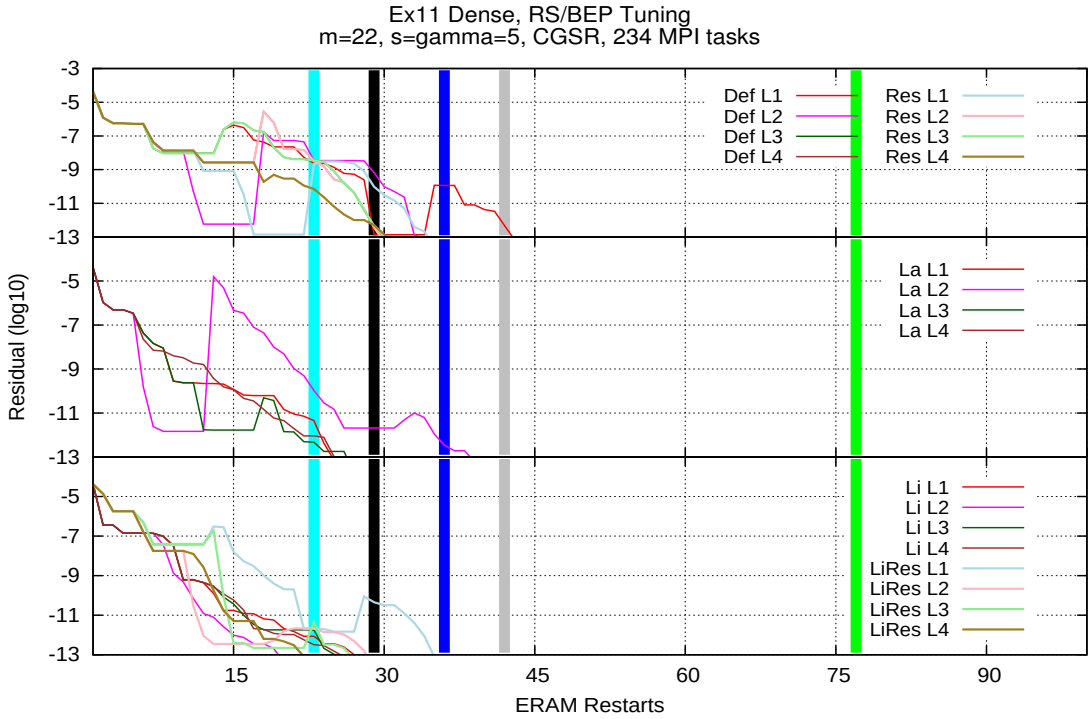

Figure 8.13: $Ex11$ Dense Matrix, Mixed Restarting Strategies/Best Ritz Eigenpairs Tuning.
The ERAM has $m = 22$, $s = \gamma = 5$, a CGSR orthogonalization process. We present the dominant eigenvalue residuals. We used 234 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. Each stripe color corresponds to the ERAM using respectively $\{\alpha_{Res}$,α_{La},α_{Def},α_{LaRes} and $\alpha_{Li}\}$ restarting strategy during its complete execution (α_{LiRes} did not converge) . We presented separately the twin restarting strategies (α_{Def},α_{Res} then α_{La}, α_{LaRes} and finally α_{Li},α_{LiRes}). For each restarting strategy, we present 4 different executions of the Algorithm 17 to show its nondeterministic behavior. As an illustration, Res L4 refers to one execution of the Algorithm 17 starting with the α_{Res} restarting strategy. The ERAM Algorithm 17 starting with α_{LaRes} is not presented as neither divergence nor stagnation was detected.

We present on the Figure 8.14 the results of the Algorithm 17 applied to the Mixtank_new Dense Matrix. The improvements regarding the α_{La} and α_{LaRes} are considerable compared to the basic ERAM with mixed restarting strategy auto-tuning (cf the Figure 8.5).

The integration of the Best Ritz Eigenpairs have a large impact on this matrix, however, this is still not sufficient to avoid completely (such as the Fission Dense, $\text{Fission}_{2^{15}}$ Dense or the Ex11 Dense matrices) the slow convergence behavior in every case but still it is greatly limited compared to the original configurations and the first draft of the ERAM with mixed restarting strategy auto-tuning.

All the ERAM (except the one starting with α_{Def}) using the Algorithm 17 convergence are improved compared to the ERAM using the Algorithm 13.

The ERAM using the Algorithm 17 starting with α_{LaRes} (respectively α_{La}) saved 10 (respectively 165) restarts compared to the ERAM using the Algorithm 13 and starting with α_{LaRes} (respectively α_{La}).

We believe that solving the eigen problem of such matrix by using ERAM would be improved by hybrid methods such as the MERAM solver, more than a single ERAM. Nevertheless, this is absolutely not a restriction as we could consider a MERAM composed of auto-tuned ERAM.

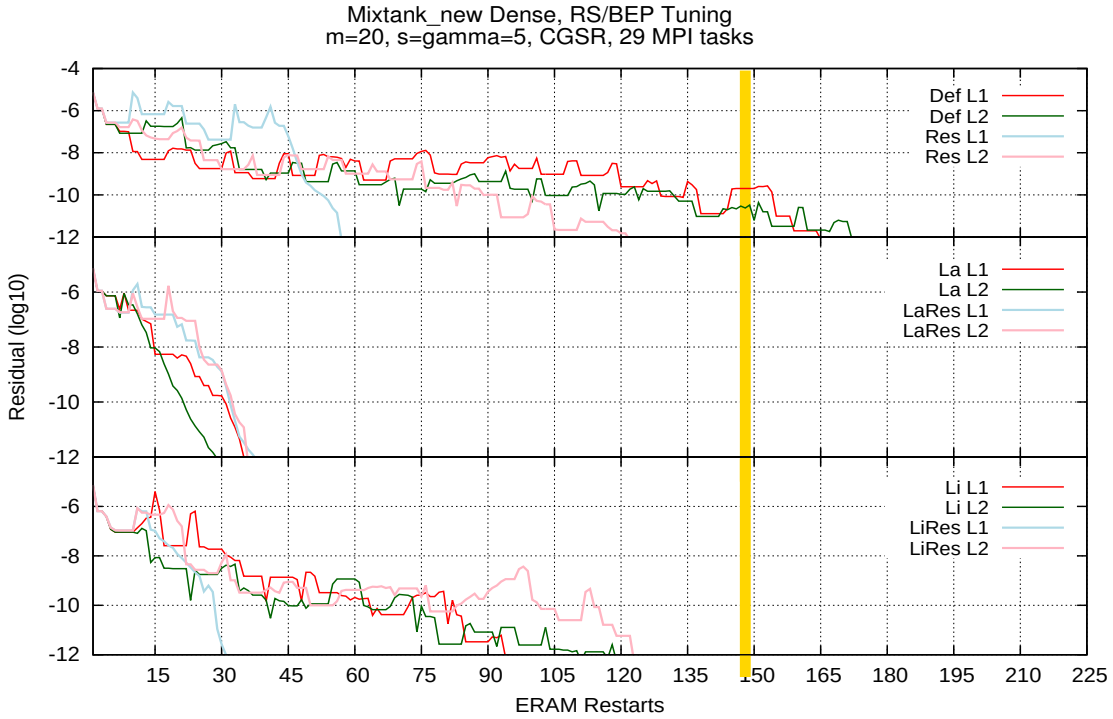

Figure 8.14: Mixtank_new Dense Matrix, Mixed Restarting Strategies/Best Ritz Eigenpairs Tuning.
The ERAM has $m = 20$, $s = \gamma = 5$, a CGSR orthogonalization process. We present the dominant eigenvalue residuals. We used 29 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. Each stripe color corresponds to the ERAM using $\{\alpha_{LiRes}\}$ restarting strategy during its complete execution (this configuration is the only one that could converge). We presented separately the twin restarting strategies (α_{Def},α_{Res} then α_{La}, α_{LaRes} and finally α_{Li},α_{LiRes}). For each restarting strategy, we present 2 different executions of the Algorithm 17 to show its nondeterministic behavior. As an illustration, Res L2 refers to one execution of the Algorithm 17 starting with the α_{Res} restarting strategy.

We present on the Figure 8.15 the Algorithm 17 results applied to the Rim Dense Matrix. The Rim Dense matrix has a convergence profile that remains more close to

the Mixtank_new Dense Matrix. Such as Mixtank_new Dense Matrix, using the Best Ritz eigenpairs has a positive impact on the ERAM convergence. If some configurations (such as La L1, LiRes L4, Def L2 as an illustration) are subject to the chaotic ERAM convergence, it is limited by the best Ritz eigenpairs behavior.

The ERAM starting with α_{Def} and α_{Res} respectively convergence are greatly ameliorated by the use of the best Ritz eigenpairs: In the worst case, the gain between the ERAM using the Algorithm 17 and Algorithm 13 is about 20 restarts for α_{Def} and 50 restarts for the ERAM starting withα_{Res}.

The ERAM starting with α_{La} and α_{LaRes} performances are similar to the ERAM using the Algorithm 13.

We notice that the Algorithm 17 improves the convergence of the ERAM starting with α_{LiRes} (compared to the Algorithm 13), the gain is about 15 restarts for the worst ERAM convergence.

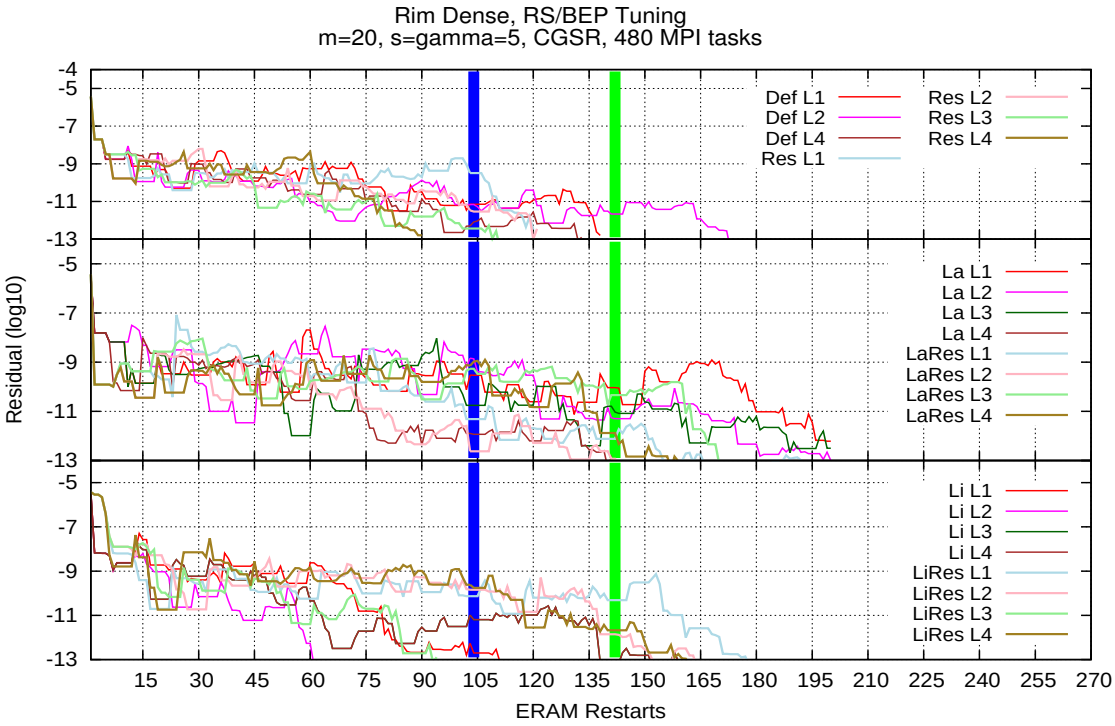

Figure 8.15: *Rim* Dense Matrix, Mixed Restarting Strategies/Best Ritz Eigenpairs Tuning.
The ERAM has $m = 20$, $s = \gamma = 5$, a CGSR orthogonalization process. We present the dominant eigenvalue residuals. We used 480 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. Each stripe color corresponds to the ERAM using respectively $\{\alpha_{Def}$ and $\alpha_{Li}\}$ restarting strategy during its complete execution. The ERAMs using α_{LaRes}, α_{LiRes}, α_{La} and α_{Res} respectively as a single restarting strategy did not converge. We presented separately the twin restarting strategies (α_{Def},α_{Res} then α_{La}, α_{LaRes} and finally α_{Li},α_{LiRes}). For each restarting strategy, we present 4 different executions of the Algorithm 17 to show its undeterministic behavior. As an illustration, Def L2 refers to one execution of the Algorithm 17 starting with the α_{Def} restarting strategy.

Finally, we executed the Algorithm 17 on the Diagonal Band 2^{16} Matrix by using several Krylov subspace sizes. The results are summarized on the figures below.

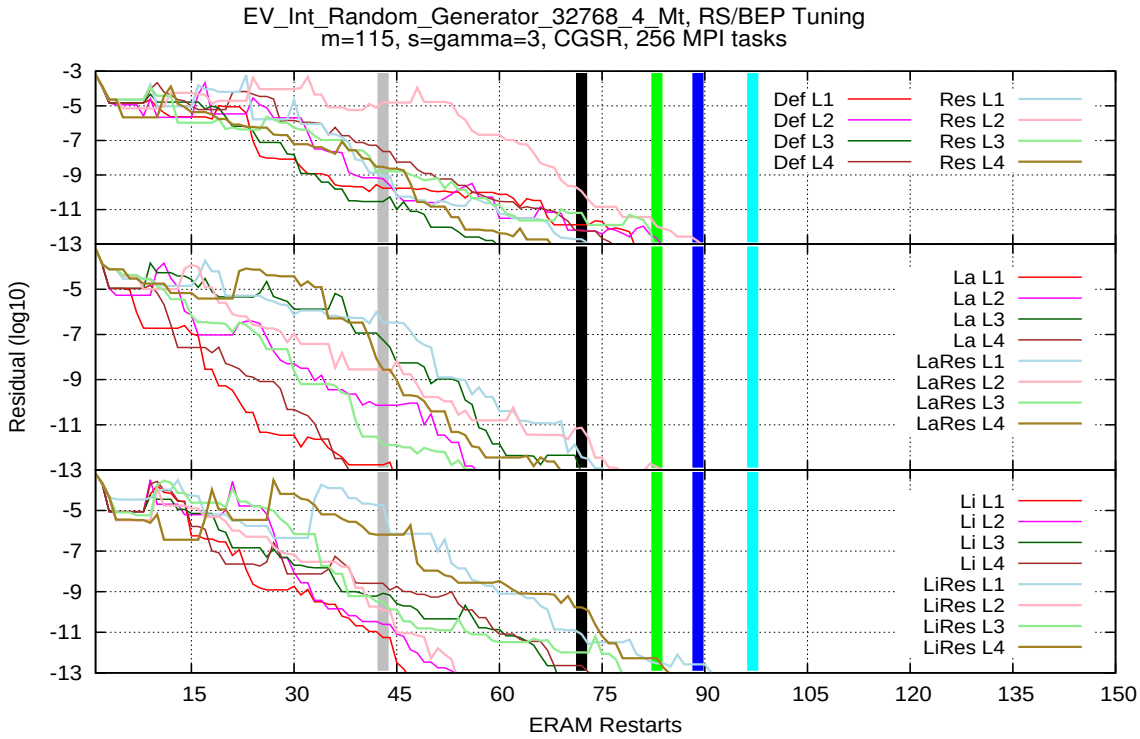

Figure 8.16: Diagonal Band 2^{16} Matrix, Mixed Restarting Strategies/Best Ritz Eigenpairs Tuning.
The ERAM has $m = 115$, $s = \gamma = 3$, a CGSR orthogonalization process. We present the dominant eigenvalue residuals. We used 256 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. Each stripe color corresponds to the ERAM using respectively $\{\alpha_{LaRes}$,α_{La}, α_{Li},α_{Def},$\alpha_{Res}\}$ restarting strategy during its complete execution. The ERAMs using α_{LiRes} as a single restarting strategy did not converge. We presented separately the twin restarting strategies (α_{Def},α_{Res} then α_{La}, α_{LaRes} and finally α_{Li},α_{LiRes}). For each restarting strategy, we present 4 different executions of the Algorithm 17 to show its nondeterministic behavior. As an illustration, La L2 refers to one execution of the Algorithm 17 starting with the α_{La} restarting strategy.

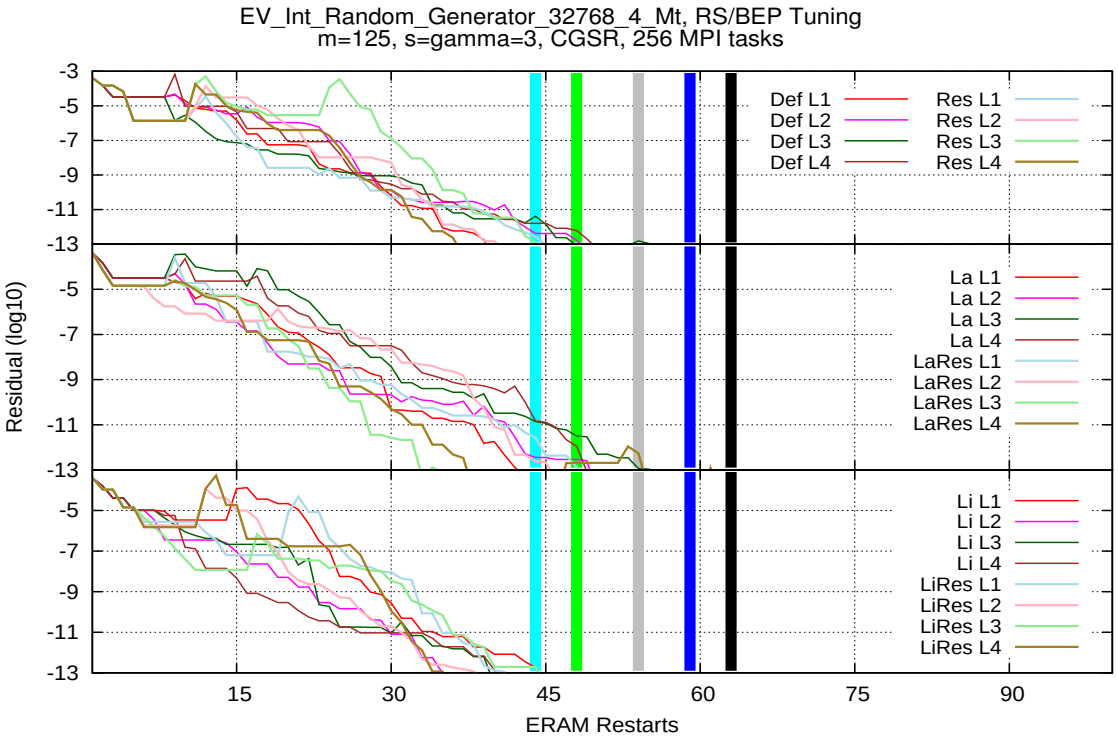

Figure 8.17: Diagonal Band 2^{16} Matrix, Mixed Restarting Strategies/Best Ritz Eigenpairs Tuning.
The ERAM has $m = 125$, $s = \gamma = 3$, a CGSR orthogonalization process. We present the dominant eigenvalue residuals. We used 256 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. Each stripe color corresponds to the ERAM using respectively $\{\alpha_{Res}, \alpha_{Li}, \alpha_{LaRes}, \alpha_{Def}$ and $\alpha_{La}\}$ restarting strategy during its complete execution. The ERAMs using α_{LiRes} and α_{La} respectively as a single restarting strategy did not converge. We presented separately the twin restarting strategies (α_{Def},α_{Res} then α_{La}, α_{LaRes} and finally α_{Li},α_{LiRes}). For each restarting strategy, we present 4 different executions of the Algorithm 17 to show its nondeterministic behavior. As an illustration, LaRes L2 refers to one execution of the Algorithm 17 starting with the α_{LaRes} restarting strategy.

The Best Ritz eigenpairs improved the three configurations presented below especially

the configurations that use the subspace sizes 120 and 125. The configuration with $m = 115$ is not deteriorated compared to the basic ERAm with mixed restarting strategy auto tuning but the convergence acceleration is not as good as $m = 120$ and $m = 125$. For the *"classic"* schemes, we would observe more convergence improvement for the smallest subspace size configurations (as their convergence is supposed to be slower), but the diagonal-band matrices profile definitely does not belong to such classic schemes.

The best Ritz eigenpairs impact is limited by the fact that the configurations presented here have a relatively smooth convergence.

As an illustration, the case $m = 115$ (Figure 8.16) does not accelerate the best ERAM using a single restarting strategy but considerably improved the other scheme by using the α_{LaRes} restarting strategy during its process.

As a comparison with the ERAM using the Algorithm 17, the ERAM starting with α_{LaRes} and α_{La} convergence is ameliorated compared to the ERAM using the Algorithm 13.

The other ERAMs (starting with α_{Def}, α_{Res}, α_{Li} and α_{LiRes}) have a similar convergence whether we use the Algorithm 13 or 17.

We make the same observation for the ERAM using $m = 120$, the results are pretty similar with the ERAM using the Algorithm 13. The reader can consult the Algorithm 17 results with m=120 in the appendix C.

The Algorithm 17 has more impact on the ERAM convergence by using m=125. Basically, we aim to think that the ERAM using the largest subspace size will have the best performances, but this is not the case with the Diagonal band matrices. The performances do not evolve according with the subspace size value.

Therefore, this is not surprising with this matrix that the ERAM with m=125 has a slower convergence than an ERAM with m=115 or 120. This also explains why this configuration is more sensitive to the ERAM with auto-tuned restarting strategies. Using the best Ritz eigenpairs improved the ERAM starting with α_{Res} and α_{LiRes}. For the α_{Res} case, we save from 10 to 25 restarts compared to the ERAM using the Algorithm 13. The ERAM starting with α_{LiRes} convergence saves 5 restarts compared to the ERAM using the Algorithm 13.

The Diagonal Band 2^{16} matrix is less sensitive to the use of best Ritz Eigenpairs than the Fission Dense matrix or the Mixtank_new one for example.

Nevertheless, compared to the first ERAM with restarting strategies auto-tuning Algorithm 13, whether we obtain similar results (such as m=120 or m=115), whether we improve the convergence (such as m=125).

The Algorithm 17 efficiency has been tested on target matrices issued from the matrix generators presented in the Chapter 5. We aim to use matrices whose spectrum are very different so as to study the right efficiency of the Algorithm 17 on several eigenvalues distribution. Nevertheless, the ERAM using the Algorithm 17 remains nondeterministic and we may find a configuration in which the Algorithm 17 may be inefficient: the auto-tuning remain based on observations that occur very often. The auto-tuning are in the border of the computer science and mathematical properties, therefore this highlights the

urge need to model and integrate the parallel computing parameters onto the theoretical methods design. The theoretical study of the restarting strategy impact for the ERAM is our next priority.

The Algorithm 17 efficiency remains whether close or better to the initial restarting strategy auto tuning Algorithm 13 presented in the previous paragraph. Conserve the best Ritz eigenpair has the advantage to force the restart at the best configuration we *"ever had so far"*. The use of a new restarting strategy guarantees that cyclic Krylov subspaces will not arise. Additionally, we mentioned that using the best Ritz eigenpairs to restart the ERAM process is a similar restart as the MERAM, as the Ritz eigenpairs do not provide from the same restarts. Nevertheless, we recall that the ERAM with best Ritz eigenpairs has a deterministic behavior, unlikely the MERAM.

Conclusion

This chapter links the all work presented in this thesis. Building a complete auto-tuning algorithm is not as simple as it may seem. Many questions must be solved before the creation of a heuristic and each player of the auto tuning must be wisely chosen.

During the first chapters of this thesis, we answered to most of the auto-tuning questions, namely when shall we intervene in the ERAM convergence, and what shall we change. We highlighted a method to study and characterize the ERAM convergence, then emphasized the major role of the restarting strategies regarding the ERAM convergence behavior. Thanks to the proof of concept of the ERAM with mixed restarting strategies, we could propose several options to build an ERAM using mixed restarting strategies with auto-tuning methods.

This chapter linked the two previous questions by explaining how shall we change the ERAM restarting strategy.

We first proposed a *"natural and straightforward"* version of the ERAM with mixed restarting strategies using auto tuning methods and emphasized the positive impact on the ERAM convergence compared to the ERAM using a single restarting strategy.

Such tests have been made possible thanks to the matrix generators that provided many matrices with different spectrum schemes (values, distribution ...), allowing us to conclude on a general efficiency of the ERAM with mixed restarting strategy auto tuning.

According to the mixed restarting strategies tests presented in the Chapter 7, we improved the auto-tuned ERAM by using an approach that can be compared to the MERAM one (and in a larger sense, MRAM). We introduced the best Ritz eigenpairs, id est the approximated eigenpairs whose residual is the lowest observed so far.

It results from the tests experimented on our target matrices that such Algorithm improves the ERAM original configurations and performs better than the first version of the ERAM with mixed restarting strategies auto-tuning. This leads to a smoother ERAM convergence and a better gain in terms of number of restarts to reach the convergence.

This Algorithm can be improved by adding many other possibilities in terms of the choice and *"regulation"* of the best Ritz eigenpairs and moreover by changing other ERAM parameters such as the subspace size or the orthogonalization scheme, as we mentioned in the very beginning of this chapter.

Finally, we think that the Algorithm 17 may have a large impact in the case of the hybrid methods such as the MERAM or MRAM in a large sense. Indeed, we will take advantage of more Best Ritz Eigenpairs thanks to the MRAM algorithms (the best eigenpairs may come from the ERAM itself or an other ERAM) and each ERAM will benefit from its own restarting strategy auto-tuning acceleration.

Conclusion & Features

In this thesis we focused on the Krylov eigensolver ERAM as this method is well adapted to compute a subset of the dominant eigenpairs. The good convergence of the ERAM depends on the user ability to fix the right set of parameters.

Therefore, one lack of ERAM is that its convergence may drastically differ depending on its input parameters. We emphasized the preponderant role of the Krylov subspace size m: this parameter is leading both the convergence efficiency and the parallel execution time of the ERAM. Unfortunately, these two parameters evolve in the opposite direction. We exposed the possibility to reduce the number of restarts of the ERAM by using a pertinent restarting strategy. It appears that such parameter does not increase the parallel execution time per restart neiher it requires any additionnal parallel communications nor operations. The ERAM restarting strategies is the basis of this study. The finality of this work is to propose an auto-adaptive ERAM, able to dynamically change its own parameters. In complement of this study, we proposed two matrix generators that present many qualities in the current context of ultra-scale computing. Such matrix generators allow to test the numerical validity and the parallel scalability of current/future eigensolvers.

The first study presented at the Chapter 4 consists of assessing the ERAM convergence. We exposed that the Krylov method GMRES used to solve linear systems dispose of such method to appreciate its numerical convergence at the runtime execution. In the case of the GMRES solver, the convergence metric is used to optimize the subspace size value with the convergence rate: In the case of a strong convergence, the subspace size will be decreased, so as to reduce the parallel execution time per restart while maintaining the GMRES convergence at a statisfiable rate. Conversely, the opposite modification will be done so as the GMRES can reach the convergence. Despite the solvers similarity, the GMRES convergence criteria can not be transposed directly to the ERAM method. The first step is to determine a convergence metric to appreciate the ERAM convergence. Based on this metric, we presented an algorithm that detects the ERAM convergence behavior. A part of this work has been executed during a summer internship at the Computational Research Group of the Lawrence Berkeley National Laboratory under the supervision of M. Leroy Anthony Drummond. We experimented this algorithm on several matrices whose spectrums are widely different, so as their convergence. For each matrix, we experimented the algorithm with different ERAM configurations to favor slow/fast/chaotic convergence behavior. We fixed two priorities regarding the ERAM convergence detection. First, the divergence or stagnation status must be quickly detected. We aim to avoid such behavior by modifying ERAM parameters so as to reach the convergence faster. This implies the second condition, which is properly detect the convergence behavior. The results presented in this chapter shown the efficiency of the algorithm whatever the spectrum distribution. We presented many convergence schemes and shown for each of them the efficient detection of ERAM convergence.

The Chapter 5 is dedicated to the study of two matrix generators. This work has been motivated by the fact that we are missing of test matrices with known spectrum that matches our requirement. Such matrices were needed so as to proceed to numerical tests for the ERAM eigensolver. The matrix collections readily available do not propose such matrices, or these do not have the required mathematical properties. In the context of the French-Japanese collaboration for the *Framework Programming for Post-Petascale Computing (FP3C)* Project, two matrix generators (developped by M. Hêrvé Galicher) have been validated in this thesis. Both matrix generators provide matrices from an imposed spectrum and verify that the generated matrix is non-Hermitian. Secondly, both matrix generators have the advantage to provide large matrices, in terms of dimension and/or number of non-zeros elements. These matrix generators where definitely needed to realize this thesis, as we were craving of large non-Hermitian matrix, non confidential and with a known spectrum. The large majority of the results presented in this thesis use matrices issued from the two matrix generators. In this chapter, we exposed the methods and emphaiszed their benefits/drawbacks in terms of parallelism. We finally conlude on results that show the numerical efficiency of both matrix generators.

The Chapter 6 is dedicated to the ERAM restarting strategies. In this chapter, we will present some restarting strategies and emphasize their contribution to the ERAM convergence. We experimented all the restarting strategies by using many ERAM configurations and argued on the considerable gain they may provide. Each experimentation is executed on the PRACE CURIE Supercomputer (26^{th} most powerfull supercomputer according to the Top500 list of June 2014) by using matrices generated by both methods presented in the Chapter 4. From these experimentations, we could emphasize some spectrum profiles sensibility to specific restarting strategies. As a consequence, we introduced the Twin restarting strategy behavior, meaning that some of the restarting strategies presented in this thesis may behave the same way -or not-: the twin restarting strategy behavior is linked with the spectrum distribution. As a complementary study, we present some results on the restarting strategies efficiency with respect to the orthogonalization process scheme. It is known that the Krylov basis may suffer from a lack of orthogonality. A reorthogonalization of the Krylov basis can be necessary to stabilize numerically the Arnoldi Method. However, this reorthogonalization step is very greedy in terms of parallel computation costs: the problematic still remains on the numerical efficiency versus the execution time per restart.

Starting from all the restarting strategies behavior analyzed in the Chapter 6, we used the ERAM convergence heuristic to manually change the restarting strategy and observe if such proceeding could have an impact on the ERAM convergence. The results presented in the Chapter 7 are promising, as the gain may be spectacular dpending on the matrix spectrum of course. For each target matrix, we proceed to these changes and conlude on an astonishing behavior that we did not expected: Mixing the restarting strategies can turn an inefficient restarting strategy (we mean inefficient if used during the complete ERAM execution) onto a very efficient one. It means that there is the possibility to find an optimum restarting strategy combination so as every ERAM could reach the

convergence faster than its original configuration. We recall that reducing the number of restarts by using mixed restarting strategies is a *"gain"* compared to the subspace size dynamic-adaptations. Indeed, the restarting coefficients only reuse the Ritz eigenpairs information: Therefore this requires neither parallel communications nor operations to use them.

Finally, the Chapter 8 explores several topics to dynamically tune the ERAM restarting strategy with respect to its convergence. Based on the results presented in the Chapter 7, we first study a basic version to smart-tune the ERAM restarting strategies with respect to its convergence. We impose to the ERAM to switch its restarting strategy as soon as a divergence or stagnation status has been detected thanks to the Algorithm presented in the Chapter 4. We first test all the restarting strategies one by one, and finally pick a new one among the restarting strategies that provided the convergence status. We applied this algorithm to the matrices generated in the Chapter 5, thanks to the matrices generators. We conclude on a good efficiency of the ERAM with smart-tuned restarting strategies, but still we can improve this algorithm.

In the second part of this study, we add to this smart-tuning version the possibility to use a Ritz eigenpairs historic, meaning that we remain the Ritz eigenpairs with the lowest residual and potentially mix them with current Ritz eigenpairs. This aims to go on the ERAM convergence while maintaining the accuracy of the Ritz eigenpairs. We finally conclude on the good efficiency of the ERAM using the restarting strategies with smart tuned and best Ritz eigenpairs, as most of the ERAM convergence previously presented had their convergence ameliorated.

We aim to explore some ways of improvement or possible extension in view of the achieved work in this thesis.

We will first continue our matrix generator project. As a first observation, there is a need to developp the diagonal-band and the dense matrix generators frameworks: Many improvement in terms of parallel programmation can be added for both frameworks. The diagonal-band matrix generators deserves to be optimized in terms of parallel data distribution and parallel communications. As a second observation, the diagonal-band matrix generator exact arithmetic must be sacrificed to encompass more spectrums, not only the integer ones. Finally, we would like to go on our study of the theoritical method so as to generate random sparse shapes matrices. As an illustration, we aim to use our matrix generators and then apply some transformation to ensure a random shape of the matrix and conserve its eigenvalues. We recall that the final aim is to share these matrix generators with the HPC community.

We are developping a MERAM version using the smart-tuned ERAMs presented in this thesis. The aim is to outcome to many comethods with different parameters all along, each of them is focusing on accelerating different part of the desired spectrum. We do not aim to outcome to a MERAM where all ERAM have the same optimum parameters: in this case, we lost the eigen information mixing which the heart of the numerical acceleration convergence. The hybrid methods concept allow us to perform many ameliorations both in terms of numerical and parallel communication efficiency.

In this context we want to extend the Hybrid co-method to the GMRES using the smallest eigenvalues to build a preconditioner and accelerate the system convergence. Such hybrid methods would first benefit of accurate and asynchronously sent eigenvalues and then add the resiliency property to the GMRES method. If one component of MERAM fails, then we would turn it into a GMRES and go on the solution convergence. This future work is largely inspired from the MRAM methods presented in this thesis and the works realized in [Aquilanti 2011a].

This achieved work during this thesis would have really been rendered real with the Hybrid Multiple Krylov Restarted Method (whether it is a linear or eigen system) ongoing research.

Bibliography

[Alexander 2011] F. Alexander and al. *A Mutlifaceted Mathematical Approach for Complex Systems*. U.S. Department of Energy, Office of Science, September 13-14, 2011. (Cited on pages 14, 15 and 16.)

[Ang 2012] J. Ang and al. *Report on the Workshop on Extreme-Scale Solvers: Transition to Future Architectures*. U.S. Department of Energy, Office of Advanced Scientific Computing Research (ASCR), March 8-9, 2012. (Cited on pages 14 and 15.)

[Antoniu 2013] G. Antoniu, T. Boku, C. Calvin, P. Codognet, M. Dayde, N. Emad, Y. Ishikawa, S. Matsuoka, K. Nakajima, H. Nakashima, R. Namyst, S. Petiton, T. Sakurai and M. Sato. *Towards exascale with the ANR-JST japanese-french project FP3C (Framework and Programming for Post- Petascale Computing)*. January 2013. (Not cited.)

[Aquilanti 2011a] P. Y. Aquilanti. *Méthodes de Krylov Réparties et Massivement Parallèles pour la Résolution de Problèmes de Géoscience sur Machines Hétérogènes Dépassant le Pétaflop*. Université des Sciences et Technologies de Lille, 2011. (Cited on pages 25 and 180.)

[Aquilanti 2011b] P. Y. Aquilanti, S. Petiton and H. Calandra. *Parallel GMRES Incomplete Orthogonalization AutoTuning*. Procedia Computer Science, vol. 4, pages 2246–2256, January 2011. (Cited on pages 25 and 131.)

[Arnoldi 1951] W. E. Arnoldi. *The Principle of Minimized Iterations in the Solution of the Matrix Eigenvalue Problem*. Quarterly of Applied Mathematics, vol. 9, pages 17–29, 1951. (Cited on page 21.)

[Ashby 2010] S. Ashby and al. *The Opportunities and Challenges of Exascale Computing*. Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee, pages 53–65, 2010. (Cited on pages 7 and 8.)

[Baker 2005] A. Baker, E. R. Jessup and T. Manteufel. *A Technique for Accelerating the Convergence of Restarted GMRES*. SIAM Journal on Matrix Analysis and Applications, 2005. (Cited on pages 47, 48 and 131.)

[Baker 2009] A. Baker, E. R. Jessup and T.V. Kolev. *A Simple Strategy for Varying the Restart Parameter in GMRES(m)*. Journal of Computational and Applied Mathematics, pages 751–761, 2009. (Cited on page 131.)

[Bartlett 1951] M. S. Bartlett. *An Inverse Matrix Adjustment Arising in Discriminant Analysis*. The Annals of Mathematical Statistics, pages 107–111, March 1951. (Cited on page 67.)

[Bathe 1972] K. J. Bathe and E. L. Wilson. *Large Eigenvalue Problems in Dynamic Analysis*. Journal of the Engineering Mechanics Division, Proceedings of the American Society of Civil Engineers, vol. 98, pages 1471–1485, 1972. (Not cited.)

[Bathe 1976] K. J. Bathe and E. L. Wilson. *Numerical Methods in Finite Element Analysis*. Prentice-Hall, 1976. (Not cited.)

[Beckman 2012] P. Beckman and al. *Exascale Operating Systems and Runtime Software Report*. U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR), December 28, 2012. (Cited on pages 8, 12, 13 and 14.)

[Bianchini 2005] M. Bianchini, M. Gori and F. Scarselli. *Inside PageRank*. ACM Transactions on Internet Technology (TOIT), vol. 5, pages 92–128, 2005. (Cited on page 16.)

[Bierens 2013] H. J. Bierens. *The Inverse of a Partitioned Matrix*. June, 2013. (Cited on page 67.)

[Boillod-Cerneux 2012] F. Boillod-Cerneux, S. Petiton, C. Calvin and J. Dubois. *Toward Smart-Tuned Hybrid Asynchronous Krylov Eigenvalue Computing with Multi-Restarted Strategies*. The International Workshop on PMAA, June, 2012. (Not cited.)

[Booth 2011] S. Booth and al. *Exascale and Beyond: Configuring, Reasoning, Scaling (Report of the 2011 Workshop on Architectures II: Exascale and Beyond)*. U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR), August 8-10, 2011. (Cited on pages 6, 9, 11 and 15.)

[Braconnier 2000] T. Braconnier, P. Langlois and J.C. Rioual. *The Influence of Orthogonality on the Arnoldi Method*. Linear Algebra and its Applications, vol. 309, pages 307–323, April 2000. (Cited on page 24.)

[Brown 2010] D. L. Brown, P. Messina and al. *Scientific Grand Challenges: Crosscutting Technologies for Computing at the Exascale*. U.S. Department of Energy, Office of Advanced Scientific Computing Research (ASCR), Office of Science, Office of Advanced Simulation and Computing, National Nuclear Security Administration, February 2-4, 2010. (Cited on pages 7, 8, 9, 11 and 15.)

[Burrage 1998] K. Burrage and al. *A Deflation Technique for Linear Systems of Equations*. SIAM Journal on Scientific Computing, vol. 19, pages 1245–1260, 1998. (Not cited.)

[Carden 2009] R. Carden. Ritz values and arnoldi convergence for nonsymmetric matrices. `http://www.caam.rice.edu/~rlc2/RCarden_thesis.pdf`, April 2009. (Not cited.)

[Carden 2011] R. Carden. Ritz values and arnoldi convergence for non-hermitian matrices. http://www.caam.rice.edu/~rlc2/RCarden_2011.pdf, August 2011. (Cited on page 33.)

[Châtelin 1988] F. Châtelin. *Valeurs Propres de Matrices*. 1988. (Cited on pages 24 and 27.)

[Chen 2005] G. Chen and Z. Jia. *A Refined Harmonic Rayleigh-Ritz Procedure and an Explicitly Restarted Refined Harmonic Arnoldi Algorithm*. volume 41, pages 615–627, March-April 2005. (Cited on page 30.)

[Daly 2011] J. Daly and al. *Tools for Exascale Computing: Challenges and Strategies (Report of the 2011 ASCR Exascale Tools Workshop)*. U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR), October 13-14, 2011. (Cited on pages 5, 7, 9, 10 and 14.)

[Davis] T. Davis. The university of florida sparse matrix collection. http://www.cise.ufl.edu/research/sparse/matrices/. (Cited on pages 51, 64, 69, 92, 133, 195, 196 and 198.)

[Decker 2010] J. Decker and al. *Exascale Workshop Panel Meeting Report*. U.S. Department of Energy, Office of Advanced Scientific Computing Research (ASCR), January 19-20, 2010. (Cited on page 8.)

[Diciunas 1998] V. Diciunas. *Simply Invertible Matrices and Fast Prediction*. Informatica, pages 315–324, 1998. (Cited on page 67.)

[Dongarra 2014] J. Dongarra and al. *Applied Mathematics Research for Exascale Computing*. U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR), March, 2014. (Cited on pages 7, 9, 14 and 15.)

[Dookhitram 2009] K. Dookhitram, R. Boojhawon and M. Bhuruth. *A New Method for Accelerating Arnoldi Algorithms for Large Scale Eigenproblems*. Mathematics and Computers in Simulation, vol. 80, pages 387–401, October 2009. (Not cited.)

[Dubois 2011a] J. Dubois. *Contribution à l'Algorithmique et à la Programmation Efficace des Nouvelles Architectures Parallèles Comportant des Accélérateurs de Calcul dans le Domaine de la Neutronique et de la Radioprotection*. Université des Sciences et Technologies de Lille, Octobre 2011. (Cited on pages 21, 24, 35, 37, 40 and 113.)

[Dubois 2011b] J. Dubois, C. Calvin and S. Petiton. *Auto-Tuned Linear Algebra Computations for Krylov Methods on Multicore and GPUs*. SIAM Scientific Computing, Copper Mountain Special Issue, 2011. (Not cited.)

[Dubois 2011c] J. Dubois, C. Calvin and S. Petiton. *Improving Scalability with Asynchronous Hybrid Methods for non-Hermitian Eigenproblems.* Proceedings of the International Conference on Computational Science, ICCS, vol. 4, pages 222–230, 2011. (Not cited.)

[Dubois 2011d] J. Dubois, C. Calvin and S. Petiton. *Performance and Numerical Accuracy Evaluation of Heterogeneous Multicore Systems for Krylov Orthogonal Basis Computation.* High Performance Computing for Computational Science – VECPAR 2010, vol. 6449, pages 45–57, 2011. (Cited on pages 24 and 91.)

[Dufek 2009a] J. Dufek and W. Gudowski. *Fission Matrix Based Monte Carlo Criticality Calculations.* Annal of Nuclear Energy, Technical Note, vol. 36, pages 1270–1275, 2009. (Not cited.)

[Dufek 2009b] J. Dufek and W. Gudowski. *Scalability and Convergence Problems of the Monte Carlo Fission Matrix Acceleration Methods.* Annal of Nuclear Energy, Technical Note, vol. 36, pages 1648–1651, 2009. (Not cited.)

[Edjlali 1994] G. Edjlali, S. Petiton and N. Emad. *Hybrid Methods on Network of Heterogeneous Computers.* 14th IMACS World Congress, 1994. (Cited on pages 35 and 40.)

[Edjlali 1995] G. Edjlali. *Contribution à la Parallèlisation de Méthodes Itératives Hybrides pour Matrices Creuses sur Architectures Hétérogènes.* Octobre 1995. (Cited on page 35.)

[Edjlali 1996] G. Edjlali, S. Petiton and N. Emad. *Interleaved Parallel Hybrid Arnoldi Method for a Parallel Machine and a Network of Workstations.* Conference on Information, Systems, Analysis and Synthesis (ISAS'96), 1996. (Cited on page 35.)

[Eiermann 2011] M. Eiermann, O. G. Ernst and S. Göttel. *Deflated Restarting for Matrix Functions.* SIAM Journal on Matrix Analysis Applications, vol. 32, pages 621–641, 2011. (Not cited.)

[Emad 2001] N. Emad, S. Petiton and A. Sedrakian. *A Comparison Between Multiple Explicitly Restarted Arnoldi Method and Explicitly Restarted Block Arnoldi Method.* 10^{th} SIAM Conference on Parallel Processing for Scientific Computing, 2001. (Cited on page 35.)

[Emad 2005] N. Emad, S. Petiton and G. Edjlali. *Multiple Explicitly Restarted Arnoldi Method for Solving Large Eigenproblems.* SIAM Journal on Scientific Computing SJSC, vol. 27, pages 253–277, 2005. (Cited on page 35.)

[Embree 2006] M. Embree. *Misconvergence of Arnoldi Eigenvalue Iterations.* June 2006. (Not cited.)

[Galicher 2014] H. Galicher, S. Petiton, C. Calvin and F. Boillod-Cerneux. *Matrix Generators for Ultra-Scale Computing*. Private Communication, to be pusblished, August 2014. (Cited on page 66.)

[Giraud 2002] L. Giraud and J. Langou. *Robust Selective Gram-Schmidt Reorthogonalization*. CERFACS Technical Report, No. TR/PA/02/52, 2002. (Not cited.)

[Giraud 2005] L. Giraud, J. Langou and M. Rozloznik. *The Loss of Orthogonality in the Gram-Schmidt Orthogonalization Process*. Computers Mathematics with Applications, vol. 50, pages 1069–1075, 2005. (Cited on page 24.)

[Gram 1883] J. P. Gram and E. Schmidt. *Über die Entwicklung Reeller Functionen in Reiben Mittelst des Methode der Kleinsten Quadrate*. Journal für die Reine und Angewante Mathematik, vol. 94, pages 41–73, 1883. (Cited on pages 15 and 21.)

[Gre] Green500. http://www.green500.org/. (Cited on page 9.)

[Gustafsson 2012] M. Gustafsson, J. Demmel and S. Holmgren. *Numerical evaluation of the Communication-Avoiding Lanczos algorithm*. Technical Report, Department of Information Technology,Uppsala University, 2012. (Not cited.)

[H. Kuroda 2000] T. Katagiri H. Kuroda and Y. Kanada. *Performance of Automatically Tuned Parallel GMRES(m) Method on Distributed Memory Machines*. 2000. (Cited on page 47.)

[Habu 2000] M. Habu and T. Nodera. *GMRES(m) Algorithm with Changing the Restart Cycle Adaptively*. Proceedings of Algorithmy 2000 Conference on Scientific Computing, pages 254–263, 2000. (Not cited.)

[Hernandez 2006] V. Hernandez, A. Tomas J. E. Roman and V. Vidal. *Arnoldi Methods in SLEPc*. SLEPc Technical Report STR-4, October 2006. (Cited on pages 34, 72 and 78.)

[Hernandez 2007a] V. Hernandez, A. Tomas J. E. Roman and V. Vidal. *Krylov-Schur Methods in SLEPc*. SLEPc Technical Report STR-7, June 2007. (Cited on pages 34, 72 and 78.)

[Hernandez 2007b] V. Hernandez, J.E. Roman and A. Tomas. *Parallel Arnoldi Eigensolvers with Enhanced Scalability via Global Communications Rearrangement*. Parallel Computing, vol. 33, pages 521–540, 2007. (Not cited.)

[Heroux 2004] M. Heroux and all. *An Overview of the Trilinos Project*. 2004. (Not cited.)

[Jia 1995] Z. Jia and L. Elsner. *Improving Eigenvectors in Arnoldi's Method*, 1995. (Not cited.)

[Jia 2004a] Z. Jia. *Some Theoretical Comparisons of Refined Ritz Vectors and Ritz Vectors*. Science in China Series A: Mathematics, vol. 47, pages 222–223, 2004. (Not cited.)

[Jia 2004b] Z. Jia. *The Convergence of Harmonic Ritz Values, Harmonic Ritz Vectors, and Refined Harmonic Ritz Vectors*. Mathematics of Computations, vol. 74, pages 1441–1456, June 2004. (Cited on page 30.)

[Joubert 1994] W. Joubert. *On the Convergence Behavior of the Restarted GMRES Algorithm for Solving Nonsymmetric Linear Systems*. Numerical Linear Algebra with Applications, vol. 1, pages 427–447, 1994. (Cited on page 47.)

[Kamvar 2004] S. Kamvar, T. Haveliwala and D. Golub. *Adaptive Methods for the Computation of PageRank*. Linear Algebra Appl., vol. 386, pages 51–65, 2004. (Cited on page 16.)

[Katagiri 2012] T. Katagiri, P.-Y. Aquilanti and S. Petiton. *A Smart-Tuning Strategy for Restart Frequency of GMRES(m) with Hierarchical Cache Sizes*. VECPAR, Lecture Note in Computer Science, vol. 7851, pages 314–328, 2012. (Cited on pages 47 and 131.)

[Langeville 2004] A. N. Langeville and C. D. Meyer. *Deeper Inside PageRank*. Internet Mathematics, vol. 1, 2004. (Not cited.)

[Lascaux 2004] P. Lascaux. *Analyse Numérique Matricielle Appliquée à l'Art de l'Ingénieur, Tome 2 - Méthodes Itératives*. Dunod, 2004. (Not cited.)

[Lee 1979] I. W. Lee and A. R. Robinson. *Solution Techniques for Large Eigenvalue Problems in Structural Dynamics*. Structural Research Series N. 462, Technical Report of Research, The Office of Naval Research, Department of the Navy, 1979. (Not cited.)

[Lehoucq 1999] R. Lehoucq. *On The Convergence Of An Implicitly Restarted Arnoldi Method*. SIAM J. Matrix Anal. Appl, vol. 23, pages 551–562, 1999. (Not cited.)

[Lewis 2004] E. E. Lewis and W. F. Miller Jr. *Computational Methods of Neutron Transport*. John Wiley & Sons, New York, 2004. (Cited on page 16.)

[Lin] Linpack. `www.netlib.org/linpack`. (Not cited.)

[Mandry 2001] C. Mandry and E. Traviesas. *Convergence de la Méthode d'Arnoldi en Précision Finie en Fonction du Vecteur Initial*. CERFACS Working Notes WN/PA/01/36, 2001. (Not cited.)

[Mat] Matrix market. `http://math.nist.gov/MatrixMarket/`. (Cited on page 64.)

[Mattson 1997] T. G. Mattson and G. Henry. *The ASCI Option Red Supercomputer.* Thirteenth Annual Conference, Intel Supercomputer Users Group, 1997. (Cited on page 6.)

[Mattson 2009] T. Mattson, I. Buck, M. Houston and B. Gaster. *A Standard Platform for Programming Heterogenous Parallel Computers.* International Conference on Supercomputing, 2009. (Not cited.)

[Meuer] H. Meuer, E. Strohmaier, J. Dongarra and H. Simon. Top500. http://www.top500.org/. (Cited on pages 8 and 110.)

[Mika 1968] J. Mika. *Existence and Uniqueness of the Solution to the Critical Problem in Neutron Transport Theory.* Studia Mathematica, vol. 37, pages 213–225, 1968. (Not cited.)

[Mohiyuddin 2009] M. Mohiyuddin, M. Hoemmen J. Demmel and K. Yelick. *Minimizing communication in sparse matrix solvers.* 2009. (Not cited.)

[Moo 2005] *Excerpts from a Conversation with Gordon Moore: Moore's Law.* Video Transcript, Intel Corporation, 2005. (Cited on page 6.)

[Moore 1965] G. E. Moore. *Cramming more Components onto Integrated Circuits.* Electronics Magazine, vol. 38, April 19, 1965. (Cited on page 6.)

[MPI] Mpi 3.0 fault tolerance working group. http://meetings.mpi-forum.org/mpi3.0_ft.php. (Cited on page 13.)

[New 1929] Super computing machines shown, March 1, 1929. (Cited on page 6.)

[Petiton 1988] S. Petiton. *Du Développement de Logiciels Numériques en Environnements Parallèles.* Université Pierre et Marie Curie, 1988. (Not cited.)

[Roa 2008] Roadrunner supercomputer puts research at a new scale, June 12, 2008. (Cited on page 6.)

[Rupp 2013] K. Rupp. Cpu, gpu and mic hardware characteristics over time. http://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/, 2013. (Cited on page 10.)

[Saad 1980] Y. Saad. *Variations on Arnoldi's Method for Computing Eigenelements of Large Unsymmetric Matrices.* Linear Algebra and its Applications, vol. 34, pages 269–295, 1980. (Cited on pages 21, 25, 29, 31 and 33.)

[Saad 1986] Y. Saad and M. Schultz. *GMRES: a Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems.* SIAM Journal on Scientific and Statistical Computing, vol. 7, pages 856–869, 1986. (Cited on page 46.)

[Saad 2003] Y. Saad. *Iterative Methods for Sparse linear Systems, Second Edition*. SIAM, 2003. (Cited on page 46.)

[Saad 2011] Y. Saad. *Numerical Solution of Large Nonsymmetric Eigenvalue Problems,Second Edition*. 2011. (Cited on pages 21, 23, 25, 26, 27, 29, 31, 33, 89 and 131.)

[Schmidt 1908] E. Schmidt. *Über die Auflösung linearer Gleichungen mit Unendlich vielen Unbekannten*. Rendiconti del Circolo Matematico di Palermo, vol. 25, pages 53–77, December 1908. (Cited on pages 15 and 21.)

[Scott 1995] J. A. Scott. *An Arnoldi Code for Computing Selected Eigenvalues of Sparse, Real, Unsymmetric Matrices*. ACM Transaction on Mathematical Software, vol. 21, pages 432–475, 1995. (Not cited.)

[Sedrakian 2005] A. Sedrakian. *Toward a Decision-Making Aid for Re-Usable Parallel Hybrid Iterative Methods*. Versailles University, Department of Computer Science, May 2005. (Cited on page 89.)

[SLE] Slepc - scalable library for eigenvalue problem computations. `www.grycap.upv.es/slepc/`. (Cited on pages 34, 46 and 78.)

[Snir 2011] M. Snir, W. Gropp and P. Kogge. *Exascale Research: Preparing for the Post Moore Era*. June 19, 2011. (Cited on page 7.)

[Sorensen 1990] D. C. Sorensen. *Implicit Application of Polynomial Filters in a k-Step Arnoldi Method*. RIACS Technical Report 90.43, October 1990. (Cited on page 33.)

[Sorensen 1996a] Sorensen, R. B. Lehoucq and D. C. *Deflation Techniques for an Implicitly Restarted Arnoldi Iteration*. SIAM Journal on Matrix Analysis and Applications, vol. 17, pages 789–821, October 1996. (Not cited.)

[Sorensen 1996b] D. C. Sorensen. *Implicitly Restarted Arnoldi/Lanczos Methods for Large Scale Eigenvalue Calculations*. NASA Contractor Report 198342 ICASE Report No. 96-40, 1996. (Not cited.)

[Springel 2011] V. Springel. *Cosmological Simulation*. PASCOS Conference, July, 2011. (Cited on page 5.)

[van der Vorst 2009] H. A. van der Vorst and C. Vuik. *Superlinear Convergence Behaviour of GMRES*. Journal of Computational and Applied Mathematics, vol. 48, pages 327–341, 2009. (Not cited.)

[Vorst 1997] H. A. Van Der Vorst and G. H. Golub. *150 Years Old and Still Alive: Eigenproblems*, 1997. (Cited on page 1.)

[Warsa 2004] J. S. Warsa and all. *Krylov Subspace Iterations for Deterministic k-Eigenvalue Calculations*. Nuclear Science and Engineering, vol. 147, pages 26–42, 2004. (Not cited.)

[Wilkinson 1988] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, 1988. (Not cited.)

[yml] Large-scale workflow computing. http://yml.prism.uvsq.fr/index.php?lang=fr. (Cited on page 13.)

Communications

[Boillod-Cerneux 2011] F. Boillod-Cerneux. *Co-Array Fortran programming applied to seismic imaging.* Maison de la Simulation, 5^{th} Seminar on High Performance Numerical, Octobre 2011. (Not cited.)

[Boillod-Cerneux 2012] F. Boillod-Cerneux, C. Calvin, J. Dubois and S. Petiton. *Auto-tuned Hybrid Asynchronous Krylov Iterative Eigensolver on Petascale Computer.* PMAA 2012, London, UK, June 2012. (Not cited.)

[Boillod-Cerneux 2014a] F. Boillod-Cerneux, S. Petiton, C. Calvin and L. A. Drummond. *Parameter Selection Prediction to Tune the Performance of Krylov Subspace Methods.* SIAM PP14, SCIAM Conference on Parallel Processings for Scientific Computing, February 2014. (Not cited.)

[Boillod-Cerneux 2014b] F. Boillod-Cerneux, S. Petiton, C. Calvin and L. A. Drummond. *Toward Restarting Strategies Tuning for a Krylov Eigenvalue Solver.* IWAPT 2014, The Ninth International Workshop on Automatic Performance Tuning, July 2014. (Not cited.)

[Calvin 2012] C. Calvin, S. Petiton, J. Dubois and F. Boillod-Cerneux. *An Eigenvalue Solver Using a Linear Algebra Framework for Multi-core and Accelerated Petascale Supercomputers.* 15^{th} SIAM Conference on Parallel Processing for Scientific Computing, Savannah, USA, 2012. (Not cited.)

[Calvin 2013a] C. Calvin, L. A. Drummond, F. Boillod-Cerneux and G. Ndongo Eboum J. Dubois. *Auto-tuning and Smart-tuning Approaches for Efficient Krylov Solvers on Petascale Architectures.* SIAM Conference on Computational Science and Engineering, Boston, USA, 2013. (Not cited.)

[Calvin 2013b] C. Calvin, S. Petiton, F. Ye and F. Boillod-Cerneux. *Efficient and Portable Krylov Eigensolver on Many-Cores Architectures.* SNA+MC 2013, Octobre 2013. (Not cited.)

[Calvin 2014] C. Calvin, F. Boillod-Cerneux, F. Ye, H. Galicher, S. Petiton and L. A. Drummond. *Programming Paradigms for Emerging Architectures Applied to Asynchronous Krylov Eigensolver.* SIAM PP14, SCIAM Conference on Parallel Processings for Scientific Computing, February 2014. (Not cited.)

[Petiton 2012] S. Petiton, C. Calvin, J. Dubois, L. Decobert, R. Abouchane, P.-Y. Aquilanti and F. Boillod-Cerneux. *Krylov Subspace and Incomplete Orthogonalization Auto-tuning Algorithms for GMRES on GPU Accelerated Platforms.* 15^{th} SIAM Conference on Parallel Processing for Scientific Computing, Savannah, USA, 2012. (Not cited.)

Appendix A

The Dense Matrix Generator Results

A.0.2.1 The Ex11 Matrix

The Ex11 matrix is originally sparse with 1,096,948 non-zeros elements. Using its spectrum as input parameter of the Algorithm 11 provide a *Dense* $Ex11$ matrix with $275,991,771$ non-zeros elements. The Figure A.1 shows the $Ex11$ eigenvalues without any modification. This matrix is issued from [Davis].

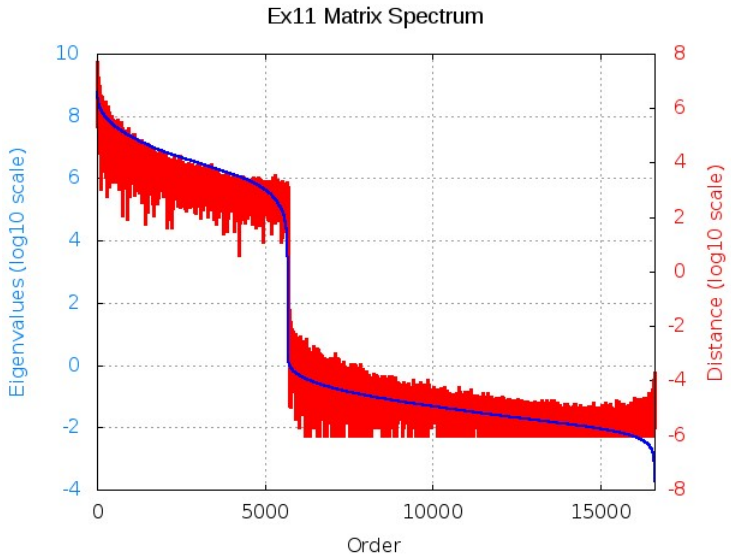

Figure A.1: The Ex11 Matrix Spectrum, Size is $16,614$

The presented spectrum contains the Ex11 matrix real modulus eigenvalues (blue line) without any modification. The red points present the distance between each successive real modulus eigenvalue. The Eigenvalues and Distance metrics use both log_{10} scale to enhance the eigenvalues distribution visibility.

The Figure A.2 presents the residuals (red line) obtained with the SLEPc eigen solvers respectively and the relative errors (blue line). The parameters remains the same as the *Dense Fission* matrix (cf Figure A.1). The precision regarding the residual is very satisfying, however, the relative error for the last 20 eigenvalues is quite high. Indeed, the Ex11 eigenvalues have a 10^8 order and 6 decimals. Getting the exact eigenvalues remains complicated, this explains the relative error difference: the last eigenvalues are *relatively* close to the real eigenvalues considering their order. For this matrix, the spectrum is

conserved through the Algorithm 11 modulo the fact that rounding and floating point operations influence may be more important than the *Dense Fission* matrix due to the Ex11 spectrum characteristics.

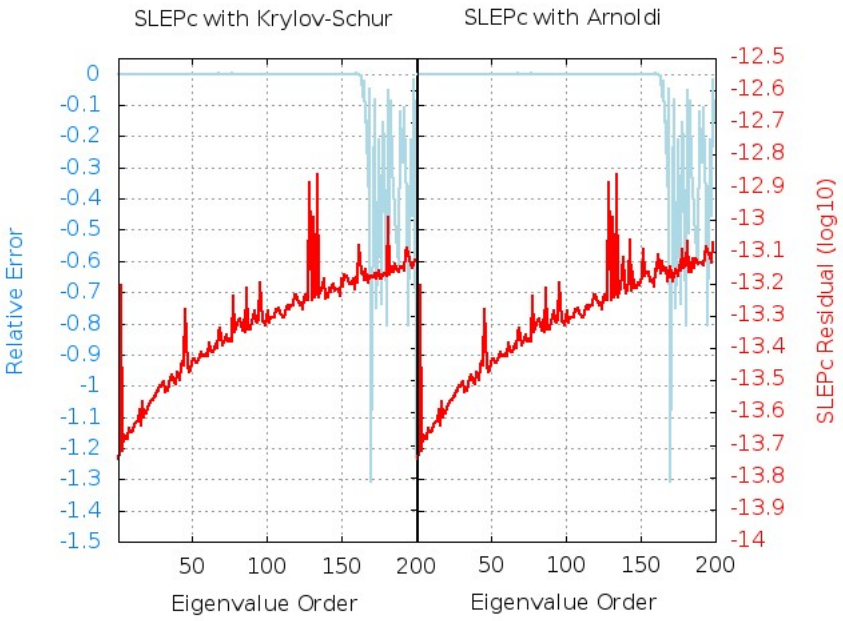

Figure A.2: Dense Ex11 Matrix, SLEPc Eigen Solvers Numerical Comparison

The blue line refers to the residual error while the red line refers to the SLEPc computed eigenpairs residual. The SLEPc eigen solvers have been executed with 50 MPI tasks, using the thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. We fixed $s = 200$ and $m = 400$ (the default SLEPc value). The SLEPc Arnoldi (respectively Krylov-Schur) eigen solver stopped after 3 (respectively 2) restarts.

A.0.2.2 The Mixtank_new Matrix

The Mixtank_new matrix is issued from the University of Florida matrices collection [Davis]. This matrix is originally sparse with 1,995,041 non-zeros elements. The Figure A.3 shows the *Mixtank_new* eigenvalues distribution:

Appendix

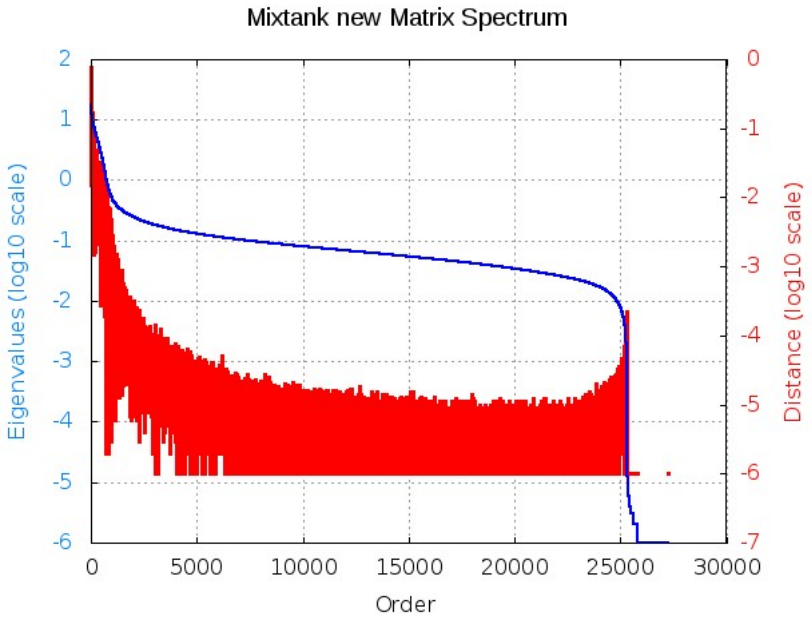

Figure A.3: The Mixtank_new Matrix Spectrum, Size is $29,957$

The presented spectrum contains the original Mixtank_new matrix real modulus eigenvalues (blue line). The red points present the distance between each successive real modulus eigenvalue. The Eigenvalues and Distance metrics use both log_{10} scale to enhance the eigenvalues distribution visibility.

Using the Mixtank_new spectrum as input parameter of the Algorithm 11 provides a *Dense Mixtank_new* matrix with $897,361,938$ non-zeros elements. We then computed the 200 dominant eigenvalues of the *Dense Mixtank_new* matrix using the SLEPc Arnoldi and Krylov-Schur eigen solvers. The tests have been executed with 50 MPI tasks using the thin nodes of the PRACE Curie supercomputer. The subspace size is automatically fixed to 400 by the SLEPc eigen solvers.

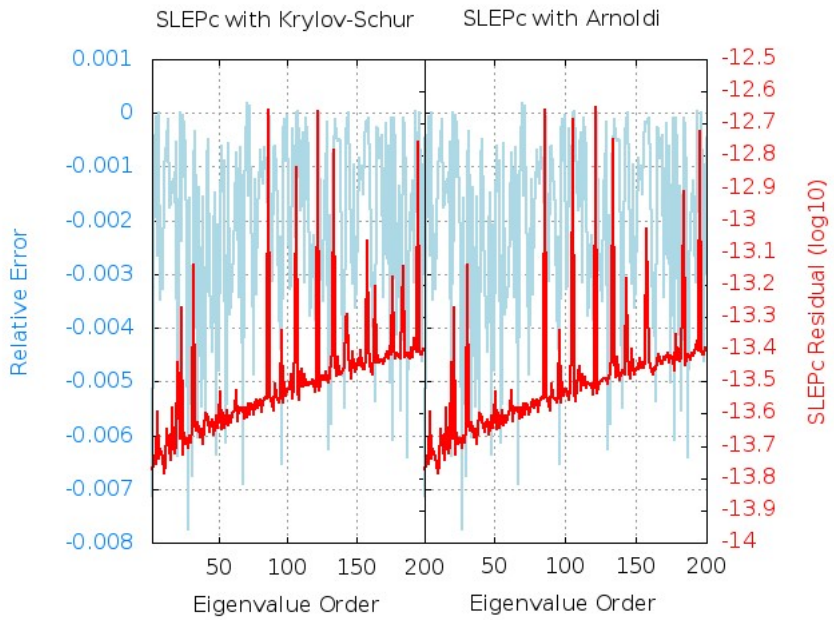

Figure A.4: Dense Mixtank_new Matrix, SLEPc Eigen Solvers Numerical Comparison

The blue line refers to the residual error while the red line refers to the SLEPc computed eigenpairs residual. The SLEPc eigen solvers have been executed with 50 MPI tasks, using the thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. We fixed $s = 200$ and $m = 400$ (the default SLEPc value). The SLEPc Arnoldi (respectively Krylov-Schur) eigen solver reached the convergence after 6 (respectively 3) restarts.

The relative errors with respect to the associated residuals are satisfying, showing that the *Dense Mixtank_new* matrix conserved the spectrum with respect to the Algorithm 11. The *Mixtank_new* eigenvalues are relatively small and simple, therefore the relative error is not as disrupted as the Ex11 matrix (cf A.0.2.1).

A.0.2.3 The Rim Matrix

The Rim matrix is extracted from the University of Florida collection [Davis]. This matrix has originally $1,014,951$ non-zeros elements. We present on the Figure A.5 the Rim matrix spectrum.

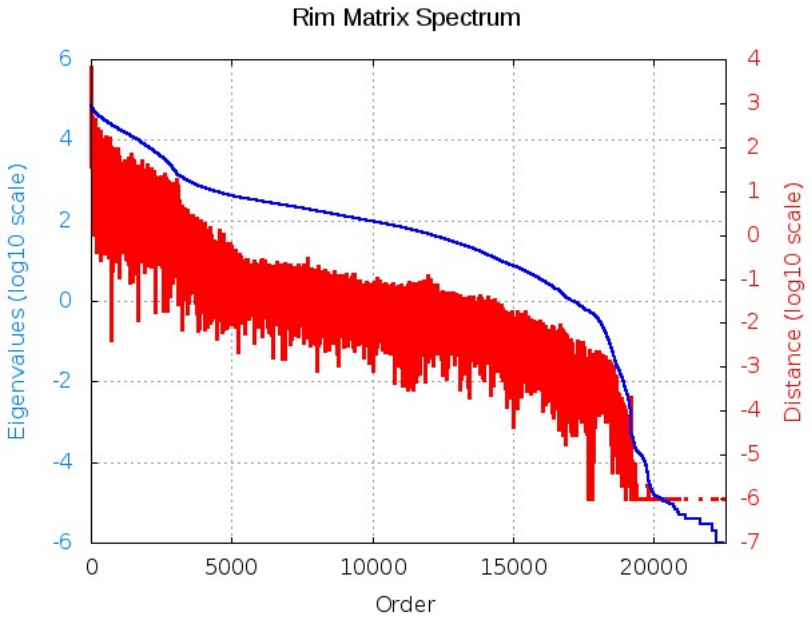

Figure A.5: The Rim Matrix Spectrum, Size is $22,560$

The presented spectrum contains the original Rim matrix real modulus eigenvalues (blue line) without any modification. The red points present the distance between each successive real modulus eigenvalue. The Eigenvalues and Distance metrics use both log_{10} scale to enhance the eigenvalues distribution visibility.

The matrix *Dense Rim* resulting from the Algorithm 11 has $508,908,483$ non-zeros elements. We compute the 200 dominant eigen values of *Dense Rim* matrix, using the SLEPc Krylov-Schur and Arnoldi solvers. Both of them were executed with 50 MPI tasks using the thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer and reached the convergence after 112 restarts. The Figure A.6 shows satisfying residuals and relative errors regarding the 200 dominant eigenvalues.

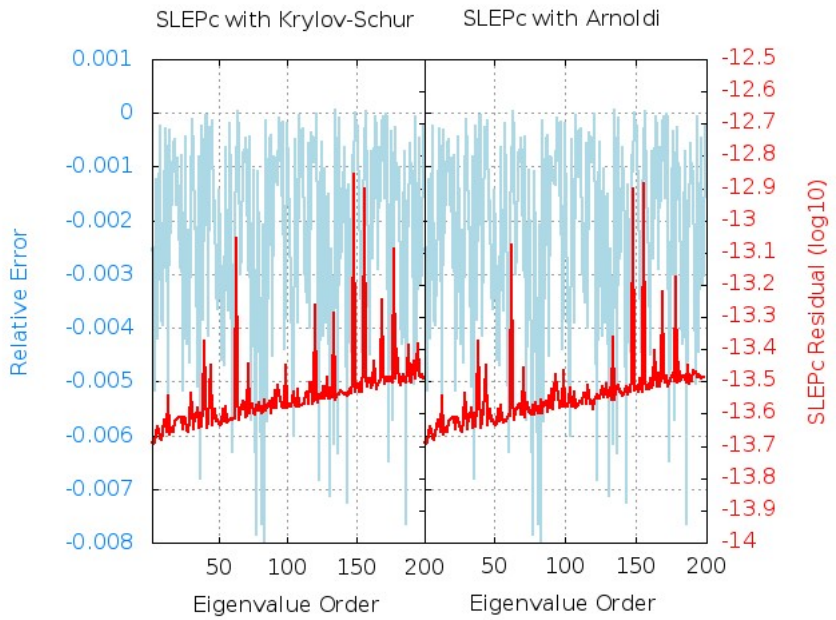

Figure A.6: Dense Rim Matrix, SLEPc Eigen Solvers Numerical Comparison

The blue line refers to the residual error while the red line refers to the SLEPc computed eigenpairs residual. The SLEPc eigen solvers have been executed with 50 MPI tasks, using the thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. We fixed $s = 200$ and $m = 400$ (the default SLEPc value). The SLEPc Arnoldi (respectively Krylov-Schur) eigen solver reached the convergence after 20 (respectively 4) restarts.

According to the residuals values presented on Figure A.6 we admit that the Ritz eigenpairs corresponds to the prescribed spectrum due to the relative error small values.

Appendix B

The ERAM with Mixed-Restarting Strategies

B.0.2.4 Mixtank_new$_{Dense}$

We continue our study with the *Mixtank_new* Dense matrix. We summarize in the following Table B.0.2.4 the results obtained by the Algorithm 10 with an ERAM using the α_{Def} restarting strategy.

	Stagnation			
Interval	{5}	[22,27]	[32,40]	[47,77]
Lowest Residual	4	18	31	41

Table B.1: The ERAM has $m = 15$, $s = \gamma = 4$, a CGSR orthogonalization process and the restarting strategy α_{Def}. We used 29 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. We summarize in this Table the results of the Algorithm 10.

The results obtained for the *Mixtank_new* Dense matrix are not as satisfying as the $Fission_{Dense}$ or $Ex11_{Dense}$ matrices but still remain interesting. We recall that the ERAM using a single strategy during its complete execution had the best convergence by using the α_{Def} restarting strategy (cf Figure 6.6).

Using the mixed restarting strategies improves the ERAM convergence but does not provide the desired Ritz eigenpairs at the desired threshold (10^{-14}).

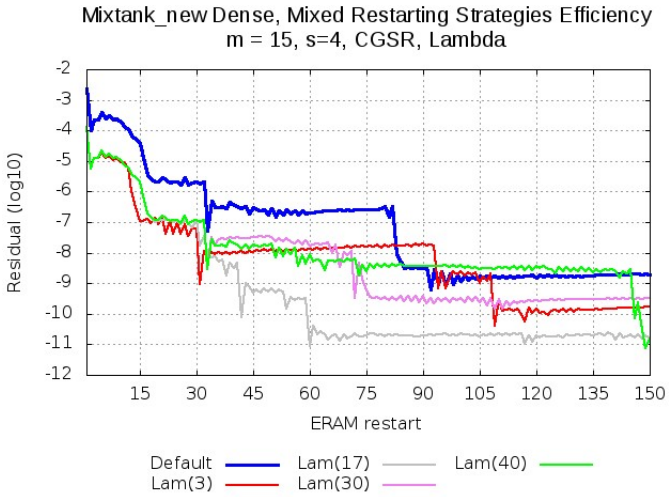

Figure B.1: Mixtank_new Dense Matrix, α_{Def}/ α_{La} Mixed Restarting Strategies.

The ERAM has $m = 15$, $s = \gamma = 4$, a CGSR orthogonalization process and the restarting strategy α_{Def}. We present the dominant eigenvalue residuals. We replaced α_{Def} by α_{La} at the restarts listed in the Table 7.2.0.1. We used 29 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer.

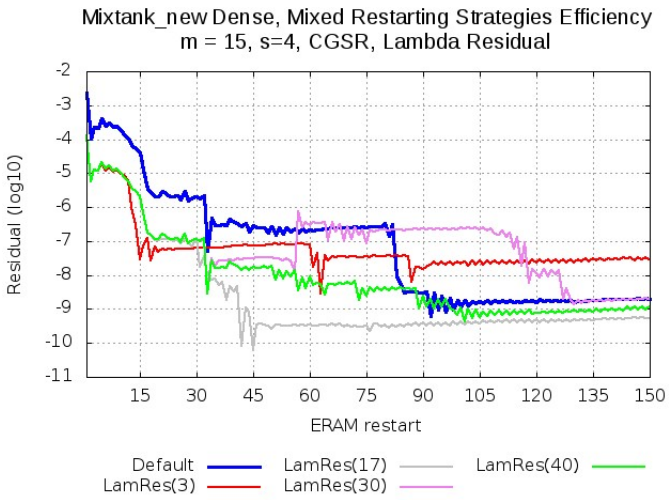

Figure B.2: Mixtank_new Dense Matrix, α_{Def}/ α_{LaRes} Mixed Restarting Strategies.

The ERAM has $m = 15$, $s = \gamma = 4$, a CGSR orthogonalization process and the restarting strategy α_{Def}. We present the dominant eigenvalue residuals. We replaced α_{Def} by α_{LaRes} at the restarts listed in the Table 7.2.0.1. We used 29 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer.

Appendix

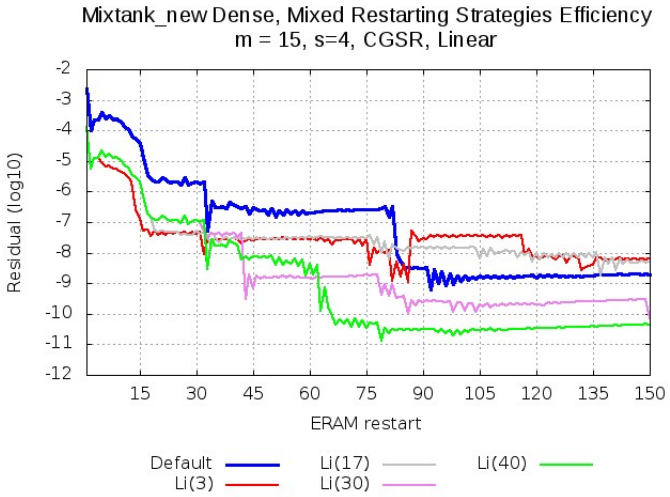

Figure B.3: Mixtank_new Dense Matrix, α_{Def}/ α_{Li} Mixed Restarting Strategies.

The ERAM has $m = 15$, $s = \gamma = 4$, a CGSR orthogonalization process and the restarting strategy α_{Def}. We present the dominant eigenvalue residuals. We replaced α_{Def} by α_{Li} at the restarts listed in the Table 7.2.0.1. We used 29 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer.

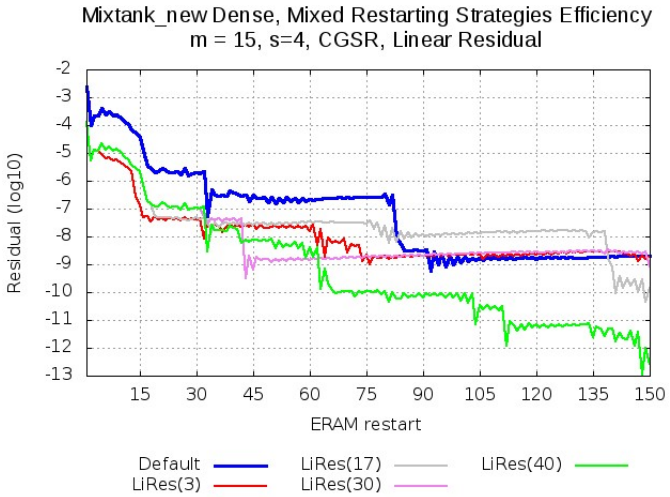

Figure B.4: Mixtank_new Dense Matrix, α_{Def}/ α_{LiRes} Mixed Restarting Strategies.

The ERAM has $m = 15$, $s = \gamma = 4$, a CGSR orthogonalization process and the restarting strategy α_{Def}. We present the dominant eigenvalue residuals. We replaced α_{Def} by α_{LiRes} at the restarts listed in the Table 7.2.0.1. We used 29 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer.

B.0.2.5 Diagonal Band 2^{15}

	150		160		170		180		190		200	
150	**0,15**	**1,45**	**0,01**	**0,01**	**0,02**	**0,03**	**0,03**	**0,07**	**0,04**	**0,12**	**0,05**	**0,18**
160	#	#	**0,16**	**1,64**	**0,01**	**0,01**	**0,02**	**0,03**	**0,03**	**0,07**	**0,04**	**0,12**
170	#	#	#	#	**0,17**	**1,85**	**0,01**	**0,01**	**0,02**	**0,03**	**0,03**	**0,07**
180	#	#	#	#	#	#	**0,18**	**2,07**	**0,01**	**0,01**	**0,02**	**0,03**
190	#	#	#	#	#	#	#	#	**0,19**	**2,3**	**0,01**	**0,01**
200	#	#	#	#	#	#	#	#	#	#	**0,2**	**2,54**

Table B.2: We present in this Table the **memory gain (Goctets)** and the **Flop gain (GFlop)** between each ERAM with respect to the subspace size we used. The diagonal represents the global **memory storage** (respectively **Flop**) used for a complete ERAM restart. All the other cases present the gain of a complete ERAM restart using two different subspace size.

Appendix

Single / Mixed	α_{Def}			α_{Res}			α_{Li}		
	180	190	200	180	190	200	180	190	200
$\{m = 150, \alpha_{LaRes}, it = 3\}$	**13,83**	**8**	**4,89**	**8,61**	**13,17**	**5,56**	**11,72**	**4,89**	**6,11**
$\{m = 150, \alpha_{LaRes}, it = 6\}$	**13,11**	**7,58**	**4,63**	**8,16**	**12,47**	**5,26**	**11,11**	**4,63**	**5,79**
$\{m = 150, \alpha_{LaRes}, it = 14\}$	**8,3**	**4,8**	**2,93**	**5,17**	**7,9**	**3,33**	**7,03**	**2,93**	**3,67**
$\{m = 150, \alpha_{LaRes}, it = 24\}$	**6,07**	**3,51**	**2,15**	**3,78**	**5,78**	**2,44**	**5,15**	**2,15**	**2,68**
$\{m = 150, \alpha_{Res}, it = 3\}$	**1,45**	0,84	0,51	0,9	**1,38**	0,58	**1,23**	0,51	0,64
$\{m = 150, \alpha_{Res}, it = 14\}$	**1,1**	0,64	0,39	0,69	1,05	0,44	0,93	0,39	0,49
$\{m = 160, \alpha_{LaRes}, it = 25\}$	**6,07**	**3,51**	**2,15**	**3,78**	**5,78**	**2,44**	**5,15**	**2,15**	**2,68**
$\{m = 160, \alpha_{LaRes}, it = 27\}$	**6,07**	**3,51**	**2,15**	**3,78**	**5,78**	**2,44**	**5,15**	**2,15**	**2,68**
$\{m = 170, \alpha_{Li}, it = 3\}$	**1,05**	0,61	0,37	0,65	1	0,42	0,89	0,37	0,46
$\{m = 180, \alpha_{La}, it = 6\}$	**1,38**	0,8	0,49	0,86	**1,32**	0,56	**1,17**	0,49	0,61
$\{m = 180, \alpha_{La}, it = 13\}$	1	0,58	0,35	0,62	0,95	0,4	0,84	0,35	0,44
$\{m = 180, \alpha_{La}, it = 34\}$	**1,39**	0,8	0,49	0,87	**1,32**	0,56	**1,18**	0,49	0,61
$\{m = 180, \alpha_{LaRes}, it = 6\}$	**13,11**	**7,58**	**4,63**	**8,16**	**12,47**	**5,26**	**11,11**	**4,63**	**5,79**
$\{m = 180, \alpha_{LaRes}, it = 13\}$	**10,83**	**6,26**	**3,83**	**6,74**	**10,3**	**4,35**	**9,17**	**3,83**	**4,78**
$\{m = 180, \alpha_{LaRes}, it = 34\}$	**5,53**	**3,2**	**1,96**	**3,44**	**5,27**	**2,22**	**4,69**	**1,96**	**2,44**
$\{m = 180, \alpha_{Li}, it = 6\}$	0,93	0,54	0,33	0,58	0,88	0,37	0,79	0,33	0,41
$\{m = 180, \alpha_{Li}, it = 13\}$	0,97	0,56	0,34	0,6	0,92	0,39	0,82	0,34	0,43
$\{m = 180, \alpha_{Res}, it = 6\}$	**1,31**	0,76	0,46	0,82	**1,25**	0,53	**1,11**	0,46	0,58
$\{m = 180, \alpha_{Res}, it = 13\}$	**1,07**	0,62	0,38	0,67	**1,02**	0,43	0,91	0,38	0,47
$\{m = 190, \alpha_{La}, it = 2\}$	**1,48**	0,86	0,52	0,92	**1,41**	0,6	**1,26**	0,52	0,65
$\{m = 190, \alpha_{La}, it = 7\}$	**1,46**	0,84	0,51	0,91	**1,39**	0,58	**1,23**	0,51	0,64
$\{m = 190, \alpha_{La}, it = 15\}$	1	0,58	0,35	0,63	0,96	0,4	0,85	0,35	0,44
$\{m = 190, \alpha_{LaRes}, it = 2\}$	**20,75**	**12**	**7,33**	**12,92**	**19,75**	**8,33**	**17,58**	**7,33**	**9,17**
$\{m = 190, \alpha_{LaRes}, it = 7\}$	**15,56**	**9**	**5,5**	**9,69**	**14,81**	**6,25**	**13,19**	**5,5**	**6,88**
$\{m = 190, \alpha_{LaRes}, it = 15\}$	**9,96**	**5,76**	**3,52**	**6,2**	**9,48**	**4**	**8,44**	**3,52**	**4,4**
$\{m = 190, \alpha_{Li}, it = 2\}$	**1,77**	**1,02**	0,62	**1,1**	**1,68**	0,71	**1,5**	0,62	0,78
$\{m = 190, \alpha_{Li}, it = 7\}$	**2,65**	**1,53**	0,94	**1,65**	**2,52**	**1,06**	**2,24**	0,94	**1,17**
$\{m = 190, \alpha_{Li}, it = 15\}$	**2,24**	**1,3**	0,79	**1,4**	**2,14**	0,9	**1,9**	0,79	0,99
$\{m = 190, \alpha_{Res}, it = 15\}$	**2,52**	**1,45**	0,89	**1,57**	**2,39**	1,01	**2,13**	0,89	**1,11**
$\{m = 200, \alpha_{La}, it = 5\}$	**2,83**	**1,64**	1	**1,76**	**2,69**	**1,14**	**2,4**	1	**1,25**
$\{m = 200, \alpha_{La}, it = 26\}$	**2,71**	**1,57**	0,96	**1,68**	**2,58**	**1,09**	**2,29**	0,96	**1,2**
$\{m = 200, \alpha_{LaRes}, it = 5\}$	**17,79**	**10,29**	**6,29**	**11,07**	**16,93**	**7,14**	**15,07**	**6,29**	**7,86**
$\{m = 200, \alpha_{LaRes}, it = 26\}$	**7,78**	**4,5**	**2,75**	**4,84**	**7,41**	**3,13**	**6,59**	**2,75**	**3,44**
$\{m = 200, \alpha_{Li}, it = 5\}$	**2,57**	**1,48**	0,91	**1,6**	**2,44**	**1,03**	**2,18**	0,91	**1,13**
$\{m = 200, \alpha_{Li}, it = 26\}$	**2,71**	**1,57**	0,96	**1,68**	**2,58**	**1,09**	**2,29**	0,96	**1,2**
$\{m = 200, \alpha_{Res}, it = 26\}$	**4,29**	**2,48**	**1,52**	**2,67**	**4,09**	**1,72**	**3,64**	**1,52**	**1,9**

Table B.3: Diagonal Band 2^{15} Matrix, Mixed Restarting Strategies Mixed Restarting Strategies Gains/Losses.

We compare the gains/losses (in terms of number of restarts until the convergence) of ERAM with mixed restarting strategies with ERAM using a single restarting strategy during its complete execution. Each column corresponds to one ERAM using a single restarting strategy. Each line corresponds to one execution of ERAM using mixed restarting strategies (presented in Figure 7.3).

Single / Mixed	α_{La}		
	180	**190**	**200**
$\{m = 150, \alpha_{LaRes}, it = 3\}$	**11,94**	**9,83**	**5,44**
$\{m = 150, \alpha_{LaRes}, it = 6\}$	**11,32**	**9,32**	**5,16**
$\{m = 150, \alpha_{LaRes}, it = 14\}$	**7,17**	**5,9**	**3,27**
$\{m = 150, \alpha_{LaRes}, it = 24\}$	**5,24**	**4,32**	**2,39**
$\{m = 150, \alpha_{Res}, it = 3\}$	**1,25**	**1,03**	0,57
$\{m = 150, \alpha_{Res}, it = 14\}$	0,95	0,78	0,43
$\{m = 160, \alpha_{LaRes}, it = 25\}$	**5,24**	**4,32**	**2,39**
$\{m = 160, \alpha_{LaRes}, it = 27\}$	**5,24**	**4,32**	**2,39**
$\{m = 170, \alpha_{Li}, it = 3\}$	0,91	0,75	0,41
$\{m = 180, \alpha_{La}, it = 6\}$	**1,19**	0,98	0,54
$\{m = 180, \alpha_{La}, it = 13\}$	0,86	0,71	0,39
$\{m = 180, \alpha_{LaRes}, it = 6\}$	**1,2**	0,99	0,55
$\{m = 180, \alpha_{LaRes}, it = 13\}$	**11,32**	**9,32**	**5,16**
$\{m = 180, \alpha_{LaRes}, it = 34\}$	**9,35**	**7,7**	**4,26**
$\{m = 180, \alpha_{Li}, it = 6\}$	**4,78**	**3,93**	**2,18**
$\{m = 180, \alpha_{Li}, it = 13\}$	0,8	0,66	0,37
$\{m = 180, \alpha_{Res}, it = 6\}$	0,84	0,69	0,38
$\{m = 180, \alpha_{Res}, it = 13\}$	**1,13**	0,93	0,52
$\{m = 180, \alpha_{Res}, it = 34\}$	0,93	0,76	0,42
$\{m = 190, \alpha_{La}, it = 2\}$	**1,28**	**1,05**	0,58
$\{m = 190, \alpha_{La}, it = 7\}$	**1,26**	**1,04**	0,57
$\{m = 190, \alpha_{La}, it = 15\}$	0,87	0,71	0,4
$\{m = 190, \alpha_{LaRes}, it = 2\}$	**17,92**	**14,75**	**8,17**
$\{m = 190, \alpha_{LaRes}, it = 7\}$	**13,44**	**11,06**	**6,13**
$\{m = 190, \alpha_{LaRes}, it = 15\}$	**8,6**	**7,08**	**3,92**
$\{m = 190, \alpha_{Li}, it = 2\}$	**1,52**	**1,26**	0,7
$\{m = 190, \alpha_{Li}, it = 7\}$	**2,29**	**1,88**	**1,04**
$\{m = 190, \alpha_{Li}, it = 15\}$	**1,94**	**1,59**	0,88
$\{m = 190, \alpha_{Res}, it = 15\}$	**2,17**	**1,79**	0,99
$\{m = 200, \alpha_{La}, it = 5\}$	**2,44**	**2,01**	**1,11**
$\{m = 200, \alpha_{La}, it = 26\}$	**2,34**	**1,92**	**1,07**
$\{m = 200, \alpha_{LaRes}, it = 5\}$	**15,36**	**12,64**	**7**
$\{m = 200, \alpha_{LaRes}, it = 26\}$	**6,72**	**5,53**	**3,06**
$\{m = 200, \alpha_{Li}, it = 5\}$	**2,22**	**1,82**	**1,01**
$\{m = 200, \alpha_{Li}, it = 26\}$	**2,34**	**1,92**	**1,07**
$\{m = 200, \alpha_{Res}, it = 26\}$	**3,71**	**3,05**	**1,69**

Table B.4: Diagonal Band 2^{15} , Mixed Restarting Strategies Gains/Losses.

We compare the gains/losses (in terms of number of restarts until the convergence) of ERAM with mixed restarting strategies with ERAM using a single restarting strategy during its complete execution. Each column corresponds to one ERAM using a single restarting strategy. Each line corresponds to one execution of ERAM using mixed restarting strategies (presented in Figure 7.3).

Appendix C

The ERAM with Mixed-Restarting Strategies Smart-Tuning

C.0.2.6 Diagonal Band 2^{16} Matrix

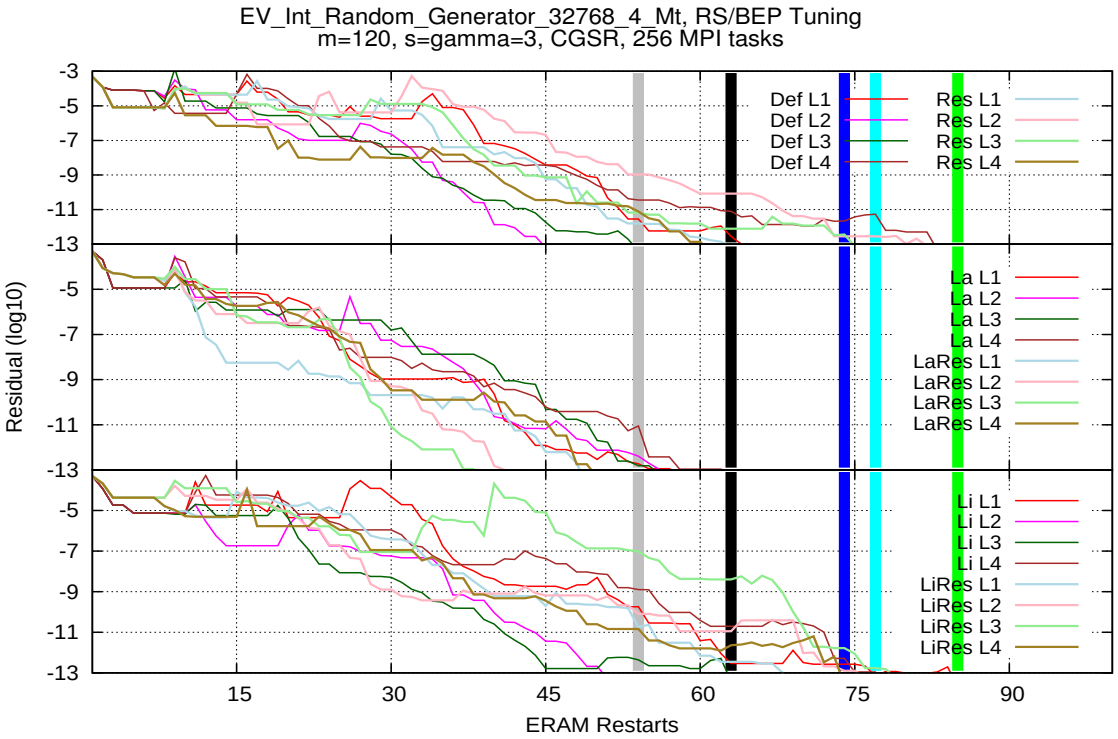

Figure C.1: Diagonal Band 2^{16} Matrix, Mixed Restarting Strategies/Best Ritz Eigenpairs Tuning.

The ERAM has $m = 120$, $s = \gamma = 3$, a CGSR orthogonalization process. We present the dominant eigenvalue residuals. We used 256 MPI tasks on thin nodes (2 processors Intel® Sandy-Bridge with 8 cores) of the PRACE Curie supercomputer. Each stripe color corresponds to the ERAM using respectively $\{\alpha_{LaRes}, \alpha_{La}, \alpha_{Def}, \alpha_{Res}, \alpha_{Li}\}$ restarting strategy during its complete execution. The ERAMs using α_{LiRes} as a single restarting strategy did not converge. We presented separately the twin restarting strategies (α_{Def},α_{Res} then α_{La}, α_{LaRes} and finally α_{Li},α_{LiRes}). For each restarting strategy, we present 4 different executions of the Algorithm 17 to show its nondeterministic behavior. As an illustration, La L1 refers to one execution of the Algorithm 17 starting with the α_{La} restarting strategy.