Hybrid direct and interactive solvers for sparse indefinite and overdetermined systems on future exascale architectures

Philippe Leleux

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National Polytechnique de Toulouse (Toulouse INP)

Discipline ou spécialité :

Mathématiques Appliquées

Présentée et soutenue par :

M. PHILIPPE LELEUX

le mercredi 3 février 2021

Titre :

Hybrid direct and interactive solvers for sparse indefinite and overdetermined systems on future exascale architectures

---

École doctorale :

Mathématiques, Informatique, Télécommunications de Toulouse (MITT)

Unité de recherche :

Centre Européen de Recherche et Formation Avancées en Calcul Scientifique (CERFACS)

Directeurs de Thèse :

M. MICHEL DAYDE

M. DANIEL RUIZ

M. RÜDE ULRICH

Rapporteurs :

M. ANDREAS FROMMER, UNIVERSITE DE WUPPERTAL

Membres du jury :

M. FRÉDÉRIC NATAF, UNIVERSITE PARIS 6, Président

M. DANIEL RUIZ, TOULOUSE INP, Membre

M. HARALD KOESTLER, UNIVERSITE D'ERLANGEN-NUREMBERG, Membre

M. IAIN DUFF, RUTHERFORD APPLETON LABORATORY OXFORD, Invité

M. MATTHIAS BOLTEN, UNIVERSITE DE WUPPERTAL, Invité

MME ALISON RAMAGE, UNIVERSITY STRATHCLYDE GLASGOW, Membre

MME HEIKE FASSBENDER, , Membre

M. MICHEL DAYDE, TOULOUSE INP, Membre

M. ULRICH RÜDE, UNIVERSITE D'ERLANGEN-NUREMBERG, Membre

Hybride direkte und iterative Löser

für dünn besetzte indefinite und überbestimmte Systeme auf zukünftigen Exascale-Architekturen

Der Technische Fakultät, Lehrstuhl Informatik 10 (Systemsimulation)
der Friedrich-Alexander-Universität
Erlangen-Nürnberg
zur
Erlangung des Doktorgrades

Doktor-Ingenieur (Dr.-Ing.)

vorgelegt von

Philippe Leleux

aus Toulouse

Als Dissertation genehmigt
von der Technische Fakultät, Lehrstuhl Informatik 10 (Systemsimulation)
der Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der mündlichen Prüfung:	der 03 Februar 2021
Vorsitzender des Promotionsorgans:	Prof. Dr.-Ing. habil. Andreas Paul Fröba
Gutachter:	Prof. Frédéric Nataf
	Prof. Dr. Andreas Frommer

Aknowledgements / Remerciements

> It really seems to me that in the midst of great tragedy, there is always the possibility that something terribly funny will happen.
>
> K. Dick

There are so many people to thank for how this PhD turned out to be that I do not know where to start. A PhD is an incredible experience in terms of research but also on a human perspective. So many people were involved to help and support me that I am afraid to forget any. So I will start with a big THANK YOU to all of you who accompanied me in this adventure, and also to the people who will actually manage to hold until the end of this "doorstop" !

Il y a tellement de personnes que je voudrais remercier pour cette thèse que je ne sais pas où commencer. Une thèse est une expérience incroyable que ce soit en terme de recherche ou d'un point de vue humain. Tellement de gens m'ont aidé pendant ces 3 années que j'ai peur d'en oublier. Je vais donc commencer par un énorme MERCI à tous ceux qui m'ont accompagné dans cette aventure, et aussi à tous ceux qui vont vraiment lire jusqu'au bout ce "pavé" !

Doing a PhD is not always an easy thing and my supervisors where the one who made it possible. I would like to thank Daniel Ruiz first. We should have believed him from the start when he said: doing a PhD is like a marathon, and writing the thesis is like delivering a baby. Thank you Daniel for always giving the right advice, and for teaching me in length about linear algebra... and jokes ! Also thanks for finding the time to discuss even though you are always so overbooked. I hope we can have a beer at the "Poêle de la bête" soon with everyone to celebrate this 3-years work ! Then I thank Ulrich Ruede who introduced me to this great research field that are multigrid methods. You were always here with good advice and to help me find the right directions. You were there supporting me even when seemingly reaching dead-ends in research, but these are pretty natural since the nastiness is always conserved ! Thank you also for giving me the opportunity to have this cotutelle, it was not easy but we made it. I am looking forward to future visits in Erlangen for some new biergarten. Finally, I would like to thank Michel Dayde who supervised my PhD on very specific aspects. The PhD would not have been possible without you.

I would then like to thank first the 2 reviewers of this dissertation, F. Nataf and A. Frommer, for their kind words and comments. Including them, I could not have wished for better jury and guests for

the PhD defense, so thanks to all of you. In a more general sense, this thesis was only possible thanks to the funding from CERFACS, together with hosting at IRIT-ENSEEIHT and FAU Erlangen-Nürnberg.

In all of these places there were kind people that I was happy to meet. First thank you to the algo team at CERFACS, it is really a team that feels like family. Thanks to you I also learned the importance of coffee breaks in science ! Please stay just the way you all are. Also thank you to the APO team at IRIT and the LSS at FAU, I was very lucky to find 3 laboratories where it is so pleasant to stay. In particular, I would like to thank my fellow PhD students Pierre, Nicolas, Luce, Auguste, Theo, Thibaut, Thibaud, & Oliver. Pierre, tu vas y arriver, la cotutelle ça marche au final ! Among my colleagues, there are way too many great people I would like to thank but I cannot write a 10-pages acknowledgments... or can I ? Aussi un grand merci à toutes les personnes de l'administration du CERFACS, en particulier Chantal et Michèle pour votre bonne humeur. Un merci particulier à Brigitte, la première personne que j'ai rencontré au CERFACS et la pierre angulaire de l'équipe algo, sans toi l'équipe serait complètement incapable de tourner en rond ! Je remercie également Agnès Requis de l'EDMITT, qui semble être la seule personne vraiment là pour aider les thésards, EDMITT et INPT confondus.

Maintenant, il n'y a pas que le travail dans la vie et clairement pour tenir le coup pendant 3 ans de travail acharné, surtout pendant cette dernière année très très spéciale, il faut être bien entouré ! Je tiens donc à remercier ma famille. Merci maman et papa pour votre aide et pour m'avoir toujours soutenu même dans les moment difficiles. C'est cliché mais sans vous je ne serais jamais devenu ce que je suis maintenant ! Merci aussi à Marie, Thierry et leur petite famille, je vous fais tous pleins de bisous !

Une pensée très spéciale à Mamaman et Mamie qui ont grandement participé à me construire, et qui m'ont appris à aimer la bonne cuisine.

Il aurait été difficile d'en arriver là sans tous les amis qui m'entourent. Je pense à vous les chattons de Fermat, on va pouvoir reprendre les apéros et les parties de Bodega ! Je pense aussi à vous les copains de Barthou et de l'n7, avec un peu de chance je pourrai vous rendre visite maintenant ! Sans écran interposé quoi...
Merci à toi aussi mon fillot, pour ta bonne humeur et parce que tu chippotes.. Oh ça va on plaisante ! On plaisante on plaisante.. A quand le prochain film ?
Bien sûr j'en oublie encore tant d'autres, merci à vous ! Et je ne t'oublies pas Camille, mon amour, merci pour le temps que nous avons passé tous les deux et m'avoir supporté même quand je fais semblant d'écouter tout en pensant au boulot. Tu as réussi à organiser le meilleur pot de thèse maison, même quand c'était interdit ! Et les gens vous craquez sur les cadeaux... Je vais essayer de pas me couper avec la trancheuse promis. Maintenant une proposition: Allongés sur la plage avec un cocktail bien mérité après une randonnée au bord d'un volcan ? On va enfin pouvoir aller prendre le soleil à l'autre bout du monde !

PhD... check.
Dissertation... check.
Defense... check.
But the PhD is not the end, it is only the beginning !

Contents

List of Figures **ix**

List of Tables **xvii**

Introduction **1**

I Massively parallel multigrid **7**

I.1 PDEs and their discretization 8

I.1.1 Classical PDE problems 8

I.1.1.a Incompressible fluid flow 8

I.1.1.b Wave equation 9

I.1.1.c Boundary conditions 10

I.1.2 Discretization schemes 10

I.1.3 Stokes on a spherical shell 11

I.2 Linear solvers and HPC 13

I.2.1 High Performance Computing 13

I.2.1.a Sequential performance: NUMA and ILP 13

I.2.1.b Parallel programming 15

I.2.1.c Numerical linear algebra 17

I.2.2 Hybrid solvers 22

I.2.2.a Domain decomposition methods 22

I.2.2.b Multigrid methods 25

I.2.3 Multigrid solution of saddle-point problems 30

I.2.3.a Hierarchical hybrid grids 31

I.2.3.b Coarse grid solver 32

I.3 Approximate Coarse Grid Solvers 34

I.3.1 Agglomeration 34

I.3.2 MUMPS 36

I.3.2.a Method . . . 36
I.3.2.b Block low-rank approximation . . . 37
I.3.3 Scaling experiments . . . 38
I.3.3.a Approximate sparse direct solver with agglomeration . . 38
I.3.3.b Multigrid solver combined with the approximate coarse level solver . . . 40

II Solution of full rank linear systems 45
II.1 Block projection methods . . . 46
II.1.1 The Cimmino and Kaczmarz methods . . . 46
II.1.2 The block Cimmino method . . . 49
II.1.2.a The iteration matrices . . . 50
II.1.2.b Computing the projections . . . 51
II.1.2.c Block Cimmino as a domain decomposition method . . . 51
II.2 CG-accelerated block Cimmino . . . 52
II.2.1 Solving underdetermined systems . . . 53
II.2.1.a Stabilised block-CG . . . 53
II.2.1.b The stabilisation process . . . 54
II.2.1.c Computation of the projections . . . 55
II.2.2 Solving least-squares problems . . . 56
II.3 The augmented block Cimmino method . . . 60
II.3.1 Enforcing the orthogonality between spaces . . . 60
II.3.1.a The augmentation scheme . . . 61
II.3.1.b Properties of the augmented matrix . . . 63
II.3.1.c Decoupling subdomains through a Schur complement . . 67
II.3.2 Computing the pseudo-direct solution . . . 71
II.3.2.a Underdetermined systems . . . 71
II.3.2.b Least-squares problems . . . 73
II.4 Numerical Experiments . . . 77
II.4.1 Accelerated block Cimmino iterations . . . 77
II.4.2 Augmented block Cimmino . . . 81
II.4.3 Handling unsymmetric square matrices . . . 83

III Parallel implementation 87
III.1 Preprocessing . . . 88
III.1.1 Scaling . . . 88
III.1.1.a Iterative block Cimmino . . . 89

III.1.1.b Augmented block Cimmino 89
III.1.2 Partitioning 91
III.1.2.a Targeting the angles with partitioning 92
III.1.2.b Reduction of bandwidth for 2-blocks partitioning 95
III.1.2.c Graph partitioning 96
III.1.2.d Partitioning in practice 99
III.1.2.e Overlapping partitioning 101
III.1.3 Alternative techniques for the augmentation 105
III.1.3.a Normal equations duplication 105
III.1.3.b Straight forward duplication 106
III.1.3.c Full rank augmentation 107
III.1.3.d Augmentation in practice 109
III.2 Parallel implementation scheme 112
III.2.1 Hybrid parallelism 112
III.2.2 Improving the scalability of the ABCD-Solver 113
III.2.2.a Load balancing: distribution of partitions 115
III.2.2.b Placement of masters and workers 116
III.2.3 Parallel experiments 118
III.2.3.a Communication reducing distribution of partitions 119
III.2.3.b Parallel solution of the Schur complement 120
III.2.3.c Node aware placement of master and worker processes . . 122
III.2.3.d Comparison with direct solvers 124

IV Multigrid-based ABCD **131**
IV.1 Controlling the Schur complement 132
IV.1.1 Using coarser levels 132
IV.1.2 Challenging PDE problems 134
IV.1.2.a Domains and discretization 134
IV.1.2.b Test problems 134
IV.1.2.c Multigrid elements and partitioning 136
IV.2 Enforcing a relaxed orthogonality 138
IV.2.1 General discussion on ways to construct a relaxed augmentation . 139
IV.2.2 A relaxed ABCD 143
IV.2.2.a Size of the augmentation 144
IV.2.2.b Widening the angles between partitions 146
IV.2.3 The relaxed augmented block Cimmino method 146

IV.3 A block Cimmino approximation of the closure equations 154
IV.3.1 Construction of W 154
IV.3.2 Approximation of W and convergence of a global BC 158
IV.3.2.a Block-CG size for the approximation of W 158
IV.3.2.b Stopping criterion for the Block-CG approximation of W 160
IV.4 Implicit approximation of the Schur complement 162
IV.4.1 Iterative solution with S 163
IV.4.2 Preconditioning S 164

Conclusion and perspectives **176**

Bibliography **177**

Appendix A Solution of full rank linear systems **191**
A.1 Block Cimmino as a Domain Decomposition Method 191
A.2 Impact of the block size for the accelerated block Cimmino 193
A.3 Handling unsymmetric square matrices with the ABCD-Solver 194

Appendix B Parallel implementation **197**
B.1 Preprocessing 197
B.1.1 PaToH vs GRIP partitioner 197
B.1.2 Matrix structure and augmentations $\mathcal{F}^{C_{ij}}$ and $\mathcal{F}^{FR}$ 199
B.2 Architecture aware placement of masters and workers 202

Appendix C Multigrid-inspired relaxed ABCD **205**
C.1 Overdetermined augmentation as a multigrid scheme 205
C.2 C-ABCD parallelism with the construction of W 208

Appendix D Résumé étendu **211**
D.1 Multigrille à très grande échelle 213
D.1.1 Problèmes à point de selle 214
D.1.2 Problème de scalabilité 214
D.1.3 Solveur sur la grille grossière 215
D.2 Méthodes Cimmino par blocs, itérative et pseudo-directe 221
D.2.1 Accélération de la méthode Cimmino par blocs 222
D.2.2 Méthode Cimmino par blocs augmentée 225
D.3 Implémentation parallèle: ABCD-Solver 230
D.3.1 Schéma de parallélisation 230

D.3.2 Efficacité parallèle 231
D.4 Une approche pour le ABCD-Solver inspirée des méthodes multigrilles . . 234
D.4.1 Problèmes d'EDPs et multigrille 235
D.4.2 Augmentation relâchée et construction de la solution 236
D.4.3 Construction itérative de la solution 237
D.4.3.a Solution part $\overline{A}^{+}b$ 238
D.4.3.b Solution part $W^{+}f$ 239
D.5 Conclusion 241

List of Figures

1 The numerical simulation process involves the solution of sparse linear systems from the discretization of PDEs. 2

I.1 2D discretization of a circle with 6 divisions in the radial direction and 12 divisions in the tangential direction. 11
I.2 Qualitative representation of the evolution for the CPU and Memory performance over the last 48 years. 14
I.3 Common memory hierarchy for modern computing architectures. 14
I.4 Error associated to a linear problem before and after a few iterations of a classical iterative method. The oscillatory error component is damped. . . 20
I.5 The first domain decomposition method (Schwarz's original drawing). . . 23
I.6 hierarchy of 2D nested meshes on a uniform grid. 26
I.7 Illustration of different interpolation operators for nested or non-nested grids. 26
I.8 Evolution of the error in an iteration of the 2-grids cycle, in the case of a Poisson problem. 28
I.9 The coarse grid perceives a wave as more oscillatory on the coarse grid than on the fine grid. 29
I.10 Cycle structures, for various grid levels and number of cycle recursions γ. 30
I.11 *Left* two refined input elements; *Right* ghost layer structure of two input elements. 32
I.12 Showcase with reduction factor $r = 3$ and $m = 2$, for agglomeration techniques. 35
I.13 Comparison of 3 coarse grid solvers in HHG for the three different sizes of problem (I.5): PMINRES (P); MUMPS using BLR and single precision with master-worker agglomeration (M) and Superman agglomeration (S). 44

II.1 2D visualisation of an iteration for *(Blue)* the BC method, and *(Red)* the block Kaczmarz method, applied to an initial iterate $x^{(0)}$. 48

II.2 *(Left)* An iteration of the BC method in 2D. *(Right)* The space has been augmented with one dimension, lines are included into orthogonal planes. The closure equations $W\overline{x} = f$ correspond to a new plane orthogonal to the 2 other ones. One iteration of the BC method gets directly to the solution of the system. 63

II.3 1D grid with 2 domains corresponding to the partitioning of the matrix for block Cimmino. 68

II.4 1D grid with the 2 domains decoupled then re-coupled with compatibility conditions on the interface points. 70

II.5 Convergence of the block Cimmino method applied to the matrix `deltaX` with 16 partitions. The block-CG size varies by power of 2 from 1 to 256. 79

II.6 *(Plain)* Equivalent number of iterations relative to the number of CG iterations. *(Dashed)* Timing for an equivalent iteration relative to the timing of a CG iteration. 80

II.7 *(Left)* Structure of the augmented matrix `deltaX` with a partitioning in 4 blocks of columns, and *(Right)* structure of the resulting Schur complement S. 81

II.8 The BC and ABCD methods applied to `TSOPF_RS_b39_c30` partitioned in 8 blocks of rows or 8 blocks of columns. *(Left)* Convergence of the iterative method. *(Right)* Spectrum of the Schur complement S. 83

III.1 Spectrum of the Schur complement matrix obtained using the augmented block Cimmino method applied to the matrix `bayer01` partitioned in 8 blocks of rows. 91

III.2 Block tridiagonal matrix with a two-blocks partitioning and the corresponding normal equations, sharing also a block tridiagonal structure with respect to the chosen partitioning. Even numbered partitions are not interconnected, and neither are the odd numbered partitions. 94

III.3 A sparse matrix and its hypergraph representation. The white circles represent the nodes, i.e. the row, and the black circles are the nets, i.e. the interconnection columns. 97

III.4 A sparse matrix partitioned with *(Left)* a uniform and *(Right)* a hypergraph partitioner. Grey columns represent interconnections between partitions. . 98

III.5 Normal equations of a matrix partitioned with *(Left)* a uniform and *(Right)* the GRIP partitioner. 99

III.6 Graph illustration of before and after replication of two nodes for a 3-way partitioning. . . . 104

III.7 Distribution of the augmentation size, relative to the size of the original system, for 38 least squares *(green plain line)*, 153 underdetermined *(dashed red line)* and 241 unsymmetric square *(dotted blue line)* problems, sorted by increasing augmentation size. The 5%, 20% and 100% thresholds are drawn. . . . 110

III.8 Two matrices partitioned in 3 blocks of rows, with respectively dense rows and dense columns. The corresponding augmentation blocks obtained with $\mathcal{F}^{FR}$ and $\mathcal{F}^{C_{ij}}$ are displayed. . . . 111

III.9 Hybrid parallelisation scheme of the ABCD-Solver. . . . 114

III.10 3 nodes with 4 processes on each and we have 3 masters with 3 workers each. M_i corresponds to the master i and the S_j of the same colour is its worker j. *(Left)* Compact scheme, *(Right)* Scatter scheme. . . . 118

III.11 Relative difference between the use of the ABCD-Solver and *(Left)* MUMPS or *(Right)* QR-MUMPS in terms of *(x-axis)* execution time and *(y-axis)* memory consumption for the block Cimmino iterative method applied to 187 and 128 classes of problems from the SuiteSparse Matrix Collection. Positive values show a smaller execution time (resp. smaller memory consumption) when using the ABCD-Solver. . . . 125

III.12 Structure of the normal equations after reordering with a SymAMD (Symmetric Approximate Minimum Degree) algorithm for the matrices *(Left)* `LargeRegFile` and *(Right)* `neos`. . . . 129

IV.1 Partitioning of a cube with 16 division in every directions. . . . 133

IV.2 Structured grid after 2 levels of refinement for the *(Left)* square domain $\Omega_{\square}$ and *(Right)* L-shaped domain $\Omega_{\mathbf{L}}$. . . . 135

IV.3 Shape of the solution for *(Left)* the Helmholtz problem (IV.2) *(Right)* the convection-diffusion problem (IV.3). . . . 136

IV.4 *(Left)* Diffusivity and boundary conditions for the Diffusion problem (IV.4) on the L-shaped domain $\Omega_{\mathbf{L}}$. The bottom is a Dirichlet, and the right a Neumann boundary condition, while we have an adiabatic condition on the sides. *(Right)* Shape of the solution. . . . 136

IV.5 Partitioning of the domains based on the geometry: *(Left)* $\Omega_{\square}$ into 9 subdomains and *(Right)* $\Omega_{\mathbf{L}}$ into 12 subdomains. . . . 137

IV.6 Convergence of the BC method applied on the original system *(blue)*, and on the augmented systems for the convection-diffusion problem with grids from 3 and 5 refinements. The augmentations considered are the underdetermined, the generic augmentation with $C = P$, and the overdetermined variants using P the prolongation between the finest grid (size m) and the next coarser grid (size m_c). The prolongator used in the augmentation is *(red)* the original, or *(pink)* the version improved with a Chebyshev filtering. 142

IV.7 Matrix $A \in \mathbb{R}^{1408\times1408}$, prolongation $P \in \mathbb{R}^{1408\times364}$, the product AP and the computed augmentation $C = \mathcal{F}^{A_{ij}}(AP) \in \mathbb{R}^{1408\times248}$ obtained for the heterogeneous diffusion problem (IV.4) with 2 grid levels. 145

IV.8 Non-zero spectrum of the iteration matrix H^{row} for the BC iterations applied on *(Above)* A and *(Below)* $\overline{A}$, obtained for the heterogeneous diffusion problem (IV.4) in $\mathcal{P}_{small}$ with 2 grid levels. 147

IV.9 Convergence of the BC method applied *(plain line, plus marks)* on A (IV.8) and *(dashed line, round marks)* on $\overline{A}$ augmented using the grid level 2 for the 3 PDE problems in $\mathcal{P}_{Large}$ 149

IV.10 Application of the augmentation from the relaxed augmentation method for the problems in $\mathcal{P}_{small}$. Representation of the 2 solution parts, one from the augmented partitions and the other from the closure equations.. 151

IV.11 Error (in log10) from the application of the augmentation from the relaxed augmentation method for the problems in $\mathcal{P}_{small}$ 152

IV.12 Number of iterations *(plain lines, left y-axis)* and size of the augmentation *(dashed lines, right y-axis)* for the BC method applied on $\overline{A}$, obtained using the different coarse grid levels for the test set $\mathcal{P}_{large}$. The augmentation sizes for Helmholtz and convection-diffusion completely overlap 152

IV.13 Number of iterations *(plain lines, left y-axis)*, and augmentation size *(dashed lines, right y-axis)* for the BC method applied on $\overline{A}$, using different number of partitions: $3ndiv^2$ for the Diffusion (IV.4), and $ndiv^2$ for Helmholtz (IV.2) and convection-diffusion problems (IV.3). The augmentation sizes for Helmholtz and Convection-Diffusion completely overlap. . 153

IV.14 Spectrum of the Schur complement S obtained with *(Above)* the ABCD method, and *(Below)* the C-ABCD method for the heterogeneous diffusion problem in $\mathcal{P}_{small}$ 153

IV.15 Evolution of the average number of iterations to approximate a part of W of size s relative to the average number of iterations with a block size of 1. The results for Diffusion and Convection-Diffusion completely overlap. . . 159

IV.16 For the problems in $\mathcal{P}_{large}$, we show the evolution of *(dashed line with left y-axis)* the average number of iterations for BC-W and *(plain line with right y-axis)* the number of iterations for the global BC depending on the stopping criterion ϵ chosen for the approximation of W. 161

IV.17 Convergence of BC applied on the large Helmholtz problem with a threshold ϵ for BC-W varying between 10^{-12} and 10^{-4}. The convergence for $\epsilon = 10^{-12}$, $\epsilon = 10^{-10}$, and $\epsilon = 10^{-8}$ completely overlaps. 161

A.1 Relative difference between row and column partitioning for *(x-axis)* the number of iterations, and *(y-axis)* the size of the augmentation. These results are obtained for 158 and 238 classes of problems, resp. for BC and ABCD, from matrices extracted from the SuiteSparse Matrix Collection. . 195

B.1 Relative difference between the GRIP and the PaToH partitioners in terms of (x-axis) iterations and (y-axis) total time in seconds for the block Cimmino iterative method applied on 325 classes of problems from the SuiteSparse Matrix Collection. Positive values show a smaller number of iterations (resp. smaller execution time) when using the PaToH partitioner. 198

B.2 Two matrices with respectively dense rows and dense columns, and partitioned in 3 blocks of rows. The corresponding augmentation blocks obtained with $\mathcal{F}^{FR}$ and $\mathcal{F}^{C_{ij}}$ are displayed. 199

B.3 Sparse structure of the test matrices. 201

C.1 Illustration of *(Left)* classical and *(Right)* additive Multigrid V-cycle iterations in terms of successive projections on non-orthogonal subspaces: ($\mathcal{L}$,$\mathcal{H}$) corresponds to the low/high-frequency modes subspaces and ($\mathcal{N}$,$\mathcal{R}$) corresponds to $\mathcal{N}(RA)$, with $R = P^T A$, and $\mathcal{R}(P)$. 207

C.2 Hybrid parallelisation scheme of the ABCD-Solver with the approximation of W and its inclusion as a partition in the global block-CG. 209

D.1 The numerical simulation process involves the solution of sparse linear systems from the discretization of PDEs. 212

D.2 In the PhD, we explore the world of hybrid methods for the solution of sparse linear systems. 213

D.3 Coupe transversale du maillage initial avec 6 divisions radiales et 12 divisions tangentielles. 214

D.4 Scalabilité faible pour un problème similaire à (D.1). Nous distinguons le temps moyen pour le traitement des grilles fine et de la grille la plus grossière. L'efficacité parallèle est calculée par rapport au temps moyen du plus petit problème. 216
D.5 Scalabilité faible pour le problème (D.1) avec viscosité (Gauche) *iso-viscous* et (Droite) *jump-410*. Nous distinguons le temps moyen pour le traitement des grilles fine et de la grille la plus grossière. L'efficacité parallèle est calculée par rapport au temps moyen du plus petit problème. 216
D.6 Source: thèse de Theo Mary [100] 218
D.7 *(Gauche)* Les niveaux de grille grossiers ont de moins en moins de DDL par coeur, *(Droite)* ce qui entraîne une baisse de la granularité des sous-problèmes résolus dans MUMPS. 218
D.8 Schémas d'agglomération proposés. 219
D.9 Temps d'exécution de MUMPS pour les 3 tailles de problème selon le facteur de réduction utilisé pour l'agglomération. 220
D.10 Comparaison de 3 solveurs sur la grille grossière dans HHG pour les 3 tailles du problème (D.1): PMINRES (P); MUMPS utilisant BLR et simple précision couplé avec agglomération Maître-Ouvrier (M) ou agglomération Superman (S). 221
D.11 Itération en 2D de la méthode de projection par ligne Cimmino. *Source:* [133] . 222
D.12 Convergence de BCG: `deltaX` ($7 \cdot 10^4 \times 2 \cdot 10^4$) avec 16 partitions 225
D.13 Augmentation d'une matrice tri-diagonale par bloc. 226
D.14 Cimmino par bloc vs Cimmino par bloc augmenté 228
D.15 Distribution de la taille d'augmentation, relatif à la taille du système original, pour 38 problèmes sur-déterminés *(points verts)*, 153 problèmes sous-déterminés *(croix rouges)* et 241 problèmes carrés non-symétriques *(signes plus bleus)* problèmes, classés par taille d'augmentation croissante. Les seuils de 5%, 20% et 100% sont indiqués. 229
D.16 ABCD-Solver: Schéma de parallélisation. 231
D.17 Différence relative entre l'ABCD-Solver et *(Gauche)* MUMPS ou *(Droite)* QR-MUMPS en terme *(x-axis)* de temps d'exécution et *(y-axis)* de mémoire pour 187 et 128 classes de problèmes de la SuiteSparse Matrix Collection. Une valeur positive indique un avantage pour l'ABCD-Solver (temps ou mémoire plus petit). 232
D.18 Différence relative entre l'ABCD-Solver et QR-MUMPS en terme *(x-axis)* de temps d'exécution et *(y-axis)* de mémoire. Les plus grosses classes de problème (temps $>$ 1s) pour lesquelles chaque solveur est meilleur sont indiquées, ainsi que 3 exemples spécifiques. 233

D.19 Structure des équations normales après reordering grâce à l'algorithme SymAMD (Symmetric Approximate Minimum Degree). 234
D.20 Maillage et partitionnement des domaines carré et en L. 235
D.21 Solution des problèmes d'EDPs. 236
D.22 Itérations de bloc Cimmino avec et sans augmentation, en utilisant le niveau $l = 2$ (4 niveaux) 239
D.23 Block Cimmino appliqué sur $\overline{A}$ avec différents choix de niveau grossier. . . 239

List of Tables

I.1 Total run-times (in seconds) of the V_{var} application: total, fine and coarse grid timings for the asthenosphere scenarios iso-viscous and jump-410. The number of iterations of the MG method (it) and the average number of iterations of the coarse grid solver ($C.it$) are also displayed. 34

I.2 Scaling of the sparse direct solver MUMPS (with exact double precision arithmetic): analysis, factorisation and solve run-times in seconds. r: reduction factor, m: corresponding number of processes. 39

I.3 Influence of the viscosity scenario and BLR ϵ parameter, with double and single precision, on the accuracy and the run-time of the direct solver. Run-times (in seconds) are separated in analysis, factorisation and solve steps. Each process runs on a separate node. 41

I.4 Weak scaling of the V_{var}-cycle with a sparse direct and a block low-rank coarse level solver with or without single precision (SP) arithmetic. The parallel efficiency compares the average total run-time of each run to the average total run-time of the smallest case with no BLR. 42

I.5 Weak scaling of the multigrid execution with a sparse direct and a block low-rank coarse level solver with single precision (SP) arithmetic. The total execution time (in seconds) and the parallel efficiency considering the average total execution times are displayed for both master-worker and a simulated Superman agglomerations. 43

II.1 Characteristics of the test matrices. m and n: the dimensions of the matrix, "elts per row": the number of nonzero values in the matrix. . . . 77

II.2 Sequential execution time in seconds of the block-CG algorithm for rectangular matrices, with an increasing block-CG size in power of 2 starting from 1 to 256. 79

II.3 Application of the ABCD method on the test matrices. Displayed are the size/density of S, and the memory required for its factorisation. Also, we give for BC and ABCD the memory requirement for the factorisation of the projection systems, as well as the execution time and accuracy. 82

II.4 For each method (BC or ABCD), number of classes of problems where the partitioning giving best result is row, column, or both. The choice is decided in terms of smallest number of iterations or augmentation size. . . 84

III.1 Characteristics of the test matrices. n: the order of the matrix, nnz: the number of nonzero values in the matrix. 90

III.2 Effect of a scaling applied to the matrix before the application of the augmented block Cimmino method. The conditioning of the Schur complement S and the average conditioning of the projection systems are given. 91

III.3 Best partitioner in terms of iterations for the BC method and in terms of augmentation size for the ABCD method applied to resp. 361 (BC) and 429 (ABCD) classes of problems from the SuiteSparse Matrix Collection. The matrices are split in least-squares problems, and underdetermined or square matrices. 100

III.4 Characteristics of the test matrices. n: the order of the matrix, nnz: the number of nonzero values in the matrix. 104

III.5 Impact of the Duplication Method on the number of iterations, relatively to the disjoint partitioning case. The number of duplications is bounded by a fixed percentage of the matrix size. 105

III.6 Best augmentation method in terms of size for different classes of problems from the SuiteSparse Matrix Collection: 36 Least-Squares, 161 underdetermined and 241 unsymmetric square matrices. 111

III.7 Characteristics of the test matrices. n: the order of the matrix, nnz: the number of nonzero values in the matrix. 119

III.8 Impact of the distribution of partitions on the execution times. All runs were performed with 1024 partitions and 128 MPI processes with 2 threads on 16 nodes. (Com. col %: Normalised column reduction values with respect to the Greedy algorithm. tot: Total time in seconds. Fact. imb. %: ratio of maximum over average factorisation times. BCG it: Number of iterations required for convergence. Sol. time: Total solution time in seconds) 121

III.9 Execution time for the solution of the Schur complement system depending on the number of processes used. The number of processes is displayed relatively to the number of partitions used for each matrix. 122

III.10 Impact of the placement of masters and workers on the execution times of the ABCD Solver. All runs were performed with 32 partitions and 128 MPI processes with 2 threads on 16 nodes. 123

III.11 Best solver in terms of execution time or memory for different classes of problems from the SuiteSparse Matrix Collection. We distinguish the comparison in distributed memory MUMPS vs ABCD-Solver (187 classes), and the comparison in shared-memory QR-MUMPS vs ABCD-Solver (128 classes). 126

III.12 Execution times for the different steps involved in BC, ABCD and QR-MUMPS on the problem `neos` partitioned into 64 partitions. The backward error ω on the normal equations for the computed solution is given. 127

III.13 Execution time for the different steps involved in block Cimmino, and `QR-MUMPS` on the problem `LargeRegFile` partitioned into 128 partitions. The backward error ω^{LS} on the normal equations for the computed solution is also given. 128

III.14 Execution time for the different steps involved in BC, ABCD and QR-MUMPS on the problem shar_te2-b2 partitioned into 64 partitions. The backward error ω^{LS} for the computed solution is also given. 129

IV.1 Size of the matrix depending on the grid level for the different test sets and problems. 138

IV.2 Size of the prolongation and the augmentation depending on the grid level chosen for the 3 PDE problems in test set $\mathcal{P}_{Large}$. 146

IV.3 Evolution of the number of iterations for the convergence of the global BC method applied on the system augmented on the coarse grid level 2, depending on the block-CG size for BC-W. 160

IV.4 Conditioning of S obtained with C-ABCD applied on the large problems in the set $\mathcal{P}_{large}$, with varying ϵ threshold for BC-W between 10^{-12} and 10^{-4}. 162

IV.5 Conditioning of the Schur complement before and after preconditioning with various methods. 164

A.1 Application of the block Cimmino accelerated method on rectangular matrices with an increasing block size for the block-CG algorithm in power of 2 from 1 to 256. The first table displays the number of iterations for convergence and the second the total time in seconds for the block-CG iterations. 193

B.1 Characteristics of the test matrices. n: the order of the matrix, nnz: the number of nonzero values in the matrix. 200
B.2 Application of ABCD with the augmentation methods $\mathcal{F}^{C_{ij}}$ and $\mathcal{F}^{FR}$. The size and number of elements per row of the matrix S are given as well as its condition number, and the sequential execution time to compute the solution. 200

D.1 Taille des problèmes fins obtenus à partir de différents maillages initiaux, avec le nombre de tétrahèdres et la résolution correspondantes. La taille du problème sur la grille grossière est également donnée. 214
D.2 MUMPS on the largest problem $(1.94 \cdot 10^6 \ DOFs)$ with $jump - 410$. FYI: PMINRES runtime is 165.5s 217
D.3 MUMPS sur le problème le plus gros $(1.94 \cdot 10^6 \ DOFs)$ with $jump - 410$. FYI: PMINRES runtime is 165.5s 220
D.4 Size of the prolongator P and augmentation $\mathcal{F}(AP)$ depending on level . 237

Summary

The comprehension of the physics behind complex physical problems is the goal of numerical simulation. The models behind these simulations are often based on Partial Differential Equations (PDE) coupled with some boundary conditions [111]. From these equations, discretization techniques like the Finite element methods are applied on a mesh, which introduces large sparse linear systems of the form $Ax = b$ [53, 64]. The solution of these systems is of major interest in high performance computing. Great efforts were spent in the last decades in order to design solution methods for these linear systems, called solvers, executed in parallel on supercomputing facilities.

Historically, two kinds of solvers were developed [128]. The direct methods are based on the elimination of unknowns, leading to a factorisation of the matrix, to compute the solution [46, 61]. The iterative methods improve an approximation in a succession of updates until the convergence to a desired accuracy has been reached [75, 114]. While direct methods are robust with respect to the problem solved, their cost in terms of computation and memory is high compared to the iterative methods, for which the convergence is very dependent on the problem. Lastly introduced, hybrid methods were built in order to benefit from the advantages of both direct and iterative methods.

Multigrid methods

We first focus on a kind of hybrid methods called the multigrid methods [27, 129]. The application of a discretization method on a PDE problem can be performed on grids with various levels of refinement, creating linear systems of according size. Using a hierarchy of such refinements, the multigrid methods are motivated by two observations. First, standard iterative methods are able to damp rapidly oscillatory error components, while keeping smooth components almost untouched, thus called smoothers. Secondly, smooth functions, which are well represented on a coarse grid level, can then appear more oscillatory on such level, and smoothers are then efficient again. In a global iterative process, these two aspects are used to compute the solution of a linear system by going through each level in cycles. On the coarsest level of grids, the system is generally considered small enough to allow the application of a direct solver. Making multigrid methods scale on large computing systems is difficult [109, 110]. Using multigrid methods on supercomputers mostly comes with a large number of grid refinements and then the ratio of grid points from the finest to the coarsest level becomes very large. When moving to the coarsest levels, the computing efficiency deteriorates if all computing cores are implied. In Chapter I, we introduce an agglomeration of the coarse grid problem onto

fewer processors in order to compensate for this effect [101]. The convergence of multigrid methods is theoretically proven only when a direct solver is applied on the coarsest grid [25, 73]. For systems where the model and/or the data present some incertitude, a popular alternative is to only approximate the coarse grid solution using an iterative method with an appropriately chosen accuracy. In cases where the iterative solver has trouble converging, the use of a direct solver combined with an approximated factorisation is ideal, thanks to its robustness and relaxed computation and memory requirements. In Chapter I, we study the solution, using the multigrid HHG framework (Hierarchical Hybrid Grids) [18, 85], of a saddle-point problem with jumping coefficients up to 10^11 degrees of freedom, inspired from Earth's mantle convection [14]. We demonstrate an improvement of the overall multigrid scheme with the use of MUMPS[1] [6], combined with a block low-rank approximation [1, 4] and the use of single precision arithmetic, on the agglomerated coarse grid problem, compared to the use of an iterative method on the coarse grid [30]. This is an essential improvement e.g. for Earth Mantle simulation scenarios [16].

Block Cimmino iterative and pseudo-direct methods

The other major type of hybrid methods is the Domain Decomposition Methods (DDM) [40, 98, 127] which, in order to make the best use of parallel computing platforms, decompose the complete linear system into subproblems to be solved independently. An instance of a direct solver is then executed in parallel simultaneously on each subproblem, and a global iterative method combines the obtained parts to ensure the consistency of the global solution. We are interested in a particular class of iterative methods which can be interpreted as DDM [40]: the block projection methods [34, 52, 88]. Based on a partitioning of the matrix in blocks of rows or columns, these methods compute the solution through successive projections on the subspaces spanned by the partitions. Among the projection methods, the block Cimmino iterations are of special interest since at each iteration, the projections are computed independently. The convergence of the block Cimmino iterations is known to be slow. An acceleration of the iterations is possible through the use of a stabilised block conjugate gradient (block-CG) algorithm [11, 112]. The accelerated block Cimmino (BC) takes benefit from a combination of iteration reduction, and the use of computationally efficient matrix operations [80]. As the convergence of this method remains problem dependent, an alternative was proposed based on the augmentation of the original system with additional variables and constraints in order to orthogonalize the partitions. Block Cimmino is guaranteed to converge in

[1] `http://mumps-solver.org/`

one iteration on this augmented system, and we obtain the pseudo-direct Augmented Block Cimmino method (ABCD) [47, 133]. The additional variables and constraints are acting on the interconnections between partitions. Geometrically, and using the language of DDM, the augmentation introduces a splitting on the interfaces between subdomains. Then, the central point of ABCD is the embarrassingly parallel construction of a condensed Schur complement matrix. The solution of this relatively smaller matrix, using a direct solver, can require a large amount of memory, and in such case the block Cimmino iterations stay a good option. These two approaches were developed using a row partitioning for the solution of unsymmetric square systems. In Chapter II, we extend the iterative and augmented methods to full rank systems with the minimum-norm solution of underdetermined systems, and the solution of least-squares problems [50]. The latter is based on a column partitioning of the matrix. In the case of unsymmetric square systems, the choice is then open between row and column partitioning which makes a large difference, in terms of convergence speed or augmentation size, for a wide range of applications.

Parallel implementation of the ABCD-Solver

In order to improve the numerical properties of the block Cimmino iterative and augmented methods, preprocessing techniques are applied, introduced in the first part of Chapter III. After scaling the system by a simultaneous normalisation of both rows and columns [113], the algebraic partitioning methods used are discussed [42, 112]. We emphasise the use of graph-based partitioners which specifically aim either at the acceleration of the convergence of BC, with the numerically-aware partitioner GRIP [126], or at the minimisation of the size of the interfaces corresponding to the augmentation size in ABCD, with the hypergraph partitioner PaToH [32]. A new augmentation method additionally allows to decrease the augmentation size in ABCD, and thus the size of the Schur complement.

The iterative method BC and the augmented method ABCD are implemented in parallel in the ABCD-Solver[2] based on MPI processes [71] (distributed) and OpenMP threads [37] (shared-memory) parallelism. The solver follows a classical hybrid parallelisation scheme [133]. Specific processes, called masters, receive one or several partitions and compute independently in parallel the associated projection using a direct solver. In the iterative scheme, the masters sum their projections in parallel, using MPI communications, to update the current approximation. We propose a new algorithm to simultaneously minimise the future communication between masters while keeping a global balance in

[2] `http://abcd.enseeiht.fr/`

workload. Two additional levels of parallelism are associated with the direct solver. The computationally intensive dense kernels used internally are parallelised in shared-memory. Additionally, processes with no partitions, called workers, can be linked to a master to execute the direct solver in parallel. We show that deciding which process is a master or a worker depending on their place in the parallel architecture at runtime has an important impact. In particular, spreading the masters over the nodes greatly increases the efficiency of cache access in the direct solver and the dense kernels, inducing a good speed-up for the ABCD-Solver.

We demonstrate a good behaviour of the ABCD-Solver, in terms of execution time and memory, compared to the state-of-the-art direct solvers MUMPS[3] [6], for square matrices, and QR-MUMPS[4] [29], for rectangular matrices.

Multigrid inspired relaxation of the augmented block Cimmino method

While the iterative method has low memory requirements, its convergence is problem dependent. The augmented method, on the contrary, converges in 1 iteration but is highly dependent on the solution of the Schur complement which solution can require prohibitive amounts of memory. In the context of discretized PDE problems [53], it is possible to choose the partitioning based on the geometry of the domain, then the size of the Schur is linked to the size of interfaces between subdomains. From multigrid methods, we use the idea of a hierarchy of grids. A natural way to decrease the size of the Schur is then to augment the matrix based on a coarser representation of the interfaces between subdomains. Through the choice of a specific coarse grid level, we can control the size of the augmentation. In Chapter IV, we focus on the solution of square matrices using a partitioning in blocks of rows. We demonstrate that the iterative block Cimmino applied on the matrix augmented this way, has a fast linear convergence for a wide range of systems arising from the discretisation of 2D PDE problems. The central issue of this approach is the construction and solution of the additional constraints in the system and thus the Schur complement matrix. We introduce a construction method based on the application of BC with linear convergence on canonical vectors. The right hand side corresponding to the additional constraints is simultaneously updated during this construction. The downside of this method is that the obtained constraints are very dense. Finally, we present some research tracks, as perspective, which consists in finding a good preconditioner for the Schur complement, without constructing the actual matrix. This preconditioner would then be used in global preconditioned-CG [75] iterations to

[3] `http://mumps-solver.org/`
[4] `http://buttari.perso.enseeiht.fr/qr_mumps/`

approximate the inverse of the Schur, using internal projections computed with the BC method at each iteration.

Perspective and future works

Overlapping partitioning

Considering the block Cimmino methods as DDM, the natural evolution is to introduce the notion of overlaps between subdomains [40]. In our context, overlaps correspond to non-disjoint partitions for BC. We introduce in Section III.1.2.e, a method which iteratively improves an existing partitioning by duplicating some rows into other partitions. The choice of duplicated rows is based on the values of the normal equations inspired from the GRIP partitioner [126]. The replication of rows allows an efficient acceleration of the convergence of BC for some systems, but for others the added interconnections between previously unlinked partitions through replication is counter-productive. One direction of research is to find an algorithm which take these added links into account to guarantee an acceleration of convergence.

Possible extensions to the relaxed augmentation

The multigrid-inspired relaxed ABCD has been developed in this thesis for a row partitioning in the context of discretised PDE problems. The next step is to extend this method to a column partitioning. The main difficulty is the careful construction of the block-CG algorithm for the simultaneous approximation of the constraint equations and the corresponding right hand side. Additionally, in order to make the method applicable to problems not coming from PDE problems, the use of algebraic multigrid techniques can be considered in order to construct the necessary multigrid components directly from the entries in the matrix.

We introduce at the end of Chapter IV some possibilities in order to improve the approach based on the relaxed augmentation. A first track is the improvement of the very dense approximated closure equations W, either through a filtering of their entries, or through a different approximation, e.g. using Chebyshev polynomials. The parameters for these iterations are the maximum and minimum eigenvalues of the BC iteration matrix estimated thanks to the linear convergence rate of the associated BC method in the block-CG. The second track concerns a more efficient projection on W using a block Kaczmarz type of update every few iterations of the BC method on the augmented partitions. Finally, the linear convergence of our new approach could not be formally proven...

Formal proofs of linear convergence

...which would be the real evolution of this thesis. For specific PDE problems, or using algebraic properties from the multigrid elements such as the approximation property of the prolongation operator, it would be possible to show some properties on the convergence of our new approach. A particularly interesting possibility is to interpret this method as a 2-level DDM and find the direct relation to existing methods like BDDC and FETI-DP [90].

Résumé

L'intérêt de la simulation numérique est de mieux comprendre la physique derrière des systèmes physiques complexes. Ces simulations sont souvent basées sur des modèles d'équations aux dérivées partielles (EDP) couplées avec des conditions au bord [111]. Des méthodes de discrétisation, comme la méthode des éléments finis, appliquées à ces équations donnent de grands systèmes linéaires creux de la forme $Ax = b$ à résoudre [53, 64]. Leur solution est l'un des thèmes majeurs en calcul scientifique haute performance. Ces dernières décennies, des recherches poussées on été effectuées en vue de créer des méthodes pour la résolution de ces systèmes linéaires, appelés solveurs, pouvant être exécutées efficacement en parallèle sur des supercalculateurs.

Historiquement, nous distinguons 2 classes de solveurs [128]. Les méthodes directes sont basées sur des techniques d'élimination de variables, menant à une factorisation de la matrice A, afin de résoudre le système linéaire [46, 61]. Les méthodes itératives quant à elles font successivement évoluer une approximation de la solution jusqu'à obtenir la précision désirée [75, 114]. Là où les méthodes directes se montrent robustes par rapport aux propriétés numériques du problème à résoudre, leur coût en calcul et mémoire est élevé comparé à celui des méthodes itératives, qui elles présentent une convergence très variable selon le problème. Introduites plus récemment, les méthodes hybrides ont été crées afin de bénéficier des avantages des 2 types de méthodes, directes et itératives.

Méthodes multigrilles

Nous nous intéressons d'abord à des méthodes hybrides appelées méthodes multigrilles[27, 129]. Il est possible d'appliquer les méthodes de discrétisation sur des niveaux de grilles de raffinement varié, créant des systèmes de taille plus ou moins grande. En utilisant une telle hiérarchie de grilles, les méthodes multigrilles obtiennent la solution des systèmes linéaires grâce à deux principes. Tout d'abord, il a été observé que certaines classes de méthodes itératives, appelées lisseurs, atténuent rapidement les composantes d'erreur caractérisées par une haute fréquence d'oscillation, alors que les composantes à basse fréquence, i.e. lisses, restes pratiquement inchangées. De plus, les fonctions lisses sont bien représentées sur une grille grossière, où elles apparaissent comparativement plus oscillatoire. Ainsi, l'application d'une méthode itérative sur ces fonctions au niveau de la grille grossière devient à nouveau efficace. A l'intérieur d'un processus itératif global, les méthodes multigrilles utilisent ces deux aspects afin de calculer la solution du système linéaire en se déplaçant d'un niveau de grille à l'autre dans un cycle. Au niveau de la grille la plus grossière, le système est généralement considéré comme assez petit pour

pouvoir être résolu avec un solveur direct sans utiliser trop de mémoire. Implémenter une méthode multigrille de manière à ce qu'elle se comporte bien sur des systèmes de calculs très larges est difficile[109, 110]. Dans ce cas, en partant d'une grille très fine, résolue avec beaucoup d'unités de calcul (coeurs), et en se déplaçant vers une grille plus grossière, le nombre de variables par coeur devient de plus en plus petit. L'efficacité en calcul se détériore alors rapidement si tous les coeurs sont utilisés à tous les niveau. Dans le Chapitre I, afin de compenser cette effet[101], nous proposons d'utiliser une méthode d'agglomération de données sur un sous-ensemble des coeurs lorsque l'on considère le problème le plus grossier[25, 73]. De plus, la convergence des méthodes multigrilles est en théorie prouvée seulement si le problème le plus grossier est résolue de manière exacte, avec un solveur direct. Cependant, pour des systèmes comportant une incertitude sur le modèle et/ou les données, il est possible d'approcher la solution à moindre coût avec une méthode itérative avec une précision correspondant au problème. Dans des cas où la méthode itérative converge lentement, l'utilisation d'un solveur direct avec une factorisation approchée peut alors devenir idéale, grâce à la robustesse de la méthode et à un relâchement des besoins en calcul et en mémoire. Dans le Chapitre I, nous étudions l'utilisation du framework multigrille HHG (Hierarchical Hybrid Grids)[18, 85]. pour la solution d'un problème de point de selle avec des coefficient très variables d'une taille allant jusqu'à 10^{11} variables, et inspiré de la simulation de la convection du manteau terrestre [14]. Nous démontrons pour ce problème une amélioration de l'efficacité parallèle du schéma multigrille grâce à l'utilisation combinée du solveur direct MUMPS[5] [6], avec approximation par blocs de rangs faibles [1, 4] et l'utilisation de l'arithmétique simple précision, comparée à l'utilisation d'une méthode itérative classique sur la grille la plus grossière [30]. Cette approche est une contribution pour la simulation numérique et, dans notre cas, pour la simulation du manteau terrestre [16].

Méthodes Cimmino par blocs, itérative et pseudo-directe

L'autre grande classe de méthodes hybrides sont les méthodes de décomposition de domaine (DDM) [40, 98, 127] qui décomposent le problème en sous-problèmes résolus indépendamment, afin d'être efficaces en terme de calcul parallèle. Une instance de solveur direct est alors exécutée en parallèle simultanément sur chaque sous-problème, et un schéma itératif global combine les différentes solutions obtenues afin de garantir la cohérence de la solution globale. Nous nous intéressons plus particulièrement à une classe de méthodes itératives, pouvant être interprétée en tant que méthodes de décomposition de domaine[40]: les méthodes de projection par bloc [34, 52, 88]. A partir

[5] `http://mumps-solver.org/`

d'un partitionnement de la matrice par blocs, ces méthodes se rapprochent de la solution du systèmes par projections successives sur les espaces engendrés par les partitions. Parmi ces méthodes de projection, la méthode Cimmino par blocs est d'un intérêt particulier puisqu'à chaque itération les projections peuvent être calculées indépendamment. Il est cependant connu que la convergence de cette méthode peut être lente. Il est alors possible de l'accélérer en utilisant un gradient conjugué travaillant par blocs stabilisés (bloc-CG)[11, 112]. Nous obtenons ainsi la méthode Cimmino par blocs accélérée (BC), qui tire avantage d'une réduction du nombre d'itérations, et de l'utilisation de noyau efficaces pour le calcul d'opérations sur des matrices denses[80]. Comme la convergence de cette reste méthode dépendante du problème considéré, une alternative basée sur l'augmentation du système original avec des variables et contraintes additionnelles a été proposée. Cette augmentation permet d'orthogonaliser mutuellement les partitions, garantissant ainsi la convergence en une seule itération de la méthode BC. Ainsi, une méthode pseudo-directe est obtenue, la méthode Cimmino par blocs augmentés (ABCD)[47, 133]. Les variables et contraintes additionnelles agissent sur les interconnexions entre partitions et introduisent une cassure au niveau des interfaces entre sous-domaines, d'où leur lien avec les DDM. Enfin, le point clé de la méthode ABCD est la construction de manière "embarrassingly parallel" d'un complément de Schur. La solution de cette matrice, relativement plus petite mais plus dense, est effectuée avec un solveur direct qui peut alors nécessiter une large quantité de mémoire et de calcul. Dans ces cas-là, la méthode BC itérative reste une bonne option, car peu coûteuse en mémoire. A l'origine, ces deux approches ont été développées pour la solution de problèmes carrés et non-symétriques à l'aide d'un partitionnement de la matrice en blocs de lignes. Dans le Chapter II, nous développons une extension des méthodes itératives et pseudo-directes pour la solution de problèmes sous-déterminés de norme minimale et la solution de problèmes de moindres-carrés[50]. Ces derniers sont basés sur un partitionnement par blocs de colonnes. Dans le cas des systèmes carrés, le choix est alors ouvert entre l'utilisation d'un partitionnement par ligne ou par colonne, l'un ou l'autre pouvant présenter un plus grand avantage selon le problème, en terme d'itérations ou de taille d'augmentation.

Implémentation parallèle: ABCD-Solver

Afin d'améliorer les propriété numériques des méthodes itératives et pseudo-directes, des techniques de pré-traitement sont présentées dans la première partie du Chapter III. Après avoir atténué la disparité dans les facteurs d'échelle de la matrice via une normalisation simultanée des lignes et des colonnes dans le cas des matrices carrées [113], les différentes méthodes de partitionnement existantes sont discutées [42, 112]. Nous nous intéressons en

particulier à des méthodes de partitionnement de graphes qui cherchent, soit à accélérer la convergence de la méthode BC, avec le partitionneur GRIP [126] prenant en compte les valeurs dans les équations normales, ou à diminuer la taille de l'augmentation, avec le partitionneur d'hyper-graphes PaToH PaToH [32]. Nous introduisons ensuite une nouvelle technique d'augmentation de la matrice qui permet de diminuer encore la taille d'augmentation.

Les méthodes BC et ABCD sont implémentées en parallèle dans le ABCD-Solver, basé sur l'utilisation de processus MPI [71] (distribué) et de threads OpenMP [37] (mémoire partagée). Le solveur suit un schéma de parallélisation classique en DDM: des processus MPI, appelés maîtres, reçoivent une ou plusieurs partitions et calculent en parallèle, de manière indépendante, les projections associées à l'aide d'un solveur direct. Ces projections locales sont alors sommées grâce à des communications MPI entre maîtres afin de mettre l'itéré courant à jour. Nous proposons un nouvel algorithme de distribution des partitions pour la méthode itérative qui permet, en plus d'équilibrer la charge de travail entre les processus, de diminuer la communication entre ceux-ci afin de diminuer le temps de calcul. En plus de l'indépendance entre les partitions, 2 niveaux de parallélisme sont ajoutés par le solveur direct utilisé pour les projections. Les noyau de calcul pour les opérations denses utilisés par le solveur direct ont une parallélisation en mémoire-partagée. De plus, chaque potentiel processus sans partition, appelé ouvrier, peut être assignée à un maître afin de paralléliser l'exécution du solveur direct. Nous montrons que distribuer les maîtres sur l'architecture de calcul, plutôt que de les grouper, permet de diminuer les accès mémoire concurrents et ainsi de réduire le temps d'exécution.

Enfin, nous effectuons une comparaison entre le solveur ABCD-Solver et les solveurs directs MUMPS[6] [6], pour les matrices carrées, et QR-MUMPS [29] pour les matrices rectangulaires.

Relaxation inspirée du multigrille de l'augmentation pour la méthode Cimmino par blocs

Bien que la méthode BC nécessite peu de mémoire, sa convergence dépend du problème. La méthode ABCD, au contraire, converge en une itération mais est très dépendante de la solution du complément de Schur qui peut nécessiter trop de mémoire. Pour la solution de problèmes d'EDP discrétisés, il est possible de choisir un partitionnement basé sur la géométrie des domaines. Dans ce cas, la taille du Schur est liée à la taille des interfaces entre sous-domaines. En utilisant une hiérarchie de grille, directement héritée des méthodes multigrilles, une manière naturelle de diminuer la taille du Schur est

[6] `http://mumps-solver.org/`

d'augmenter la matrice sur une représentation grossière des interfaces entre sous-domaines. Au travers du choix d'une grille plus ou moins grossière, nous contrôlons alors la taille de l'augmentation. Dans le Chapter IV, nous nous concentrons sur des matrices carrées et partitionnées par blocs de lignes. Nous montrons que la méthode BC, appliquée sur les partitions augmentées de cette manière, a une convergence linéaire et rapide pour des problèmes provenant de la discrétisation de problème d'EDP en 2D. Toute la problématique de cette approche est alors la construction des contraintes additionnelles, et la solution du complément de Schur qui en découle. Nous proposons une méthode de construction basée sur l'application d'itérations BC avec une convergence linéaire sur des vecteurs canoniques. Le second membre doit alors correspondre à ces contraintes approchées, et nous le construisons en même temps. L'inconvénient de cette méthode est que nous obtenons une matrice de contraintes dense. Nous ouvrons finalement sur quelques pistes consistant à trouver un bon pré-conditionneur du complément de Schur, sans construction explicite de celui-ci, qui permettra d'approcher son inverse grâce à un algorithme CG préconditionné [75], qui utilise à chaque itération des projections basées sur la méthode BC.

Perspective et futures travaux

Partitionnement avec chevauchement

Partant de l'interprétation de la méthode BC comme méthode de décomposition de domaine, une idée naturelle est d'introduire la notion de chevauchement entre sous-domaines [40]. Dans notre contexte, le chevauchement correspond à la construction de partitions avec des lignes pouvant appartenir à plusieurs partitions. Dans le Chapitre III.1.2.e, nous proposons une méthode construisant itérativement un tel partitionnement, à partir d'un partitionnement classique existant, en dupliquant des lignes d'une partition à l'autre. Le choix pour ces duplications se base sur les valeurs contenues dans les équations normales du système, de manière similaire au partitionneur GRIP [126]. La réplication de lignes permet une accélération de la convergence de la méthode BC pour certaines matrices, bien que pour d'autres les interconnections entre partitions ajoutées par ces réplications est contre-productive. Nous cherchons à développer une méthode qui permettrait de prendre en compte ces nouvelles interconnections afin de garantir que la convergence ne soit pas ralentie via la réplication de ligne.

Améliorations possibles de l'approche C-ABCD

L'approche avec augmentation relâchée sur une grille grossière a été développée dans cette thèse pour les matrices carrées avec un partitionnement par ligne. Il serait maintenant naturel d'étendre cette approche à l'utilisation d'un partitionnement par colonnes. La difficulté principales est alors la construction d'un algorithme CG par blocs qui calcule simultanément une approximation des contraintes additionnelles et du second membre. De plus, il serait possible d'étendre l'approche avec augmentation relâchée à des problèmes non dérivés d'EDP, et possiblement rectangulaires, en construisant la hiérarchie de grilles via des techniques provenant des méthodes multigrilles algébriques.

En ouverture du Chapitre IV, différentes possibilités sont introduites pour améliorer l'approche C-ABCD. Une première piste serait d'améliorer l'approximation (très dense) des équations additionnelles, d'une part via un filtrage de ses entrées, et d'autre part en calculant une approximation différente, e.g. via des polynômes de Chebyshev. Les paramètres pour les itérations de Chebyshev sont les valeurs propres de la matrice d'itération de BC estimées grâce à quelques itérations de la méthode BC avec convergence linéaire. Une deuxième piste concerne le calcul efficace de la projection sur le bloc W grâce à une itération de la méthode Kaczmarz par blocs, appliquée toute les quelques itérations de la méthode BC. Enfin, nous n'avons pu démontrer formellement sous quelles conditions la convergence linéaire sur les partitions augmentée est garantie...

Preuve formelle d'une convergence linéaire

...ce qui serait une vraie évolution de cette thèse. Pour des problèmes d'EDP spécifiques, et en utilisant des propriétés algébriques telles que la propriété d'approximation de l'opérateur de prolongation, il serait peut-être possible de démonter la convergence de notre nouvelle approche. Il serait particulièrement intéressant de faire le lien avec des méthodes de décomposition de domaine à 2 niveaux, telles que BDDC ou FETI-DP.

Zusammenfassung

Das Verständnis der Physik hinter komplexen physikalischen Vorgängen ist der Fokus der numerischen Simulation. Die Modelle hinter diesen Simulationen basieren häufig auf partiellen Differentialgleichungen (englisch: Partial Differential Equations, kurz: PDEs), welche mit entsprechenden Randbedingungen versehen sind [111]. Ausgehend von diesen Gleichungen, werden Diskretisierungsmethoden wie die Finite Elemente Methode angewandt. Durch die Anwendung auf einem Gitter entstehen oft große, dünnbesetzte lineare Gleichungssysteme der Form $Ax = b$ [53, 64]. Die Lösung dieser Gleichungssysteme ist eines der Hauptinteressen des Hochleistungsrechnens oder High Performance Computings. In den letzten Jahrzehnten wurden unter großem Aufwand viele Lösungsmethoden, auch Löser genannt, entwickelt, welche in parallel auf Supercomputern ausgeführt werden können.

Historisch wurden zwei Lösungsvarianten entwickelt [128]. Die direkten Methoden basieren auf einer Elimination von Unbekannten, welches zu einer Faktorisierung der Matrix führt. Diese Faktorisierung wird dann benutzt, um eine Lösung zu berechnen [46, 61]. Die iterativen Methoden verbessern eine Approximation durch sukzessive Aktualisierungen bis die Konvergenz zu einer gewünschten Genauigkeit erreicht wurde [75, 114]. Während die direkten Methoden sehr robust in der Problemlösung sind, sind ihre Kosten hinsichtlich Rechenzeit und Speicherbedarf im Vergleich zu iterativen Verfahren hoch. Iterative Verfahren hingegen hängen in ihrer Konvergenz stark von der Problemstellung ab. Die zuletzt eingeführten hybriden Methoden wurden entwickelt, um die Vorteile von direkten und iterativen Methoden zu kombinieren.

Mehrgittermethoden

Die Anwendung einer Diskretisierung auf eine PDE-Problemstellung kann auf Gittern mit unterschiedlichen Verfeinerungsstufen geschehen. Dies liefert verschiedene lineare Gleichungssysteme entsprechender Größe. Unter Verwendung einer Hierarchie solcher Verfeinerungen basieren Mehrgitter- oder Multigrid-Methoden auf zwei Beobachtungen [27, 129]. Erstens sind einfache iterative Verfahren in der Lage oszillierende Fehlerkomponenten schnell zu glätten während glatte Komponenten davon kaum berührt werden. Diese Verfahren werden daher auch Glätter genannt. Zweitens können glatte Funktionen auf gröberen Gittern gut approximiert werden[129]. In einem globalen iterativen Prozessen kombinieren Mehrgittermethoden die lokale Fehlerreduktion der Glätter mit einer Grobgitterkorrektur, um schnellstmöglich die Lösung des linearen Systems zu berechnen. Auf dem gröbsten Level ist das System im Allgemeinen so klein, dass ein

direkt Löser verwendet werden kann [109, 110]. Die Skalierung von Supercomputern auf großen Computersystemen ist schwierig. Die Verwendung von Mehrgittermethoden auf Supercomputern geht zumeist mit einer großen Zahl an Gitterverefeinerungen einher und das Verhältnis zwischen Gitterpunkten auf dem feinsten und gröbsten Gitter wird sehr hoch. Beim Übergang zu den gröberen Level verschlechtert sich dann die Effizienz der Berechnung wenn alle Rechenkerne verwendet werden sollen. In Kapitel I führen wir eine Agglomeration des Grobgitterproblems auf weniger Prozessoren aus um diesen Effekt zu vermeiden[101]. Die Konvergenz von Mehrgittermethoden ist theoretisch nur unter Einsatz von direkten Lösern für das Grobgitterproblem bewiesen [25, 73]. Für Systeme in denen das Modell oder die Daten mit Unsicherheiten behaftet sind, ist eine populäre Alternative die Anwendung von iterativen Lösern zur Approximation der Grobgitterlösung. In Fällen, in denen der iterative Löser Konvergenzschwierigkeiten aufzeigt, können auch direkte Löser mit approximativem Faktorisierungen verwendet werden. Letztere haben reduzierte Rechenzeit- oder Speicheranforderungen, sind aber weiterhin sehr robust. In Kapitel I studieren wir das Multigrid HHG (Hierarchical Hybrid Grids) Framework[18, 85] für Lösungen von Sattelpunktproblemstellungen aus der Erdmantelkonvektion mit springenden Koeffizienten und bis zu 10^{11} Freiheitsgraden. Wir zeigen eine Verbesserung des gesamten Mehrgitterschemas durch Verwendung von MUMPS [6] und einer block low-rank Approximation [1, 4] mit einfacher Genauigkeit für das agglomerierte Grobgitterproblem auf [30]. Dies ist eine wesentliche Verbesserung, z. B. für Erdmantel-Simulationsszenarien [16].

Block Cimmino iterative und pseudo-direkte Methoden

Gebietszerlegungsverfahren (englisch: Domain Decomposition Methods, kurz: DDM) sind neben Mehrgittermethoden die bekanntesten hybriden Verfahren [40, 98, 127]. Bei Gebietszerlegungsverfahren wird das globale lineare System in möglichst unabhängig voneinander zu lösende Subprobleme zerlegt, um das Beste aus den parallelen Rechnerstrukturen herauszuholen. Auf diesen Subproblemen kann dann in parallel ein direkter Löser ausgeführt werden. Ein globaler iterativer Prozess kombiniert die erhaltenen Teile, um die Konsistenz der globalen Lösung zu gewährleisten. Hierbei sind wir insbesondere an einer Klasse von iterativen Methoden interessiert, die als DDM aufgefasst werden können[40], die Block-Projektionsmethoden[34, 52, 88]. Basierend auf einer Partitionierung der Matrix in Blöcke von Zeilen und Spalten, berechnen diese Methoden sukzessive Projektionen auf den Unterraum, der von den Partitionen aufgespannt wird.

Unter den Projektionsmethoden sind die Block-Cimmino Iterationen von besonderem Interesse, da in jeder Iteration die Projektionen unabhängig voneinander berechnet werden

können. Die Konvergenz der Block-Cimmino-Iterationen ist langsam, aber eine Beschleunigung durch die Verwendung eines stabilisierten Block-Konjugierten-Gradientenverfahrens (englisch: Conjugate Gradients, kurz CG) beschleunigt werden[11, 112]. Das beschleunigte Block-Cimmino-Verfahren (BC) profitiert von einer Kombination von Iterationsreduzierung und der Verwendung effizienter Matrixoperationen [80]. Da die Konvergenz dieser Methode weiterhin problemabhängig ist, wurde eine Alternative mit zusätzlichen Variablen und Nebenbedingungen eingeführt, um die Partitionen zu orthogonalisieren. Block-Cimmino konvergiert in einer Iteration auf diesem augmentierten System und wir erhalten das pseudo-direkte Augmented Block Cimmino-Verfahren (ABCD) [47, 133]. Die zusätzlichen Variablen und Nebenbedingungen agieren auf die Verbindungen zwischen den Partitionen. Geometrisch und in der Sprache der Gebietszerlegungsverfahren führt die Erweiterung ein Splitting auf dem Interface zwischen Teilgebieten ein. Schlussendlich ist der zentrale Punkt in ABCD die vollständig parallele Konstruktion eines komprimierten Schurkomplements. Die Lösung dieser relativ kleinen Matrix kann bei Verwendung eines direkten Lösers immer noch enorme Speicherkapazitäten verlangen und in solchen Fällen sind die Block-Cimmino-Iterationen weiterhin eine gute Option. Diese zuvor genannten Ansätze wurden für die Betrachtung einer Zeilenpartitionierung eines unsymmetrischen quadratischen Systems entwickelt.

In Kapitel II erweitern wir die iterativen und augmentierten Methoden für Systeme mit vollem Rang auf die Lösung mit minimaler Norm von unterbestimmten Systemen und die Lösung von Kleinsten-Quadrate-Problemstellungen [50]. Letztere basieren auf einer Spaltenpartitionierung der Matrix. Im Falle unsymmetrischer quadratischer Systeme kann dann zwischen Zeilen- und Spaltenpartitionierung gewählt werden. Hierbei existieren je nach Anwendungsgebiet große Unterschiede in der Konvergenzgeschwindigkeit oder Augmentierungsgröße.

Parallele Implementierung des ABCD-Lösers

Um die numerischen Eigenschaften der iterativen und augmentieren Block-Cimmino-Methode zu verbessern, werden Präprozessierungstechniken wie im ersten Teil von Kapitel III eingeführt, verwendet. Nach Skalierung des Systems durch gleichzeitige Normalisierung von Zeilen und Spalten [113] werden die algebraischen Partitionierungsmethoden diskutiert [42, 112]. Wir unterstreichen die Verwendung von graph-basierten Partitionierern, welche entweder die Konvergenzbeschleunigung des BC-Verfahrens (GRIP Partitionierer) [126] oder die Minimisierung der Partitionierungsschnittstellen des ABCD-Verfahrens (PaToH Partitionierer) [32] zum Ziel haben.

Eine neue Augmentierungsmethode erlaubt die Verringerung der Augmentierungsgröße

in ABCD und verringert daher auch die Größe des Schurkomplements. Die iterative BC- und die augmentierte ABCD-Methode wurden sowohl unter Verwendung eines distributed- (MPI) [71] als auch eines shared-memory (OpenMP) [37] Ansatzes parallel implementiert. Der Löser folgt dem klassischen hybriden Parallelisierungsschema [133] der Gebietszerlegungsverfahren: bestimmte Berechnungseinheiten (Master genannt) erhalten eine oder mehrere Partitionen and berechnen unabhängig und in parallel die zugehörige Projektion mittels eines direkten Lösers. Im iterativen Schema summieren die Master ihre Projektionen in parallel und unter Verwendung von MPI Kommunikation auf, um die vorhandene Approximation zu aktualisieren. Wir führen einen neuen Algorithmus ein, der die künftige Kommunikation zwischen Mastern minimiert während er einen globalen Lastausgleich garantiert. Zwei zusätzliche Parallelisierungslevel werden mit dem direkten Löser verbunden. Die rechenintensiven, dichtbesetzten Kernels werden in shared-memory parallelisiert. Zusätzliche Recheneinheiten ohne Partitionen, sogenannte Worker, können mit einem Master verbunden werden, um den direkten Löser in parallel auszuführen.

Wir zeigen, dass die Auswahl der Recheneinheiten als Master oder Worker, je nach ihrer Position in der parallelen Rechnerarchitektur, zur Laufzeit einen wichtigen Effekt hat. Die Verteilung der Mastereinheiten über die Rechenknoten erhöht die Effizienz des Cachezugriffs deutlich und sorgt für eine Beschleunigung des ABCD-Lösers. Wir zeigen außerdem gutes Verhalten des ABCD-Lösers in Bezug auf seine Ausführungszeit und seinen Speicherbedarf, verglichen mit hochmodernen direkten Lösern wie MUMPS [7] [6] für quadratische oder QR MUMPS [8] [29] für rechteckige Matrizen.

Multigrid-inspirierte Relaxierung der Augmented Block-Cimmino-Methode

Während die iterative Methode niedrige Speicheranforderungen hat, ist ihre Konvergenz problemabhängig. Die augmentierte Methode konvergiert im Gegenzug nach einer Iteration, hat je nach Größe des Schurkomplements aber einen unzumutbar hohen Speicherbedarf. Im Kontext der diskretisierten PDE-Problemstellungen [53] ist es möglich die Partitionierung basierend auf der Gebietsgeometrie zu wählen. Die Größe des Schurkomplements steht dann im Verhältnis zur Größe des Interfaces zwischen den Teilgebieten. Motiviert durch Mehrgittermethoden nutzen wir eine Gitterhierarchie. Ein natürlicher Weg die Größe des Schurkomplements zu reduzieren besteht in der Augmentierung der Matrix, basierend auf einer gröberen Repräsentierung des Interfaces zwischen den Teilgebieten. Durch die Wahl eines bestimmten Grobgitterlevels können wir die Größe der Augmentierung kontrollieren. In Kapitel IV demonstrieren wir, dass die Block-Cimmino-

[7] `http://mumps-solver.org/`
[8] `http://buttari.perso.enseeiht.fr/qr_mumps/`

Methode, angewandt auf eine Matrix, die wie beschrieben augmentiert wurde, eine schnelle lineare Konvergenz für viele Systeme aus zweidimensionalen PDE-Problemstellungen aufweist. Die zentrale Aufgabe dieses Ansatzes ist die Konstruktion und Lösung der zusätzlichen Nebenbedingungen im System und daher der Schurkomplementmatrix. We führen eine Konstruktionsmethode, basierend auf der BC-Anwendung mit linearer Konvergenz auf kanonische Vektoren, ein. Die den Nebenbedingungen zugehörige rechte Seite wird während der Konstruktion gleichzeitig aktualisiert. Der Nachteil dieser Methode ist, dass die erhaltenen Nebenbedingungen dichtbesetzt sind. Schlussendlich führen wir Vorkonditionierungsmethoden für das Schurkomplement ein [75]. Hierbei werden seine Inverse als iterative Summe von Projektionen ohne explizite Kobnstruktion angewandt.

Ausblick und künftige Arbeiten

Nicht-disjunkte Partitionierung

Die Block-Cimmino-Methode als Gebietszerlegungsverfahren betrachtend ist ihre natürliche Weiterentwicklung die Einführung von Gebietsüberlappungen [40]. In diesem Kontext entsprechend Überlappungen nicht-disjunkten Partitionierungen für BC. In Abschnitt III.1.2.e führen wir eine Methode ein, welche iterativ eine existierende Partitionierung verbessert indem sie einige Zeilen dupliziert. Die Wahl der duplizierten Zeilen basiert auf den Werten der Normalengleichungen, inspiriert durch den GRIP Partitionierer [126]. Die Verdopplung dieser Zeilen kann für einige Problemstellungen zur effizienten Beschleunigung der BC-Konvergenz führen, für andere Systeme sind die hinzugefügten Verbindungen zwischen zuvor unverbundenden Partitionierungen kontraproduktiv. Ziel weiterer Forschung ist die Findung eines Algorithmus', der diese Verbindungen betrachtet und zur Konvergenzbeschleunigung beiträgt.

Mögliche Verbesserungen der relaxierten Augmentierung

Der Mehrgitter-inspirierte augmentierte ABCD wurde in dieser Arbeit für Zeilenpartitionierungen entwickelt. Der nächste natürliche Schritt ist die Erweiterung auf Spaltenpartitionierungen.

Die Hauptschwierigkeit ist die Konstruktion eines Block-CG-Verfahrens für die gleichzeitige Approximation der Nebenbedingungen und der zugehörigen rechten Seite.

Um die Methode auch für Aufgabenstellungen, welche nicht aus partiellen Differentialgleichungen resultieren, anwendbar zu machen, können algebraische Mehrgittertechniken betrachtet werden. Da die Mehrgitterkomponenten hierbei direkt aus den Matrixeinträgen

generiert werden, wird hierfür allerdings kein theoretischer Beweis erwartet.

Formal proofs of linear convergence

...was die eigentliche Entwicklung dieser These wäre. Für spezielle PDE-Probleme, oder unter Verwendung von algebraische Eigenschaften aus den Mehrgitterelementen wie z.B. die Approximationseigenschaft, it wäre es möglich, einige Eigenschaften hinsichtlich der Konvergenz unseres neuen Ansatzes aufzuzeigen. A besonders interessante Möglichkeit ist, diese Methode als 2-Ebenen-DDM zu interpretieren und die direkte Beziehung zu bestehenden Methoden wie BDDC und FETI-DP [90].

Introduction

> Deep in the human unconscious is a pervasive need for a logical universe that makes sense. But the real universe is always one step beyond logic.
>
> Frank Herbert, *Dune*

Scientific computing for system simulation is a hot topic for research and was one of the major reasons behind the development of *high performance computing* (HPC) facilities. Taking the example of nuclear fusion reactors or the simulation of Earth mantle convection, the comprehension of the physics behind complex physical problems require simulations when in-situ experiments are not feasible[111].

These simulations use models often based on partial differential equations (PDEs) coupled with some boundary conditions [53]. The discretization of these systems of equations with high resolution meshes results in huge sparse systems of equations [122] of the form $Ax = b$. This process is typical of recent large scale projects, e.g. the German funded project Terra-Neo[9] [16] (Integrated Co-Design of an Exascale Earth Mantle Modelling Framework), as illustrated in Figure 1. The goal of this project is the simulation of the Earth mantle convection in order to gain new insights on the driving forces for the plate tectonics, as well as on the causes of earthquakes and formation of mountains.

Great efforts were spent in the last decades in order to design solution methods for these linear systems, called solvers, executed in parallel on supercomputing facilities [41]. While solution methods for very small linear systems can already be found in

[9] `https://terraneo.fau.de/`

[10] `https://www.universetoday.com/40229/what-is-the-earths-mantle-made-of/`

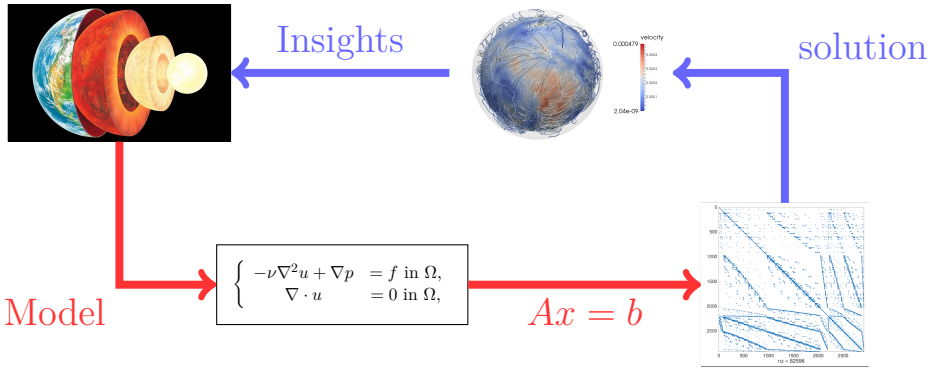


Figure 1 The numerical simulation process involves the solution of sparse linear systems from the discretization of PDEs.

Source: Earth composition from Universe Today; simulation and matrix generated from LSS FAU[10]

ancient Chinese texts [118], the history of modern solvers started with the introduction by Gauss of a technique based on the elimination of unknowns which gives a method to solve square linear systems of any size [61]. Two kinds of solvers were then developed. The direct methods are based on the factorisation of A in a specific form using the elimination of unknowns [46]. The technique used is directly inherited from the Gaussian elimination for square matrices, and using other methods such as the QR factorisation for rectangular matrices [69]. As for the iterative methods, they improve an approximation in a succession of updates until the convergence to a desired accuracy has been reached [9, 105]. While direct methods are robust with respect to the problem solved, their cost in terms of computation and memory is high compared to the iterative methods, for which the convergence is very dependent on the problem.
Lastly introduced, the hybrid methods were built in order to benefit from the advantages of both direct and iterative methods. *The topic of this thesis is the study of some hybrid methods, for square and rectangular systems, designed in order to be efficient on large scale supercomputers, with a special emphasis on systems arising from the discretisation of PDE problems.*

In Chapter I, we first introduce the different aspects of this topic. In particular, we study the multigrid methods, which use a hierarchy of grids with various levels of refinement to get fast methods [27, 129]. These multigrid methods are motivated by two facts [23]. First, some classes of iterative methods, called smoothers, have been observed to reduce quickly the oscillatory error components in a linear system, while keeping smooth components almost untouched. Secondly, the smooth functions are well

represented on a coarser grid level where they appear comparatively more oscillatory, and smoothers can again be applied efficiently. In a global iterative process, the multigrid methods recursively combine the local error reduction of smoothers with coarse grid corrections to rapidly compute the solution of the linear system. Considering the coarsest level, the corresponding linear system is generally considered small enough such that a direct solver can be applied.

The main focus in Chapter I is how to improve the efficiency of the multigrid schemes on large computing systems. Making multigrid methods scale, in the sense that they fully use the available computing power, is very difficult [109, 110]. In particular, an issue arises on the coarsest grid level where the size of the problem is so small that the computing efficiency deteriorates [101]. Typically, an agglomeration of the coarse grid data to solve on fewer processors can be used to compensate for this effect. On another line of idea, the convergence of multigrid methods is theoretically proven only when a direct solver is applied on the coarsest grid [25, 73]. For systems where the model and/or the data present some incertitude, a popular alternative is to only approximate the coarse grid solution using an iterative method with an appropriately chosen accuracy. In cases where the iterative solver has trouble converging, the use of a direct solver can then be advantageous. In order to further decrease the amount of computations and memory consumption, it is finally possible to combine the direct solver with an approximated factorisation.

This particular approach is the main contribution in Chapter I where we study the solution, using the multigrid *Hierarchical Hybrid Grids* framework (HHG) [18, 85], of a saddle-point problem with jumping coefficients up to 10^{11} degrees of freedom, inspired from Earth's mantle convection [14]. We then use the parallel direct solver MUMPS on the agglomerated coarse grid problem, combined with a block low-rank approximation [1, 4]. and the use of single precision arithmetic, to accelerate the computation. As a result, we demonstrate an overall improvement of the parallel scalability of the multigrid scheme thanks to the use of the MUMPS solver compared to the typical use of an iterative method on the coarse grid.

In the following chapters, we study some other hybrid methods based on the block projection techniques [52]. As suggested by their name, these methods are base on a partitioning of the matrix into blocks of rows or columns, and compute the solution through successive projections on the subspaces spanned by these partitions, e.g. using a direct solver. Among the projection methods, the block Cimmino iterations is of special interest in parallel computing since the projections are summed and can be computed

independently. This method is then hybrid in the sense that an instance of a direct solver is executed in parallel on each subproblem, and a global iterative method combines the obtained parts to ensure the consistency of the global solution. This scheme is typical of a classical type of hybrid solvers called the Domain Decomposition Methods (DDM) [40, 127]. The convergence of the block Cimmino iterations, geometrically linked to the principal angles between subspaces spanned by the partitions, is known to often be slow. An acceleration of the iterations is possible through the use of a stabilised block *conjugate gradient* (CG) algorithm [11, 112]. The *accelerated block Cimmino* (BC) takes benefit from a combination of iteration reduction, and the use of computationally efficient matrix operations. As the convergence of this method remains problem dependent, an alternative was proposed based on the augmentation of the original system with additional variables and constraints in order to orthogonalize the partitions. Block Cimmino applied on the augmented system is guaranteed to converge in one iteration, and we obtain the pseudo-direct *Augmented Block Cimmino method* (ABCD) [47, 133]. The additional variables and constraints are acting on the interconnections between partitions. Using the language of DDM, the augmentation introduces a splitting on the interfaces between subdomains. Finally, the central point of ABCD is the embarrassingly parallel construction of a condensed Schur complement matrix [133]. The solution of this relatively smaller matrix, using a direct solver, can require a large amount of memory, and in such case the block Cimmino iterations stay a good option. These two approaches were developed using a row partitioning for the solution of unsymmetric square systems. In Chapter II, we extend the iterative and augmented methods to full rank systems with the minimum-norm solution of underdetermined systems, and the solution of least-squares problems. The latter is based on a column partitioning of the matrix. In the case of unsymmetric square systems, the choice is then open between row and column partitioning which makes a large difference, in terms of convergence speed or augmentation size, for a wide range of applications.

In order to improve the numerical properties of the block Cimmino iterative and augmented methods, preprocessing techniques are applied, introduced in the first part of Chapter III. After scaling the system by a simultaneous normalisation of both rows and columns [113], the algebraic partitioning methods used are discussed. We emphasise the use of graph-based partitioners which specifically aim either at the acceleration of the convergence of the BC method, with the numerically-aware partitioner GRIP[126], or at the minimisation of the size of the augmentation in the ABCD method, with the hypergraph partitioner PaToH[32]. A new augmentation method additionally allows to

decrease the augmentation size in ABCD, and thus the size of the Schur complement.

The second part of the Chapter III, focuses on the implementation of the iterative method BC and the augmented method ABCD in the ABCD-Solver, based on MPI processes [71] (distributed) and OpenMP threads [37] (shared-memory). We introduce the hybrid parallelism of the ABCD-Solver [133]. From a partitioning of the matrix, specific MPI processes, called master, receive one or more partitions in order to compute the associated projection with a direct solver at each iteration. Concerning the distribution of partitions, we propose a new algorithm whose goal is to minimise the communication between masters, while keeping a global balance in workload. Two additional levels of parallelism are associated with the direct solver. The computationally intensive dense kernels used internally are parallelised in shared-memory [80]. Additionally, processes with no partitions, called workers, can be linked to a master to execute the direct solver in parallel [6]. We show that deciding which computing unit is a master or a worker depending on their place in the parallel architecture at runtime has an important impact. In particular, we propose an approach where the masters are spread over the nodes to accelerate the execution time of the ABCD-Solver. We finally compare the behaviour of the ABCD-Solver with the state-of-the-art direct solvers MUMPS[11], for square matrices, and QR-MUMPS[12], for rectangular matrices.

While the iterative BC method has low memory requirements, its convergence is problem dependent. The augmented method, on the contrary, converges in 1 iteration but is highly dependent on the solution of the Schur complement which solution can require prohibitive amounts of memory. In the context of discretized PDE problems, it is possible to choose the partitioning based on the geometry of the domain, then the size of the Schur is linked to the size of interfaces between subdomains. From multigrid methods, we use the idea of a hierarchy of grids. A natural way to decrease the size of the Schur is then to augment the matrix based on a coarser representation of the interfaces between subdomains. Through the choice of a specific coarse grid level, we can control the size of the augmentation. In Chapter IV, we demonstrate that the block Cimmino applied on the partitions augmented this way has a fast linear convergence for a wide range of systems arising from the discretisation of 2D PDE problems. We then introduce first tracks in order to efficiently construct the closure equation, including the Schur complement, with and without explicit construction.

[11] `http://mumps-solver.org/`
[12] `http://buttari.perso.enseeiht.fr/qr_mumps/`

Dear reader, I wish you a pleasant reading !

Chapter I

On the coarse grid solution of massively parallel multigrid

> We [Irving Kaplansky and Paul Halmos] share a philosophy about linear algebra: we think basis-free, we write basis-free, but when the chips are down we close the office door and compute with matrices like fury.
>
> Irving Kaplansky

In scientific computing, the models behind simulations often involves the solution of large sparse linear systems of the form $Ax = b$ coming from the discretisation of Partial Differential Equations (PDE) coupled with some boundary conditions [111]. In Section I.1, we start with the introduction of the PDE problems studied in this thesis and their discretisation [53]. We are especially interested in a simplified Stokes problem derived from the study of Earth mantle convection.

Great efforts were spent in the last decades in order to design solvers executed in parallel on supercomputing facilities. In Section I.2, we detail the issues involved in the development of efficient solvers in the context of *High Performance Computing* (HPC) using hybrid methods. Among these hybrid methods, there two main classes: the domain decomposition methods [40] and the multigrid methods [27]. In Section I.3, we then focus on improving the parallel efficiency at extreme scale of a multigrid scheme, using *the Hierarchical Hybrid Grids framework* [18, 85], thanks to a new combination of modern tools applied for the solution of the problem on the coarsest grid level.

The content of this chapter is derived from the article submitted to Wiley's Journal *Numerical Linear Algebra with Applications* in April 2020 *"Block Low Rank Single Precision Coarse Grid Solvers for Extreme Scale Multigrid Methods"*, A. Buttari, M. Huber, P. Leleux, T. Mary, U. Ruede, & B. Wohlmuth.

I.1 Partial Differential Equations and their discretization

Throughout this thesis, we consider several PDE problems which we introduce here. Only the flavour of what is required for the study of PDEs and their discretization is given here. For more details on this large topic of applied mathematics, see e.g. [53]. We focus on 2 classes of PDE problems.

I.1.1 Classical PDE problems

I.1.1.a Incompressible fluid flow

First, let's consider a Newtonian fluid of density ρ and viscosity μ moving in a 2 or 3 dimensional space Ω bounded by a surface $\partial\Omega$. We note the velocity of the fluid u, and the pressure p. f represents external body forces. Using the fundamental principles of conservation of mass, momentum, and energy, we derive the *non-linear* Navier-Stokes equations

$$\rho\left(\frac{\partial u}{\partial t} + u \cdot \nabla)u\right) = -\nabla p + \mu\nabla^2 u + \rho f \text{ in } \Omega,$$
$$\nabla \cdot u = 0 \text{ in } \Omega.$$

These equations are the basis of practical models for incompressible viscous fluid flow when combined with boundary conditions and initial data. We are focusing on $steady - state$ problems here and thus remove the time dependent term. When considering the kinematic viscosity $\nu = \frac{\mu}{\rho}$, we obtain the steady-state Navier-Stokes equations

$$\begin{aligned} -\nu\nabla^2 u + u \cdot \nabla u + \nabla p &= f \text{ in } \Omega, \\ \nabla \cdot u &= 0 \text{ in } \Omega. \end{aligned} \tag{I.1}$$

The Stokes equations

$$\begin{aligned} -\nu\nabla^2 u + \nabla p &= f \text{ in } \Omega, \\ \nabla \cdot u &= 0 \text{ in } \Omega, \end{aligned} \tag{I.2}$$

characterise the behaviour of so called Stokes flow. These are fluids flowing through narrow spaces and/or with a small velocity. As the viscous effects are dominating the inertial effects, a good approximation can still be obtained when dropping the non-linear quadratic term $u \cdot \nabla u$ from the Navier-Stokes equations (I.1), directly giving the *linear* Stokes equations (I.2).

These equations are particularly interesting for the simulation of Earth mantle convection, since the mantle is considered as an incompressible flow with a high viscosity, see Section (I.1.3). More realistic convection models sometimes include compressible flow formulations, pressure/temperature/strain dependent fluid viscosity, or sudden jumps of material properties [76]. E.g. the viscosity can abruptly vary by several orders of magnitudes due to steep temperature gradients between cold, subducting slabs and hot mantle material [124].

The Convection-Diffusion equation

$$-\nu\nabla^2 u + \overrightarrow{w} \cdot \nabla u = f \text{ in } \Omega, \tag{I.3}$$

is obtained from the Navier-Stokes equations (I.1) after dropping the pressure and linearising the quadratic term $u \cdot \nabla u$ through $\overrightarrow{w} \cdot \nabla u$, where $\overrightarrow{w}$ is known and often called the *wind*. This equation characterises the velocity of a fluid subject to a diffusive, and a convective effects.

The Poisson equation

$$-\nabla^2 u = f \text{ in } \Omega,$$

is the most famous elliptic partial differential equation. It is obtained when neglecting any convection effect in the previous equation (I.3). The homogeneous case $f = 0$ is called the Laplace equation.

I.1.1.b Wave equation

The second class of PDE problems we consider is derived from the wave equation. After using the technique of separation of variables to distinguish time and spatial parts of the

solution, the spatial component is the solution of the Helmholtz equation

$$-\nabla^2 u - k^2 u = f \text{ in } \Omega. \tag{I.4}$$

I.1.1.c Boundary conditions

Boundary conditions on u in $\partial\Omega$ must be defined such that the problem to solve is well-posed. There are three kind of conditions

1. Dirichlet $u = v_D$: specifies the value of the function on the boundary,
2. Neumann $\frac{\partial u}{\partial \vec{n}} = v_N$: specifies the value of the normal derivative on the boundary,
3. Robin $\frac{\partial u}{\partial \vec{n}} + \alpha u = v_R,\ \alpha \neq 0$: specifies the value of a linear combination of u and its normal derivative.

Different boundary conditions can be used on different parts of the boundary $\partial\Omega$.

In this chapter, we focus on the solution of saddle point problems arising from the Stokes equation (I.2), see Section I.1.3. Discretised problems based on the Poisson, convection-diffusion, and Helmholtz equation are used in Chapter IV. These 3 types of linear systems are typically used to assess the efficiency of a linear solver as the behaviour of solvers is generally dependent on the problem. In particular, Helmholtz equation and the convection-diffusion equation, with dominant convection effect, are known for giving hard linear systems to solve due to their numerical properties.

I.1.2 Discretization schemes

Many discretization techniques exist, each with their own pros and cons. We can cite among the usual methods

- Finite difference methods (FDM): these methods approximate derivative terms through truncated Taylor series. Combined with structured grids, these may be the easiest to implement and mostly encountered methods.
- Finite element methods (FEM): these are based on a finite subspace of the problem solution space. The solution is constructed from the basis functions spanning this subspace. These methods are very general and used to model complex problems and geometries. Thanks to their flexibility and adaptability, they make a powerful set of tools to model problems, from the Poisson equation to structural mechanics.

Note that The discretization of the PDE problems naturally involves the introduction of *discretization error*, i.e. that the discrete solution may be computed exactly but it is still different than the physical solution of the PDE problem. The use of a finer discretization when needed or more complex discretization methods such as Hybridised Discontinuous Galerkin (HDG) [35] can overcome this situation.

In this thesis, we introduce several methods for the solution of such sparse linear systems obtained using the FEM. In this chapter, we are interested in a challenging application inspired from the Earth mantle convection simulation.

I.1.3 Stokes on a spherical shell

Let $\Omega = \{x \in \mathbb{R}^3 : r_{\text{cmb}} < \|x\|_2 < r_{\text{srf}}\}$ be a spherical shell, where $r_{\text{cmb}} = 0.55$ and $r_{\text{srf}} = 1$ correspond to the inner and outer mantle boundary. We consider the generalised Stokes problem with velocity u and pressure p of the form

$$\begin{aligned} -\nabla \cdot \left(\frac{\nu}{2}(\nabla u + (\nabla u)^\top)\right) + \nabla p &= f \quad \text{in } \Omega, \\ \nabla \cdot u &= 0 \quad \text{in } \Omega, \\ u &= v_D \quad \text{on } \partial\Omega. \end{aligned} \tag{I.5}$$

The forcing term is given by $f = \text{Ra}\,\tau \frac{x}{\|x\|}$, where $\text{Ra} = 3.49649 \cdot 10^4$ is the dimensionless Rayleigh number and τ the normalised Earth's mantle temperature, as obtained from real-world measurements [120]. v_D is the Dirichlet boundary conditions, and ν is the positive scalar viscosity. We impose non-homogeneous Dirichlet boundary conditions derived from plate velocity data obtained by [104] on the surface and no-slip conditions at the core-mantle boundary. Here we assume that v_D satisfies the compatibility condition $\int_{\partial\Omega} v_D \cdot \vec{n}\, ds = 0$ where $\vec{n}$ is the unit outer-normal.

We discretize Ω by an initial tetrahedral mesh $\mathcal{T}_0$ following a polar decomposition, see Figure I.1. Then, we construct by uniform mesh refinement a hierarchy of meshes $\mathcal{T}_0 = \{\mathcal{T}_\ell, \ell = 0, \ldots, L\}$, $L > 0$.

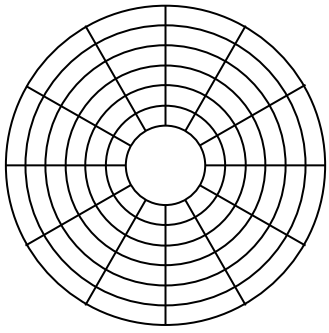


Figure I.1 2D discretization of a circle with 6 divisions in the radial direction and 12 divisions in the tangential direction.

For the discretization of (I.5), we apply the equal-order linear finite elements for

velocity and pressure, see e.g. [53]. This equal-order discretization is known to be unstable and must be stabilised [26]. To this end, we apply the pressure stabilisation Petrov-Galerkin (PSPG) technique [83]. Using (component-wise) nodal basis functions for velocity and pressure, we obtain a hierarchy of 2×2-block structured linear systems of the form

$$\begin{pmatrix} A_\ell & G_\ell \\ D_\ell & -C_\ell \end{pmatrix} \begin{pmatrix} u_\ell \\ p_\ell \end{pmatrix} = \begin{pmatrix} f_\ell \\ v_{D_\ell} \end{pmatrix}, \tag{I.6}$$

with $u_\ell \in \mathbb{R}^{n_{u;\ell}}$ and $p_\ell \in \mathbb{R}^{n_{p;\ell}}$. The dimensions of the velocity and the pressure space are denoted by $n_{u;\ell}$ and $n_{p;\ell}$. The divergence of the deviatoric stress operator in (I.5) is be associated with A_ℓ, the gradient with G_ℓ, and the divergence operator with D_ℓ. The C_ℓ-block originates from the PSPG-stabilisation. Problems of this structure are found in mantle convection simulations where they represent the most time-consuming computational tasks [14].

The viscosity in such problems can vary by several orders of magnitude and is typically non-linearly depending on u. In particular, we consider either the iso-viscous case ($\nu(x,T) \equiv 1$) or a viscosity profile, similar to the one used in [38], given by lateral and radial variations

$$\nu(x,T) = \exp\left(2.99\frac{1-\|x\|_2}{1-r_{\text{cmb}}} - 4.61\ T\right) \begin{cases} \frac{1}{10} \cdot 6.371^3 d_a^3 & \text{for } \|x\|_2 > 1 - d_a, \\ 1 & \text{otherwise,} \end{cases} \tag{I.7}$$

where d_a is the relative thickness of the asthenosphere. Thus, the Earth mantle is assumed to have layers with different viscosity characteristics. In particular, the asthenosphere, i.e. the outermost layer is assumed to be mechanically weaker. In the geophysics community, determining its depth is still an open research question [14, 38]. Here, we choose a depth of 410 km that corresponds to a viscosity jump by a factor 145.

Equation (I.6) can now be solved. Note that geodynamic simulations are subject to several types of errors, e.g., model or measurement error. In this particular context, we consider that a reduction of the residual by 5 orders of magnitude is suitable to obtain a solution with an appropriate accuracy. The efficient solution of this block system on large scale computations is studied in a number of recent articles [28, 123]. Combined with agglomeration techniques and an approximate direct solver on the coarse grid, we propose a scalable approach based on the multigrid framework HHG in Section I.3.

I.2 Linear solvers and High Performance Computing

In order to implement efficient numerical algorithms for the solution of sparse linear systems on modern computing architectures, one must first get a good understanding of the underlying techniques used to increase performance. Then, there remains the question of which linear solver should be chosen. Historically, two classes of solvers have been studied, namely direct and iterative, and the choice is based on the properties of the linear system. Lastly, hybrid techniques were designed in order to benefit from the advantages of both approaches, while making the best use of parallel computing architecture. The purpose of this section is to introduce the basic knowledge on *High Performance Computing* (HPC) and linear solvers to comprehend the contribution of this thesis.

I.2.1 High Performance Computing

Over the last 50 years, advances in the technology of processors have lead to a widening gap between the performance of processors (CPU) and the performance of memory, i.e. latency reduction. In 1975, Gordon Moore predicted that the number of transistors fit in a single processor would double every 2 years [102, 103]. This trend has been verified up until now for CPUs however, similar improvement in the access to memory takes around 7 years leading to a Processor-Memory performance gap growing of around 50% every year, see Figure I.2.

I.2.1.a Sequential performance: NUMA and ILP

In order to decrease the impact of this gap, chip designers used the *principle of locality* [77]. This principle states that if some data or instructions have been used recently in a program, they are likely to be reused soon (*temporal locality*), also data close in memory to the current data is likely to be used soon (*spatial locality*).

Based on the principle of locality, most modern architectures use a hierarchy of memory (NUMA or *Non-Uniform Memory Access*) including several levels of increasing capacity and decreasing speed. Usually, the cores have some internal registers, then there are 1 to 3 levels of cache memory inside the CPU, and finally we find the main memory, the RAM, see Figure I.3. The general idea is to load *once*, in the right level of cache, data and instructions likely to be reused in order to decrease the amount of (slow) accesses to the main memory. Issues still arise, e.g. coherency issues or cache misses,but through a

[1] `https://github.com/karlrupp/microprocessor-trend-data`

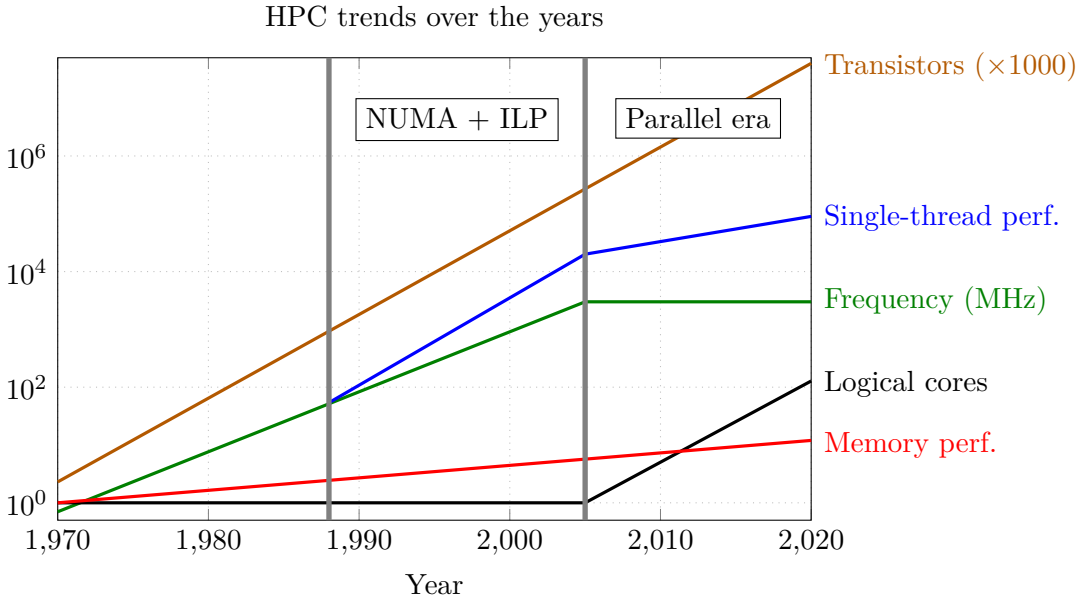


Figure I.2 Qualitative representation of the evolution for the CPU and Memory performance over the last 48 years.

Inspirations: CPU vs Memory from [77]; other trends from K. Rupp[1].

careful use of special structures in the code one can improve the efficiency of a program several folds.

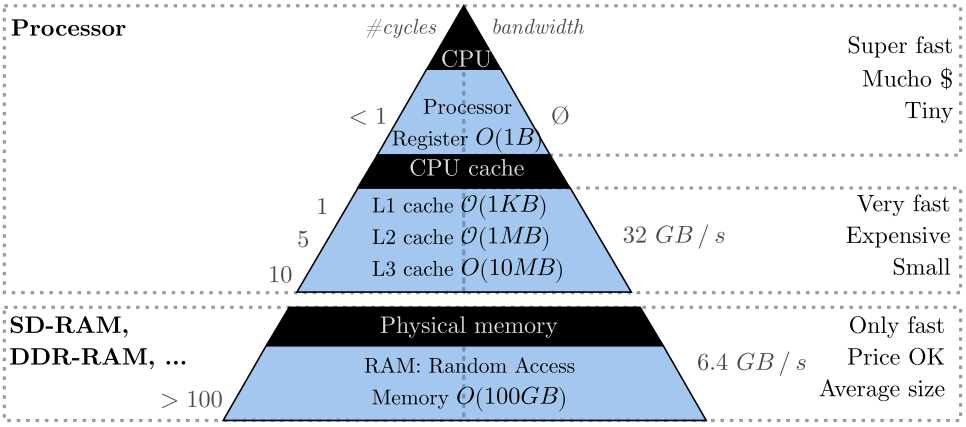


Figure I.3 Common memory hierarchy for modern computing architectures.

Source: `http://blog.zorangagic.com/2013/11/memory-hierarchy.html`

Another aspect of modern computing processors is the *Instruction Level Parallelism* (ILP). This term refers to a set of techniques allowing high processor throughput by

increasing the number of instructions (e.g. the sum of 2 floating point numbers) which can be performed during each clock cycle. Historically, ILP evolved from pipelining to current Multiple-Issue processors with, in particular, Intel's famous Advanced Vector Extensions which allow e.g. 16 double precision floating point instructions per cycle on Haswell architectures.

I.2.1.b Parallel programming

The latest challenge chip designers had to face was the limit of clock frequency in CPUs. While the frequency of processors increased steadily up to 4.7GHz in 2007, see Figure I.2, the heat generated at such frequency was a real issue (among other aspects). Then the frequency stagnated, which naturally brought chip designers and developers to a new era: multi-core processors and parallel programming. Now instead of hoping for ever better sequential performance, the idea is to use multiple computing resources concurrently and coherently. There are two types of parallel programming models

1. Shared memory parallelism occurs at the level of a single processor. In modern chips, cores are often separated in 2 sets, called *NUMA domain* or *sockets*, which share a same L3 cache allowing faster computation inside a single socket. These 2 sockets share the same main RAM memory. Computing units, called threads, then run on the physical cores concurrently and perform asynchronous read/write in memory.
2. Distributed memory parallelism allows units, called processes, to run on separate processors simultaneously, and cooperating for a global computation. These processes exchange data via messages. As the processes run on separate processors, this communication is carried on an interconnection network very slow compared to what access to the RAM memory is capable of. One of the goals in distributed programming is also to minimise the amount of communications.

In this thesis, we use the parallel programming with *OpenMP* (Open Multi-Processing) [37], using threads concurrently in shared memory, and with *MPI* (Message Passing Interface) [71], using distributed processes communicating through messages. MPI-OpenMP Hybrid parallel programming is possible which combines both programming models. In this case, the classical approach is based on divide-and-conquer. Considering a parallel computation, the complete problem is divided in evenly sized subproblems. Each subproblem is handled by a distributed process using threads for a shared-memory parallelism. The local result is then sent to the other processes. When dividing the global problem, one of the main issues is to find a good trade-off between subproblems with similar sizes and minimised communications between processes.

Once a parallel program has been implemented, remains the question of its scalability i.e. the evolution of its execution time depending on the amount of resources used and depending on the size of the problem. Considering t_n^s the execution time of the program with n computing units on a problem of size s. The usual measure is the Speedup $\mathcal{S}_u = t_1^s/t_n^s$. 2 classical studies of scalability are possible. The strong scaling considers a fixed problem size s and focus on the evolution of the execution time compared to sequential when increasing the computing resources. We define the strong scaling efficiency as

$$\mathcal{E}_{SS} = \frac{t_1^s}{n \times t_n^s}.$$

The weak scaling focuses on the evolution of the execution time compared to sequential when considering a fixed problem size per unit with an increasing number of computing units. We define the weak scaling efficiency as

$$\mathcal{E}_{WS} = \frac{t_1^s}{t_n^{n\times s}}.$$

For a perfectly scaling application, both $\mathcal{E}_{SS}$ and $\mathcal{E}_{WS}$ are close to 1. In practice, textbook scalability is very hard to obtain, both because of software issues (increasing amount of communication, low granularity of subproblems, ...) and hardware issues at very large scale.

As the end of Moore's law has been announced for the coming years [94], increasing amounts of efforts will be needed in terms of software design and parallel methods in order to tackle more challenging issues. Researchers are now preparing for upcoming exascale supercomputing centers [41], see the evolution on the TOP500 biannual ranking[2]. This is particularly the case for extreme scale simulations, and one of their most consuming part: the solution of very large sparse linear algebra systems.

[2] `https://www.top500.org/`

I.2.1.c Numerical linear algebra

The solution of linear systems can be traced back to 2 000 BC when the Babylonians knew how to solve 2×2 systems, while the Chinese were already able to solve a 6×6 linear system working on its numerical coefficients in 200 BC [118]. Then the tools opening the era of modern linear solvers appeared fairly recently, starting with the introduction of the determinant by Leibniz in 1693. Then, Cramer introduced a rule giving the solution of a square full rank system in 1750, without proof at the time. Finally, Gauss introduced in 1811 the famous systematic procedure now known as Gaussian elimination ([91], chapter 5). Nowadays, this method of elimination is still central for a class of solvers called the direct methods. The other classical type, the iterative linear solvers, is also linked to Gauss [115] who worked on such methods from the early 19th century, with Jacobi or Seidel [75]. Even now, great efforts are spent in order to improve these methods and make them usable for extreme scales, both in terms of cheer system sizes (now up to 10^{12} unknowns [93]) and computing resources.

Direct solvers

Direct solvers are methods based on the elimination of unknowns. Depending on the properties of the system, several methods exist. Commonly, Gaussian elimination is used for the solution of square linear systems [46], and the QR decomposition for the solution of least squares systems or to solve eigenvalues-problems [128].

Here, we consider the case of square linear systems of the form $Ax = b$ with A an invertible matrix of size $m \times m$, x and b vectors of size m. In this case, we can use the Gaussian elimination. This approach is the main method used on computers. The Gaussian elimination factorises the matrix A as $A = LU$, with L and U respectively lower and upper triangular matrices. This LU-factorisation is performed by substracting multiples of each row, divided by the diagonal element called *pivot*, from subsequent rows. The process requires $\mathcal{O}(\frac{2}{3}m^3)$ operations. Once the factorisation has been computed, the solution x for any particular right-hand side b is then computed through cheap forward elimination $Ly = b$, and backward substitution $Ux = y$. The advantage is that, once the cost of the factorisation is paid, multiple right-hand sides can be solved with a low number of operations ($\mathcal{O}(m^2)$).

Algorithms are constrained by floating point arithmetic on computers, far from the world of exact arithmetic. In this context, numbers are represented with a floating point format. We call machine precision, noted $\epsilon_{machine}$, the difference between a real number and its closest representation. This difference implies a rounding error from each *floating point operation* (flop), e.g. addition, multiplication, or division. Despite the individual

rounding errors are small, they remain far from negligible. In the Gaussian elimination, a pivot close to 0 can give an erroneous decomposition leading to wrong solutions. We say that classical Gaussian elimination is not backward stable [128].

The stability can be improved via preprocessing techniques such as scaling the matrix or techniques called *pivoting*. Pivoting consists in permuting the order of rows and columns of the matrix in order to avoid very small pivots. Obtaining an optimal reordering is an NP-complete problem but good heuristics exist, such as partial pivoting where better pivots are searched in the current column of the elimination [46]. Other pivoting techniques are found in modern direct solvers, e.g. using 2×2 pivots in MUMPS [6]. The condition number of a matrix is defined as

$$\kappa(A) = \left\|A^{-1}\right\| \|A\| .$$

This number measures both the sensitivity for the forward and backward operations, and does so for perturbations in x and also in A [128]. If we use the l^2-norm, the condition number is then $\kappa(A) = \frac{\sigma_{max}}{\sigma_{min}}$ where σ_{max} and σ_{min} are the largest and smallest singular values of A. $\kappa(A)$ is fundamental in numerical linear algebra and defines how accurately a system can be solved. Even in the presence of very ill-conditioned systems, direct solvers have the property to stay very robust as they guaranty to get a scaled residual of the order of machine precision. One should still expect to lose $log_{10}\kappa(A)$ digits in the solution computed by an algorithm with floating-point arithmetic.

The cost of direct methods stays a bottleneck in the general case: $\mathcal{O}(mn^2)$ flops for an $m \times n$ matrix. Taking advantage from the structure of the matrix to solve is a way to gain several orders of magnitude. In particular, matrices are often built such that most elements are zero on purpose, called *sparse matrices*. As seen in the previous section, one of the most important source for such matrices is the discretization of PDEs problems. The SuiteSparse Matrix Collection[3] [39] offers a database of sparse matrices from various real-world applications.
How much zeros to consider a matrix sparse ? There is no strict answer, usually we consider a matrix sparse as soon as we can positively exploit only its non-zeros. We note nz the number of non-zeros for a matrix $A \in \mathbb{R}^{m \times n}$, then $nz \ll m \times n$. There are multiple advantages to sparse matrices [46]

- *Storage:* matrices generated from the discretization of PDEs problems can be arbitrarily large, which ultimately gets prohibitive for storage. Several special

[3] `https://sparse.tamu.edu/`

structures have been developed to store only the non-zero elements while allowing efficient algebraic operations on matrices.

- *Computation:* only using the non-zero elements means less computation to perform at all stages. For a comparison, considering a sparse matrix A and a dense vector x, while the dense matrix-vector product Ax requires $\mathcal{O}(n^2)$ operations, the sparse equivalent requires only $\mathcal{O}(nz)$ operations. Dedicated libraries, e.g. *sparselib++*, exist to implement these sparse operators.
- *Parallelism:* the fact that a matrix is sparse means that independence between variables is high and exploiting this independence is the basis of the parallelism for parallel solvers.

A lot of modern parallel solvers have been developed in recent years to exploit efficiently the sparsity of matrices for sparse Gaussian elimination. The goal of these methods is to exhibit dense submatrices which are then processed using standard dense linear algebra kernels [33]. The main difficulties when using sparse LU factorisation is the so-called fill-in: upon factorisation the L and U factors may exhibit a lot more non-zeros than the original matrix [46]. Diverse reordering methods are used to prevent this issue, such as AMF[5] (Approximate Minimum Fill-in) or the use of graph-partitioning methods (SCOTCH[107], METIS[89]).

Two classes of sparse LU solvers can be distinguished: Multi-frontal methods[49], e.g. MUMPS[6], and the Supernodal methods, e.g. Pardiso[116], PasTiX[78] and SuperLU[97]. In both methods, an *assembly tree* is constructed which defines the dependence between dense submatrices and is a key ingredient for parallelism. Those dense submatrices are then processed by parallel standard dense linear algebra kernels. The basic difference between them is how this assembly tree is exploited. See e.g. [46] for a detailed description of these methods. Note that parallel direct solvers for rectangular systems have also been developed, e.g. QR-MUMPS [29] using the QR decomposition and based on the multifrontal method.

Iterative solvers

Iterative methods may be the right choice in order to implement a scalable solver and decrease significantly the memory requirements. Considering the linear system $Ax = b$ with solution x^*, the principle of these methods is to compute a converging sequence of approximate solutions, $x^{(k)} \underset{k \longrightarrow \infty}{\longrightarrow} x^*$, for the linear system.

The advantage compared to direct methods is that the computation of a single iteration requires a lower number of floating point operations and a lower amount of

memory. Generally, one or two vectors must be stored and the most costly part is a sparse matrix-vector product. The downside is just as important: iterative methods are less robust than direct methods, depending on the problem their convergence can vary from super fast to no convergence at all. The iterative methods are separated into 2 main classes, the fixed-point and the Krylov methods.

Starting from an arbitrary initial guess $x^{(0)}$, fixed-point methods are defined by the recursive relation

$$x^{(k+1)} = Qx^{(k)} + H, \tag{I.8}$$

where Q and H are matrices independent of k. The solver is built such that the iterations converge to a fixed-point, when $k \longrightarrow \infty$, which corresponds to the solution of the linear system. For any matrix A with eigenvalues λ_i, we define the spectral radius $\rho(A) = \max_i |\lambda_i|$ [114]. Fixed-point iterations defined from (I.8) converge if and only if $\rho(Q) < 1$ [69]. Generally, Q is called the iteration matrix.

Some classes of iterative methods called smoothers, e.g. Gauss-Seidel iterations, share the property that they remove local errors [27]. Over the iterations with an iterative method, the norm of oscillatory components of the error is damped rapidly while smooth components are mostly unchanged, see Figure I.4. We call this the *smoothing property* as standard iterative methods tend to quickly smooth the shape of the error [129]. This property is central for multilevel methods. Methods having this property include the damped-Jacobi, Gauss-Seidel and SOR. Note that smooth error components are actually geometrically smooth only in simple cases like discretized elliptic PDE problems. In the general case, smooth components are algebraically defined as those components not quickly reduced by the iterative method. The convergence of the fixed-point iterations, possibly slow, can be accelerated using polynomial techniques, also called Krylov methods [75].

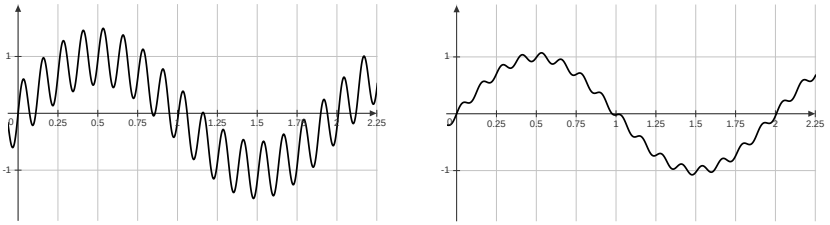


Figure I.4 Error associated to a linear problem before and after a few iterations of a classical iterative method. The oscillatory error component is damped.

Using Krylov iterative methods, the approximate solution is extracted from a subspace

of finite dimension much smaller than the dimension of A. These methods are based on projections, orthogonal or oblique, onto Krylov subspaces, i.e. subspaces spanned by vectors of the form $\Pi(A)v$ where $\Pi(A)$ is a polynomial in A [115]. In each iteration, we then have at least a sparse matrix-vector product to perform which is the most costly part of the computation. Considering $A \in \mathbb{R}^{m \times n}$, $m \leq n$, and $v \ltimes$, we call Krylov space of dimension m associated with A and v, the space $\mathcal{K}_m(A, v) = Span(v, Av, \ldots, A^{m-1}v)$.

The convergence of Krylov methods depends on the numerical properties of A. In these methods, $A^{-1}v$ is approximated as $\Pi(A)v$ where $\Pi(A)$ is a specific polynomial. It has been shown that the convergence of these methods is reached in less than n steps, where there is no upper bound for fixed-point iterations [75]. Still, n steps is large and we ideally would like to get a faster convergence which is possible through preconditioning. Adding preconditioning also on the right of the matrix, we solve for x the system $M_1 A M_2 y = M_1 b$ with $y = M_2^{-1}x$. One advantage of Krylov methods here is that neither the preconditioner nor the actual matrix have to be explicitly formed. We only need to apply them at each iteration in a cheap way, i.e. sparse-matrix product or the solution of simple linear systems, and in a parallelisable way in an HPC context. To get a fast convergence for the Krylov method, the spectrum of the preconditioned matrix must be well clustered. Ideally, when $\kappa \approx 1$ the convergence of the method is expected to be linear and fast.

A crucial point to define is when to stop the iterations. Stop too soon and the approximation is poor, stop too late and computing time is wasted. Following works from Wilkinson [130, 131], Oettli and Prager proposed their own backward error analysis [105]. Based on this theoretical framework, the authors in [9] define a set of stopping criteria which we use in this thesis. Note that all criteria used to stop the iterations can also be used to assess the quality of a solution computed with any method. First, we use the normwise backward error defined as

$$\omega_{A,b}(x^{(k)}) = \frac{\left\| Ax^{(k)} - b \right\|_\infty}{\|A\|_\infty \left\|x^{(k)}\right\|_1 + \|b\|_\infty}.$$

This backward error measures the norm of the smallest perturbation ΔA and Δb on A and b such that $x^{(k)}$ is the exact solution of $(A + \Delta A)x = (b + \Delta b)$, i.e. the distance between the initial system and the system actually solved. Alternatively, as it may be difficult to accurately compute the norm of A, we also use the normwise scaled residual

$$\omega_b(x^{(k)}) = \frac{\left\| Ax^{(k)} - b \right\|_\infty}{\|b\|_\infty}.$$

In the general case, it would be impossible to guarantee that neither of these criteria can be brought to machine precision. It may not even be advisable as the data used to construct A and b is itself perturbed most of the time . We then stop the iterations when the chosen stopping criteria is driven under a threshold chosen according to the original data.

Summary: Direct solvers are interesting for their high robustness to ill-conditioned matrices thanks to efficient pivoting techniques [81]. Additionally, though the cost of the LU factorisation is high, successive right-hand side is then solved cheaply with simple forward and backward substitutions. However, the memory required by the LU factorisation may quickly become prohibitive for very large problems, or problems with higher density, e.g. discretisations of 3D PDE problems. Iterative solvers have the advantage of being cheap, both in terms of flops and memory, as long as they display a fast convergence. And that is the main issue: these are very problem dependent methods which require specific preconditioning in order to be efficient in the general case. Fairly recently, a last type of methods arose which attempts to take the benefits from both iterative and direct methods: the hybrid methods.

I.2.2 Hybrid solvers

Hybrid methods were created specifically in order to get scalable solvers for very large linear systems in parallel computing through the combination of iterative and direct methods. Thanks to the iterative component, we split the complexity in memory and computation. The direct methods then bring their robustness. Additionally, the synergy between both approaches enables the natural management of several levels of parallelism to match the characteristics of modern computing architectures.

I.2.2.a Domain decomposition methods

The first approach we consider is *Domain Decomposition Methods* (DDM), see [125] for a review in the context of fluid flow simulation. The basic idea is very simple: considering a discretized PDE problem on a "complex" domain Ω. This domain is decomposed into p smaller (possibly overlapping) subdomains $\Omega_1, \ldots, \Omega_p$ constructed such that the PDE problem restricted to one subdomain is geometrically and computationally easy to solve with a direct solver. At each iteration of an iterative scheme, the solution of each subdomain is given to the neighbouring subdomains as boundary conditions in order to converge to a global solution in the end.

This process was first proposed by Schwarz in 1870 [117] to solve the Laplace's

equations. on a "complex" domain T of boundary L equal to the union of a circular domain T_1 and a square domain T_2 overlapping on T^*, see Figure I.5. Schwarz devised a method based on the known solution of the Laplace's equations for circular and square domains, called the *Schwarz Alternating Method* (ASM). The principle is to solve alternatively on each subdomain, using on the overlapping part the value computed in the other subdomains as a Dirichlet boundary condition (e.g. the value on L_2 from T_2 is used for the solution in T_1). These modified Dirichlet boundary conditions are also called compatibility conditions. Schwarz demonstrated that for any initial guesses $u_1^{(0)}$ and $u_2^{(0)}$, the process converges to the right solution. One hundred years later, Lions modified

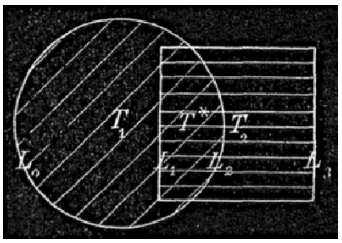


Figure I.5 The first domain decomposition method (Schwarz's original drawing).

the algorithm (and its proof) such that at iteration k, the 2 domains are independent which makes the method perfect for parallel computing [98]. While the first DDM was a method with overlapping subdomains, non-overlapping subdomains connecting only at the interface are possible. To converge with disjoint subdomains, a modification of the algorithm is necessary [99].

DDM are divide-and-conquer types of algorithms where the decomposition in subdomains splits the global complexity, under the condition that the algorithm has a fast convergence. The subdomains can be defined based on the geometry of the system or based on algebraic properties of the underlying global matrix. The convergence of Schwarz methods are dependent on the domains (global and sub), as well as on the size of the overlaps, and the interface conditions. Often faced with a slow convergence [60], the focus started to shift the DDM from solvers to preconditioners on the algebraic level.

Considering the linear system $Ax = b$, the matrix is split into smaller local matrices $A_i,\ i = 1, \ldots, p$ obtained by restriction matrices R_i such that $A_i = R_i A R_i^T$. In that case, the subdomains are constructed using graph partitioning methods to separate the unknowns with interfaces as small as possible. In a parallel computing context, these connections correspond to communications between computing units which must be minimised, while balancing the subdomain sizes which translates as having a good workload balance. In Section III.1.2.c, we study in detail possible partitioning choices for

the block Cimmino method which can be interpreted as a DDM on the normal equations of the system. Using the constructed local submatrices, we obtain the 2 algebraic iterative DDM [43, 60]

$$\text{(Multiplicative Schwarz) } x^{\left(k+\frac{i}{p}\right)} = x^{\left(k+\frac{i-1}{p}\right)} + R_i^T A_i^{-1} R_i (b - A x^{\left(k+\frac{i}{p}\right)}),\ i = 1, \dots, p,$$

$$\text{(Additive Schwarz) } x^{(k+1)} = x^{(k)} + \sum_{i=1}^{p} R_i^T A_i^{-1} R_i (b - A x^{(k)}).$$

Thanks to the independence between every component of the sum in the additive Schwarz, a great potential for parallel performance is revealed. As for overlapping subdomains, to get a convergence of the additive method a modification must be brought to the iterations by introducing diagonal matrices D_i which compensate for multiple contribution to the interfaces: $\sum_{i=1}^{p} R_i^T D_i R_i = I$. Using overlapping subdomains, the convergence improves with the size of the overlapping [60], at least in general. For problems like Helmholtz (I.4) or the convection-diffusion (I.3), this may not be verified and designing more complex transmission methods is central to the development of optimised Schwarz methods [54, 86]. In the case of p non-overlapping subdomains, once a matrix A has been partitioned, its unknowns can be reordered to the very typical bordered block diagonal form

$$A = \begin{bmatrix} A_{11} & & & & A_{1\Gamma} \\ & A_{11} & & & A_{2\Gamma} \\ & & \ddots & & \vdots \\ & & & A_{pp} & A_{p\Gamma} \\ A_{\Gamma 1} & A_{\Gamma 2} & \dots & A_{\Gamma p} & A_{\Gamma\Gamma} \end{bmatrix}, x = \begin{bmatrix} x_1 \\ \vdots \\ x_p \\ x_\Gamma \end{bmatrix}, b = \begin{bmatrix} b_1 \\ \vdots \\ b_p \\ b_\Gamma \end{bmatrix}.$$

The subscripts $1, \dots, p$ indicate unknowns in the interior of the subdomains, while Γ indicates unknowns on the interface between subdomains. After elimination of the unknowns $x_i,\ i = 1, \dots, p$, we construct a smaller system S called the Schur complement [127] and exclusively expressed on the interface such that

$$\begin{aligned} S x_\Gamma &= f, \\ S &= A_{\Gamma\Gamma} - \sum_{i=1}^{p} A_{\Gamma i} A_{ii}^{-1} A_{i\Gamma}, \\ f &= b_\Gamma - \sum_{i=1}^{p} A_{\Gamma i} A_{ii}^{-1} b_i. \end{aligned}$$

Once the Schur complement system has been solved, the local solutions $x_i,\ i = 1, \dots, p$

are then obtained directly with

$$x_i = A_{ii}^{-1}(b_i - A_{i\Gamma}x_\Gamma).$$

These equations solve independently the local systems using the interface value as Dirichlet conditions. The issue is that the Schur complement, though small compared to the original system, may be quite dense and/or ill-conditioned. Its solution usually requires specific preconditioning or the use of a direct solver. The augmented method from Section II.3 involves the solution of such Schur complement using a direct solver.

While increasing the number of subdomains splits the complexity of solving each local problem, the convergence of the DDM typically decreases due to a slower exchange of information between separate subdomains [40]. We say that the method is not scalable with respect to the number of subdomains, which is a big issue for parallel performance. It is possible to obtain methods scalable in this sense with the help of a coarser mesh level. Basically, the coarser mesh is used as a virtual subdomain and allows a faster transit of information between separate subdomains. This approach gave the 2-level and multilevel Schwarz methods [127] and is closely related to the method we introduce in Chapter IV, as well as the other type of hybrid solvers we now discuss: multigrid methods.

I.2.2.b Multigrid methods

Historically created for discretized elliptic PDE problems [23, 57], the *MultiGrid* (MG) methods uses of a hierarchy of increasingly refined meshes for the solution of linear systems. The global approach is an iterative process using a recursive combination of local error reduction, called smoothing, and global correction from a coarser mesh level.

Elements of multigrid

Let's consider we have a hierarchy of meshes $\Omega_l,\ l = 0, \dots, L-1$ with $L > 0$ with Ω_0 the finest mesh level. The linear system to solve $Ax = b$ is expressed as the discretization of the PDE problem on the finest grid. At level l, we consider the size of Ω_l is n_l. In general, for so-called standard coarsening (or refinement) methods, the number of coarse grid intervals is given by

$$n_{l+1} = \frac{1}{2^d} n_l, \tag{I.9}$$

where d is the dimension of the space [27]. These coarsening methods generally work on nested grids, i.e. a grid embeds the points from the next coarser grid, which is obtained by doubling the mesh interval size h on every dimension, see Figure I.6. This is only true for structured grids, but equation (I.9) still gives a good idea of the number of unknowns

after coarsening depending on the problem dimension. When the PDE problem has problematic properties such as anisotropy, it is better to use non-standard coarsening methods e.g. semi-coarsening [129].

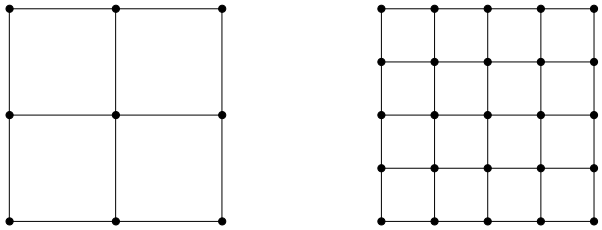


Figure I.6 hierarchy of 2D nested meshes on a uniform grid.

In order to transfer information from one grid to the other, we need a *Prolongation* operator $P_{l+1}^{l} : \mathbb{R}^{n_{l+1}} \longrightarrow \mathbb{R}^{n_l}$, and a *Restriction* operator $R_{l}^{l+1} : \mathbb{R}^{n_l} \longrightarrow \mathbb{R}^{n_{l+1}}$. There are several possibilities for the definition of these operators [25]. The usual choice for the prolongation operator is the *interpolation*. This operator transfers directly the value of a point if it is common to the 2 grid levels, else the value taken is the weighted average of the neighbouring coarse points. For non-nested grids, another choice is the *piece-wise constant* prolongation giving the value of the closest coarse point to all neighbouring points. Figure I.7 shows a 1D example for these prolongations. As for the restriction

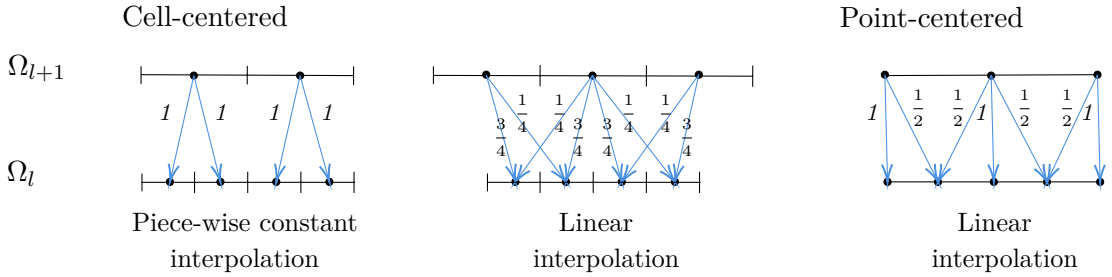


Figure I.7 Illustration of different interpolation operators for nested or non-nested grids.

operator, *injection* is possible for nested grids, i.e. taking at the coarse point the value on the fine grid, or *full-weighting* which is interpolation taken the other way around. While the prolongation and restriction can be built separately, they are generally set to respect the relation

$$P_{l}^{l+1} = cR_{l+1}^{l^T}, \tag{I.10}$$

where $c \in \mathbb{R}$ is a constant.

Now, while we have $A_0 = A$ the discretized linear operator on the finest level, how do

we define the linear operators A_l on all other levels ? One way is of course to discretize the PDE problem on the coarser grids. The most popular approach is to use instead the Galerkin coarse grid operator

$$A_{l+1} = P_{l+1}^{l} A_l R_l^{l+1}. \tag{I.11}$$

Together (I.10) and (I.11) are called the *variational properties*. We call *Geometric multigrid* (GMG) methods, techniques where the multigrid elements are constructed using the geometry of the domain. When no explicit grid defines the domain or the grid is highly unstructured, *Algebraic multigrid* (AMG) methods must be applied, where the transfer operators are built solely based on the numerical coefficient of the original matrix using aggregation methods[12]. Basically, these methods are based on the idea that unknowns linked in the adjacency graph of the matrix are strongly dependent and can be aggregated in a coarser representation [132].

2-grids cycle

Let's consider we have $L = 2$ levels of grid Ω_0 and Ω_1 with respective linear operators A and A_c, linked by the prolongation and restriction P and R. We compute the solution x^* of the system $Ax = b$. Starting from the finest level and an arbitrary initial guess x_0, the 2-grids cycle follows 3 steps

1. *(Pre-smoothing)* Apply a few iterations of a standard iterative method, called smoother, to obtain the approximation x_s. Thanks to the smoothing property introduced in Section I.2.1.c, the iterations quickly remove oscillatory components of the error on x. Only the smooth error

$$e = x^* - x_s, \tag{I.12}$$

   is left on Ω_0
2. *(Coarse grid correction)* Since the remaining error is smooth, it is well represented on a coarser grid. The goal is to compute a coarse vector e_c such that after prolongation we have $e \approx Pe_c$. On the coarse grid, this vector is obtained through the solution of the error equation

$$A_c e_c = r_c, \tag{I.13}$$

   where r_c is the coarse residual approximated as the restriction of the residual on the fine grid, i.e. $r_c = R(b - Ax_s)$. We generally consider the coarse grid is small

enough to solve (I.13) using a direct solver. Using the computed coarse error e_c, a correction is applied to the fine grid vector x_s. From (I.12), we obtain the corrected vector $x_c = x_s + Pe_c$.

3. *(Post-smoothing)* Through the coarse grid correction, some oscillatory components may be reactivated, thus a smoother is applied at last to obtain the final approximation x. If the accuracy of the obtained approximation is not satisfactory, go back to the first step using x as initial guess.

Figure I.8 illustrates 1 iteration of the 2-grids cycle process on a 2D Poisson problem. The 2-grids cycle is interpreted as the product of successive projections in [27]. The smoother then applies an approximate projection on the subspace corresponding to smooth components, and the coarse grid correction is a projection on the subspace not extrapolated from coarse vectors with P. If the multigrid is well constructed for the solved problem, these two subspaces are almost orthogonal and the convergence is fast.

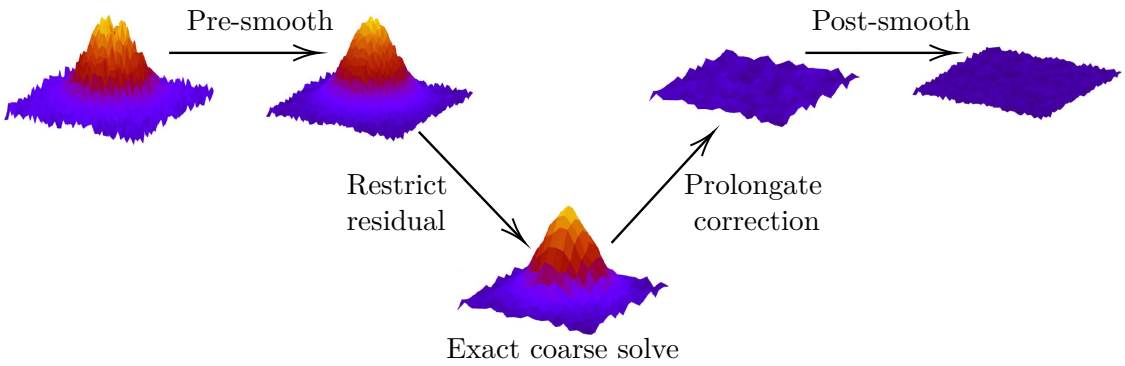


Figure I.8 Evolution of the error in an iteration of the 2-grids cycle, in the case of a Poisson problem.

Multigrid cycle

Here, we have a whole hierarchy of levels ($L \geq 2$) and there is good news. At step 2 of the 2-grids cycle, the restricted residual naturally appears oscillatory again after restriction on the coarse grid, see Figure I.9. Then, iterative methods become efficient again on the coarser grid. After smoothing the error equation (I.13), the residual can be transferred to an even coarser grid again. Once at the coarsest level, a direct solver is used on the error equations. Coarse grid correction is then applied to the finer level, followed by post-smoothing. The same correction is successively applied to all finer levels until we finally get back to the finest level approximation. This process is called a V-cycle, see

Figure I.9.

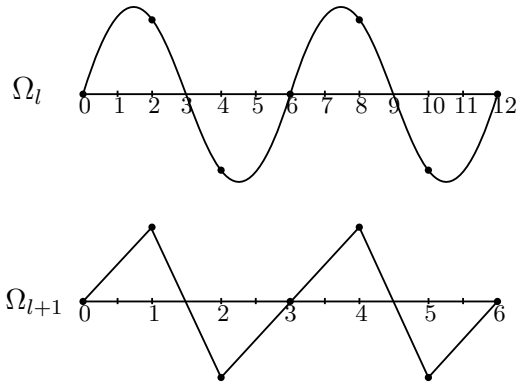


Figure I.9 The coarse grid "sees" a wave that is more oscillatory on the coarse grid than on the fine grid. *Source:* [27].

γ successive V-cycles can be applied at each level of the MG cycle, thus giving an infinity of classes of MG-cycles, see Figure I.10. The final kind of MG method we should mention is the *Full Multigrid* (FMG). In this scheme, an initial *solution* is computed on the *coarsest* grid then interpolated to a finer level where a V-cycle is launched. The obtained solution is then interpolated to the next finer level for a V-cycle, and so on until the solution finally gets to the finest level. The form of FMG really resembles the previous form of MG cycles, the real difference is that the approximated solution is also climbing up the mesh hierarchy, not just the corrections. FMG is the most efficient form of MG method [25], and corresponds to finding an excellent initial guess for a classical MG-cycle.

MG methods are the subject of a very rich literature, and in particular their convergence whose proof follows two main approaches. The first approach is based on the local Fourier analysis and was popularised by Brandt [23–25]. This method analyses the effect of applying multigrid to functions defined on an infinite grid with separate Fourier modes. From this, we obtain the *smoothing factor*, i.e. the worst factor by which oscillatory error components are reduced in one iteration of the smoother [129]. The other method, introduced by Hackbush [72, 73], is based on 2 notions: the smoothing property and the approximation property. Using these concepts, Hackbush shows the convergence of both

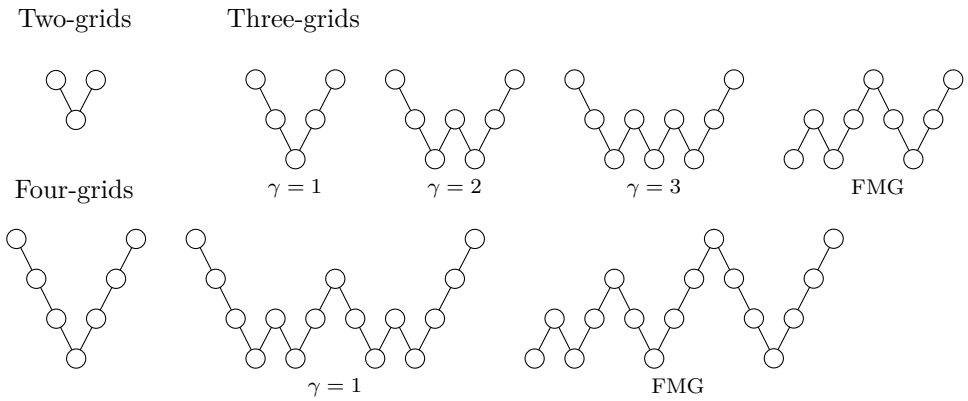


Figure I.10 Cycle structures, for various grid levels and number of cycle recursions γ.

2-grids and multigrid methods.

MG are techniques of choice in a parallel context. They may be asymptotically optimal, in the sense that the complexity to solve a linear system with sufficient accuracy grows only linearly with the number of unknowns. This has been proven for the full multigrid method [25].

I.2.3 Multigrid solution of saddle-point problems

We introduce a specific multigrid scheme to solve the Stokes type problem (I.5), inspired from Earth mantle convection, at extreme scale. These problems are often solved using a Schur complement conjugate gradient (CG) algorithm, or a preconditioned minimum residual method. In [44, 66], different types of solvers are compared for problems similar to (I.5) and it is found that a monolithic multigrid method for velocity and pressure combined performs best in terms of time-to-solution and memory. The family of 8 uniformly refined meshes $\mathcal{T}$ from I.1.3 is used. 2 levels are dedicated to obtain a properly defined coarsest grid problem, and 6 levels for the geometric multigrid method in the form of a mildly variable V–cycle that we note V_{var}. The V_{var}–cycle has the same form as a V–cycle but adds for each coarser level an additional number of smoothing steps. In our case, we add for each level two additional steps, each in the pre- and post-smoothing.

In this method, an Uzawa-type smoother is used that acts on velocity and pressure unknowns separately, see [134] and [44]. In the following, we refer to this monolithic MG variant as the *all-at-once* Uzawa MG method. The transfer operators are defined as linear interpolation for each component and their adjoint operators for restriction. Since this multigrid method acts on the whole Stokes system, we have to solve on the coarsest grid level again a saddle point problem. In theoretical considerations, one often assumes

that the coarsest grid problem is solved exactly. Getting this high accuracy may be computationally expensive in practice, e.g. using a direct solver. A popular alternative is to solve this coarse problem approximately. In this case, the tolerance has to be carefully selected to keep a mesh independent convergence for the multigrid scheme. In the next section, we explore new strategies to efficiently get such an approximated solution on the coarse grid problem in large scale and extreme scale computations.

I.2.3.a Hierarchical hybrid grids

For our studies the *hierarchical hybrid grids* (HHG) framework [18] is used, that provides data structures, parallelization and matrix-free concepts for extreme scale geometric multigrid computations. Here, we use the framework to explore coarse level strategies in large scale simulations. HHG achieves excellent performance on state-of-the-art petascale supercomputers. Problems with more than 10^{13} degrees of freedom (*DOFs*) [66] have been solved. For similar data structure concepts, we refer to [56, 84].

In this section, we briefly review the data structures and the parallel implementation of the considered multigrid framework. For more details, we refer to [14, 65, 66] and the references therein. HHG organises the nodal points of the mesh by employing the hierarchy of uniformly structured meshes $\mathcal{T}$. Through the refinement, grid points are generated on the edges, faces and within each input grid tetrahedron, also called macro tetrahedron. In the case of two tetrahedra, this is illustrated in Figure I.11 (left). Each of the nodes on each level of refinement is located on either the vertex, edge, face or the volume of the original macro tetrahedra. This structure is used to classify the nodes on each mesh level and to define container data structures that guarantee a unique assignment of each node to one container. In a distributed memory architecture, we assign each container uniquely to one processor.

To enable an efficient parallel communication across process boundaries, an additional layer of halos (ghost layers) is introduced that holds copies of *master* data, i.e. the original data, on other memory units. The data in these ghost layers can only be read and the values must be updated when the master data is modified so that they hold consistent values. In Figure I.11 (right), the ghost layer enrichment for two input mesh tetrahedra and the face container between them is illustrated.

To enable efficient parallel computations, load-balancing is also an important aspect. In HHG, the computational load can be identified with the dimensional complexity of the container data structures. Asymptotically, the volume containers produce the largest computational load, since they hold 3D data. Therefore, they are equally distributed to

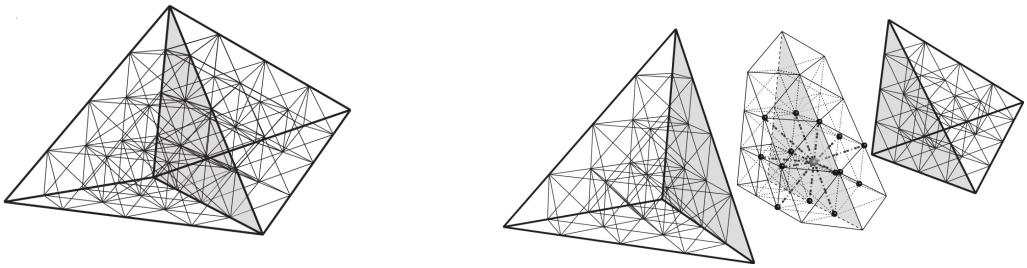


Figure I.11 *Left* two refined input elements; *Right* ghost layer structure of two input elements.

computing processes.

Matrix-free techniques are applied within HHG to avoid storing the FE matrices. In the classical assembly of the HHG framework only one stencil (i.e. a matrix row) needs to be stored per container, in the case of constant coefficient PDE problems, so that superior performance [18] is achieved. For curved domains such as the spherical shell that we consider in the following, the nodes that are generated through refinement do not reside on the boundary and thus do not fit with a simple single stencil representation. To fix this, an efficient surrogate assembly technique was developed which computes stiffness matrix entries approximately when applied, with no performance loss, by evaluating polynomials in order to avoid the expensive evaluation of the stiffness matrix using numerical quadrature [14, 15].

I.2.3.b Coarse grid solver

It now remains to choose a coarse level solver. Initially, we employ the standard method provided by the HHG package, i.e., a block-preconditioned minimal residual (*PMINRES*) iteration. This choice is motivated by the fact that Krylov space methods are easy to implement and parallelise. PMINRES is executed until the coarse level problem in each V-cycle has been solved with an accuracy corresponding to a reduction of the preconditioned residual by three orders of magnitude. The preconditioner here consists of *velocity* and *pressure* block preconditioner. For the velocity block, a Jacobi-preconditioned conjugate gradient (PCG) method is applied and for the pressure block a scaling by the lumped mass-matrix preconditioner for the pressure is used. The accuracy of the PCG method is specified by a relative residual reduction of two orders of magnitude. However, the error reduction depends on the condition number of the system matrix which deteriorates with the mesh size, and an increasing number of iterations becomes

necessary to solve the coarse grid problem with sufficient accuracy. The efficiency of the approach in many cases of interest has been demonstrated in previous publications [66], but also its limitations have been shown when viscosity models as (I.7) are considered [14].

We carry out our experiments on Hazel Hen, a petascale supercomputer at the HLRS in Stuttgart ranked on position 43 of the TOP500[4] list (June 2020). Hazel Hen is a Cray XC40 system with Haswell Intel Xeon E5-2680 v3 processors. Each compute node is a 2-socket system, where the 12 cores of each processor constitute a separate NUMA (non-uniform memory access) domain. Hazel Hen offers 64 GB per NUMA domain, which means around 5.3 GB per core. Hazel Hen uses the Cray Aries interconnect. The supercomputer has 185 088 cores in total for a theoretical peak performance of 7.42 Pflops/s.

In Table I.1, we present the total run-times (in seconds) of a V_{var}-cycle application for the scenario *iso-viscous* and the scenario *jump-410*, where the asthenosphere has a depth of 410 km, see (I.7). The displayed parallel efficiency is equal to the average total timing per iteration for the middle and large test cases compared to the average total timing per iteration for the smallest one. We observe that the scalability worsens for variable viscosity jump-410, with an efficiency decreased below 80%. For a more detailed analysis of the run-time behaviour, we also distinguish between the compute times for the coarsest grid and the finer grids. While the average run-time for the fine grids stays stable for both scenarios, resp. 89.1s and 88.7s for the largest problem, the average run-time for the coarse grid solution is getting worse with 11.6s with the scenario jump-410, compared to 2.7s in the case of iso-viscous. This is explained by the increased average number of iterations (*C.it*) for the convergence of PMINRES in the jump-410 scenario. Also, with the weak scaling, we observe that the average run-time per iteration is robust for the fine grids, while it deteriorates for the coarse grid. Note that this is expected, since we are using a sub-optimal coarse level solver and since the coarse grid problem size grows when scaling to larger number of processors. What is less expected is that the number of iterations does not increase for larger problem sizes. We are still investigating to understand the origin of this phenomenon.

For numerically challenging problems, and when the coarsest grid size is relatively large, such simple coarse grid solvers may become a bottleneck, especially since each iteration incurs a significant overhead. In the following, we propose an alternative fast and robust coarse level solver based on an approximate direct solver.

[4] `https://www.top500.org`

Table I.1 Total run-times (in seconds) of the V_{var} application: total, fine and coarse grid timings for the asthenosphere scenarios iso-viscous and jump-410. The number of iterations of the MG method (*it*) and the average number of iterations of the coarse grid solver (*C.it*) are also displayed.

proc.	**DOFs**		**iso-viscous**						**jump-410**					
	fine	*coarse*	*it*	*total*	*fine*	*coarse*	*eff.*	*C.it*	*it*	*total*	*fine*	*coarse*	*eff.*	*C.it*
1 920	$5.4 \cdot 10^9$	$9.2 \cdot 10^4$	4	313	309	3.8	1.00	25	15	1186	1133	53.4	1.00	68
15 360	$4.3 \cdot 10^{10}$	$7.0 \cdot 10^5$	5	440	430	9.7	0.89	18	13	1188	1091	96.8	0.87	49
43 200	$1.2 \cdot 10^{11}$	$1.9 \cdot 10^6$	8	735	713	21.9	0.85	17	14	1404	1242	162.5	0.79	48

I.3 Approximate Coarse Grid Solvers for Extreme Scale Multigrid Methods

Scaling GMG on large computing systems is hard in the sense that on the coarse grid levels the granularity deteriorates. Commonly, an agglomeration of data onto fewer processors is used to compensate for this effect. We propose a new approach combining agglomeration to an external approximate direct solver on the coarse grid to obtain a scalable multigrid for saddle-point problems. Our solution strategy at the coarse level consists of the following four steps

1. Convert HHG format to sparse matrix data-format (*Coordinate list format* (COO) for MUMPS).
2. Apply an agglomeration technique.
3. Solve the coarse level problem using the external library.
4. Redistribute and convert the approximation to the HHG format.

I.3.1 Agglomeration

The number of DOFs per process decreases drastically on coarser grid levels. When the balance between computation and communication worsens, and the communication overhead becomes a concern, then we propose to accumulate the data from several processes onto a single process. Thus the coarsest grid problems are redistributed. The remaining processors can either perform redundant computations or the unneeded processors stay idle [109, 110]. In our implementation, we use an external software library as coarse grid solver. Processors which are not used by the external solver will stay idle.

Within the HHG framework, we achieve this with techniques similar to [101], where routines specific to PETSc are used. In our method, we collect the data from several processes and accumulate it to a single process. This defines a reduction factor $r \in \mathbb{N}_{\geq 1}$

denoting how much the overall process count $|\mathcal{P}|$ is reduced such that we get

$$m = |\mathcal{P}|/r \tag{I.14}$$

master processes. Here, we assume, for simplicity, that the reduction factor r is a divisor of $|\mathcal{P}|$. We distinguish 2 agglomeration techniques

1. *Master-workers agglomeration:* all physical cores are used for the computation on the fine grids in the multigrid scheme, and only a subset of them is used for the coarse grid.
2. *Superman agglomeration:* from the beginning, a subset of all cores is only dedicated to the coarse grid solve. During the first cycle, the factorisation of the coarse grid matrix is performed simultaneously with the computations on the finer grids.

An example with reduction factor $r = 3$ and $m = 2$ is shown in Figure I.12. In both

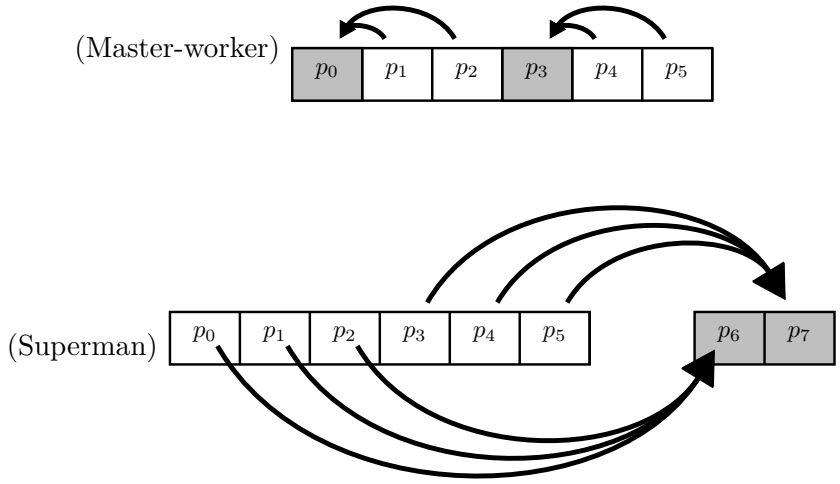


Figure I.12 Showcase with reduction factor $r = 3$ and $m = 2$, for agglomeration techniques.

cases, during the solution phase on the masters, the remaining processes stay idle. Then the solution is distributed back to the original processes. The Superman agglomeration is possible because only a very limited number of processes is used on the coarse grid after agglomeration, thus only a few additional nodes must be allocated, see Section I.3.3.

The application to sparse matrix data formats involves array-like data structures like C++ vectors, which make the agglomeration technique easy to implement and efficient to apply, since we only need to concatenate vectors and let the external solver be executed by a reduced communicator.

This approach is very pragmatic and so is the selection of a suitable r. The latter depends on the granularity of the problems solved by the direct solver and have to be determined on a case by case basis. Using the factor r, the agglomeration method can be adapted to the parallel architecture of the machine in our case, we have chosen to agglomerate all the data that resides in the same node. This makes agglomeration possible with small communication overhead, but may put extra communication burden on the parallel coarse grid solver. At the other extreme, compacting all the processes inside a same node is not advisable because of intensive memory usage by the memory bound dense kernels used by MUMPS. In the absence of time dependency in the problem to solve, the agglomeration of the system matrix is performed only once, and then kept in memory on the master processes.

I.3.2 MUMPS

MUMPS (MUltifrontal Massively Parallel direct Solver)[5] [4, 6] is a package for solving sparse systems of linear equations like (I.6) with symmetric (positive-definite or indefinite) or unsymmetric matrices, using single or double precision real/complex arithmetic. It is based on Gaussian elimination. In the multigrid context, the use of such sparse direct solver as coarse level solver provides two distinct advantages

1. Robustness in terms of accuracy and execution times, even when iterative solvers show slow convergence.
2. Saving the analysis and factorisation in memory, the most time consuming parts, in a preprocessing step. Fast coarse level solves through application of the stored factorisation for each multigrid cycle.

I.3.2.a Method

The MUMPS solver is based on the multifrontal scheme as previously introduced [6]. Like most direct solvers, MUMPS achieves the solution of a system in three steps

1. *Analysis* at this step a sequential preprocessing of the matrix is performed in order to reduce the fill-in and improve the linear system (scaling, permutation to a zero-free diagonal); this is followed by a symbolic factorisation defining dependencies between the unknowns of the system (*elimination tree*), finding dense subproblems in particular.
2. *Factorisation*: this step computes the numerical factorisation of the matrix, based on the analysis. Two levels of parallelism are managed: one comes from the

[5] `http://MUMPS-solver.org/`

independency between dense subproblems (*fronts*), the other is intrinsic to the dense linear algebra kernels applied to the fronts.

3. *Solve*: the factors are used to compute the solution through forward elimination and backward substitution.

Parallelism in MUMPS is implemented through a hybrid MPI/OpenMP model which makes the solver suited to modern distributed memory Machines equipped with multicore processors.

I.3.2.b Block low-rank approximation

In multigrid methods, the coarse grid is assumed to be solved exactly in theoretical considerations; in practice, it is common to approximate the coarse grid solution up to a given tolerance. MUMPS offers a mechanism that allows the reduction of the solution cost in exchange for a lower accuracy via the *block low-rank* (BLR) method.

Full rank sparse matrices also result in full rank fronts in the sparse factorisation. Nonetheless, it can be proven that for problems in a very broad class of applications, conveniently defined off-diagonal blocks of the fronts can be approximated with accuracy ε using a low-rank product [17]. In most cases, even with an accuracy close to the working precision, this mechanism allows for considerable reduction of the cost for the linear algebra algorithms both in terms of memory consumption and floating point operations. Several approaches have been proposed in the literature to take Advantage of this low-rank property. The MUMPS solver is based on a matrix format called BLR [1, 4] where the matrix is partitioned into blocks in a checker-board fashion and block-wise low-rank approximations are exploited to significantly reduce the theoretical complexity [2] and practical cost of the factorisation and solve phases. Although even lower theoretical complexities can be achieved by using multilevel [3] or hierarchical [74] approximations, the flexible BLR format has proven to be very efficient in the context of a general purpose, fully-featured sparse solver such as MUMPS [4, 100].

Additionally, running MUMPS in single precision arithmetic decreases the overall memory usage and time consumption of the solver. This arithmetic can be combined with the BLR approximation freely with no loss in the accuracy of the solution, as soon as the choice of the parameter ε gives an approximation with accuracy lower or equal to single precision [82].

I.3.3 Scaling experiments

In this section, we study the performance of the multigrid solver when combined with a block low-rank method using single precision arithmetic as coarse level solver for the Stokes problem introduced in Section I.2.3.b. We start with a performance study of the MUMPS solver standalone combined with agglomeration on the coarse level system in the Section I.3.3.a. Then, we use the best obtained configurations to improve the overall multigrid solver performance in a weak scaling test in Section I.3.3.b.

I.3.3.a Approximate sparse direct solver with agglomeration

For the 3 problem sizes of Section I.2.3, we seek the best agglomeration factor r to run MUMPS, in the sense that the execution time is minimised on the resulting m processes. We run MUMPS as a standalone solver, with exact double precision accuracy on the coarse grid matrices extracted from HHG. In Table I.2, we display the results from a scaling study where we increase the reduction factor r, see (I.14). The focus lies on the more challenging problem variant, i.e. the jump-410 scenario. We consider that r is a divisor of the original number of processes. We are looking for a trade-off between a high number of active cores (r not too high) and a combination of large granularity and low memory concurrency for the solver (r not too low), both cases leading to a shorter run-time.

Using all fine grid processes for the coarse grid produces a large run-time due to the excessively high volume of communications. In this scenario, the resulting set of unknowns per process is very small (less than 152 DOFs). This is clear for the smallest problem and $r = 1$, for which the total timing is more than 10 times higher than after a reduction by $r = 24$, which corresponds to removing concurrent memory access by accumulating the data of a whole node to only one process. Increasing r further decreases the overhead in communication until a too high r gives low parallelization compared to the granularity of the problem. We observe this effect for the biggest case where having $r = 192$ decreases the total timing by around 40% compared to $r = 24$, and having $r = 576$ then increases the total run-time again. To obtain minimal run-times, while increasing the problem size, we must further increase the reduction factor: for the problem with 1 920 processes the best choice is $r = 48$, with 15 360 processes, $r = 92$, and with 43 200 processes, $r = 192$. While all the timings increase when increasing the problem sizes, we observe only a very small run-time for MUMPS solve phase, which is performed at each cycle. Over several multigrid iterations, the cost of analysis and

factorisation is steadily amortised.

Table I.2 Scaling of the sparse direct solver MUMPS (with exact double precision arithmetic): analysis, factorisation and solve run-times in seconds. r: reduction factor, m: corresponding number of processes.

r	**1 920**				**15 360**				**43 200**			
	m	*analysis*	*fac.*	*solve*	*m*	*analysis*	*fac.*	*solve*	*m*	*analysis*	*fac.*	*solve*
1	1 920	6.5	10.8	13.66	-	-	-	-	-	-	-	-
24	80	1.6	1.1	0.03	640	15.6	31.1	0.86	1 800	66.6	248.2	1.87
48	**40**	1.6	0.9	0.03	320	14.6	20.7	0.28	900	45.0	199.5	0.67
96	20	1.6	1.2	0.03	**160**	13.7	19.6	0.20	450	53.6	173.0	0.73
192	10	1.7	1.7	0.07	80	14.4	24.1	0.22	**225**	41.0	134.6	0.56
576	-	-	-	-	-	-	-	-	75	42.9	158.4	0.59

In a next step, we fix the optimal reduction factor previously found for MUMPS in full-rank, assuming it stays relevant for different parameters. We now compare the performance of the BLR method, in double and single precision, with the standard sparse direct solver (Full Rank). In order to respect the setup after agglomeration in HHG for the coarse grid solver, each process runs on a separate node. The choice of the BLR ϵ parameter is important, since it controls the accuracy of the approximation as well as the performance of the factorisation. Furthermore, for different problem setups, we are interested in the robustness of the BLR with threshold ϵ in terms of solution accuracy, when increasing the problem size.

In Table I.3, we consider three different resolutions of the problem for the iso-viscous and for the jump-410 scenarios, (I.7). We compare the accuracy of the approximated solution through the scaled residual for 2 settings. Full Rank gives machine precision accuracy, while for BLR with $\epsilon = 10^{-3}$ the scaled residual is about 10^{-4}. The latter is the accuracy level that we typically require in order to keep the convergence of the MG scheme unchanged. In the next section, we see that the largest problem is the most critical one and a more detailed study is required such that we also include data for BLR with $\epsilon = 10^{-5}$ which gives an accuracy of the order 10^{-6}. The processor specification remains unchanged compared to the previous experiments. Hence, as theoretically proven in [82], the accuracy is controlled by the BLR ϵ and stays robust for different problem sizes when including viscosity variations. Furthermore, as long as ϵ stays safely below 10^{-8}, we can use single precision arithmetic without changing the quality of the solution [82]. This helps to further decrease the cost in memory and computation.

When applying the BLR method for different ϵ in comparison to the Full Rank, we observe a small increase in the timings for the analysis, around 15% for the biggest case. This overhead comes from the identification of blocks for the factorisation, necessary to the BLR method. The biggest effect of the BLR approximations on the performance of MUMPS is on the factorisation. The first evidence is a reduction of FLOPS for the factorisation: when using BLR $\epsilon = 10^{-3}$ in comparison to the Full Rank factorisation. The FLOPS are reduced up to a factor 10 for the biggest test case which translates in a run-time reduction of a factor 4. Additionally, the use of BLR approximations also reduces the cost of the solve phase, by about a factor 2 for the biggest test case.

To further accelerate the computation, we also turn to single precision arithmetic. In this case, the analysis and solve phases stay mostly unchanged, while we get a reduction of 30% of the factorisation run-time, originally dominating. Overall, using BLR combined with single precision reduces the total run-time of a complete MUMPS computation by a factor up to 2.6 for the largest problem. Finally, for a fixed problem size, the run-times for all phases of the solver as well as the accuracy of the solution appear robust with respect to the problem type iso-viscous or jump-410. For this reason, we focus on the scenario jump-410, most challenging for the PMINRES solver, in the rest of the paper.

I.3.3.b Multigrid solver combined with the approximate coarse level solver

Finally, we use the MUMPS sparse direct solver and its BLR variant to approximate the coarsest level problem within the Uzawa multigrid solver in the HHG framework. We compare the run-times of one V_{var}-cycle for this strategy with the ones using the PMINRES solver on the coarsest grid of Section I.2.3.b in a weak scaling test.

In Table I.4, we present the total run-times with up to 43 200 processes. We present again the total run-time over the V_{var}-cycle iterations. We distinguish the fine grid run-time, the run-time for MUMPS analysis and factorisation cumulated and the coarse grid run-time (i.e. MUMPS solve phase) separately. Additionally, we include the total time for data transfer, i.e. the time for agglomeration plus data conversion from HHG to MUMPS and vice versa. To get the best performance from the coarse level solver, we use the agglomeration factors $r = 48, 96$ and 192 respectively, according to the previous study in Table I.2. We use here the *master-worker agglomeration* technique. For the largest test case, we had to reduce ε from 10^{-3} to 10^{-5} in the BLR method to obtain the same iteration number as in the Full Rank case. The value of the scaled residual is still comparable with the other problem sizes when using $\varepsilon = 10^{-3}$. This justifies that an

Table I.3 Influence of the viscosity scenario and BLR ϵ parameter, with double and single precision, on the accuracy and the run-time of the direct solver. Run-times (in seconds) are separated in analysis, factorisation and solve steps. Each process runs on a separate node.

proc.	**DOFs** *coarse*	**type**	**BLR** ϵ	**analysis** *time*	**factorisation** *Flops red.*	**factorisation** *time*	**solve** *time*	**scaled res.**
40	$9.2 \cdot 10^4$	iso-viscous	Full Rank	1.51	100.0	0.86	0.03	$1.2 \cdot 10^{-18}$
			10^{-3} + single	1.74	26.2	0.70	0.01	$6.7 \cdot 10^{-5}$
		jump-410	Full Rank	1.55	100.0	0.88	0.03	$6.0 \cdot 10^{-18}$
			10^{-3} + single	1.74	26.0	0.67	0.01	$2.5 \cdot 10^{-4}$
160	$7.0 \cdot 10^5$	iso-viscous	Full Rank	13.73	100.0	20.65	0.21	$1.1 \cdot 10^{-18}$
			10^{-3} + single	15.91	10.8	6.81	0.09	$2.3 \cdot 10^{-4}$
		jump-410	Full Rank	13.74	100.0	19.58	0.20	$4.8 \cdot 10^{-18}$
			10^{-3} + single	15.86	10.5	6.62	0.09	$7.5 \cdot 10^{-5}$
225	$1.9 \cdot 10^6$	iso-viscous	Full Rank	41.10	100.0	139.17	0.55	$7.9 \cdot 10^{-19}$
			10^{-5}	47.24	12.9	35.51	0.28	$4.4 \cdot 10^{-7}$
			10^{-5} + single	47.31	12.9	25.17	0.26	$4.8 \cdot 10^{-7}$
			10^{-3} + single	47.40	7.7	21.07	0.20	$2.1 \cdot 10^{-4}$
		jump-410	Full Rank	41.02	100.0	134.61	0.56	$1.5 \cdot 10^{-18}$
			10^{-5}	47.56	13.0	36.98	0.30	$2.4 \cdot 10^{-6}$
			10^{-5} + single	47.65	13.2	25.63	0.27	$1.4 \cdot 10^{-6}$
			10^{-3} + single	47.62	7.6	21.16	0.19	$4.7 \cdot 10^{-5}$

accuracy around 10^{-5} for the coarsest grid problem is the bare minimum needed for the considered class of problems.

First, as expected, we observe that the average run-time for the processing of the fine grids is very similar to those we observe when using PMINRES, except for some small usual variations at extreme scale [121]. The coarse grid solve stays below 1s, i.e. less than 0.1% of the total run-time over the iterations. Only the portion of the analysis and factorisation step should be seen critical. The run-time of the single precision BLR method, ana.fac.+coarse, is accelerated by a factor 2.3 in comparison to the direct solve, while the number of iterations stays unchanged. Comparing the coarse grid timings for single precision BLR in MUMPS, including the analysis, factorisation and data transfer, to the ones of the PMINRES solver of Section I.2.3.b, we observe a 50 % reduction of the total run-time, from 162.5s to 83.6s for the largest case. We also get a 6% points improvement of the overall parallel efficiency of the multigrid scheme with 43 200 processes.

Table I.4 Weak scaling of the V_{var}-cycle with a sparse direct and a block low-rank coarse level solver with or without single precision (SP) arithmetic. The parallel efficiency compares the average total run-time of each run to the average total run-time of the smallest case with no BLR.

proc.	**DOFs**		**BLR** ϵ	**it**	**time (s)**					**eff.**	**scaled res.**
	fine	*coarse*			*total*	*fine*	*ana.fac*	*coarse*	*trans.*		
1 920	$5.4 \cdot 10^9$	$9.2 \cdot 10^4$	Full Rank	15	1169	1166	2.4	0.4	0.1	1.00	$1.9 \cdot 10^{-17}$
			10^{-3}	15	1179	1176	2.7	0.3	0.1	0.99	$3.4 \cdot 10^{-4}$
			10^{-3} + SP	15	1139	1136	2.5	0.3	0.1	1.03	$1.5 \cdot 10^{-3}$
15 360	$4.3 \cdot 10^{10}$	$7.0 \cdot 10^5$	Full Rank	13	1120	1081	36.3	2.8	0.3	0.90	$3.1 \cdot 10^{-18}$
			10^{-3}	13	1118	1092	24.8	1.3	0.2	0.90	$1.4 \cdot 10^{-4}$
			10^{-3} + SP	13	1091	1067	22.3	1.1	0.7	0.93	$2.4 \cdot 10^{-4}$
43 200	$1.2 \cdot 10^{11}$	$2.0 \cdot 10^6$	Full Rank	14	1382	1197	176.2	8.2	0.3	0.79	$1.0 \cdot 10^{-18}$
			10^{-5}	14	1297	1206	87.1	4.0	0.3	0.84	$3.5 \cdot 10^{-7}$
			10^{-5} + SP	14	1282	1194	79.3	3.3	1.0	0.85	$3.6 \cdot 10^{-7}$
			10^{-3}	19	1755	1672	78.4	4.4	0.3	0.84	$1.4 \cdot 10^{-4}$

Up to now, we used the master-worker agglomeration and now turn to the new Superman agglomeration. This technique is an additional contribution compared to the submitted article [30]. We use the same number of processes as before for the fine grid processing, and allocate an additional number of nodes corresponding to the number of processes used solely for MUMPS, i.e. respectively 2, 7, and 10 nodes for the 3 problem sizes. Then, we choose among all allocated nodes the processes dedicated for MUMPS, one per node maximum as before. It is possible to perform simultaneously the first descending part of the fine grid computations and the analysis/factorisation phases of MUMPS on the coarse grid. Based on the results in Table I.4, we simulate the effect of such an agglomeration method in Table I.5. The only difference is in the considered execution time for analysis and factorisation

$$t^{simu}_{ana.\&fac.} = max(t_{ana.\&fac.} - \frac{\overline{t_{fine}}}{2}, 0),$$
$$t^{simu}_{total} = t_{total} - t_{ana.\&fac.} + t^{simu}_{ana.\&fac.}.$$

In the first multigrid cycle, we only consider the analysis and factorisation if they are still not completed after the descending part of the first fine grid computations. Virtually, this agglomeration strategy allows to remove most of the execution times from the analysis and factorisation phases in most cases, when MUMPS is combined with BLR and single precision. With this approach, the parallel efficiency gets to 88% on the largest test

case, compared to 85% with the master-workers agglomeration, and to 79% when using the PMINRES iterative solver. These results are of course estimations, as the fine grid computation may take longer due to the cores dedicated to MUMPS only. The Superman agglomeration will be the subject of future research. A possible improvement is to have cores dedicated to MUMPS only on the analysis and factorisation, i.e. in parallel with the first fine grid processing, then these cores can participate in the fine grid computation during ulterior cycles.

Table I.5 Weak scaling of the multigrid execution with a sparse direct and a block low-rank coarse level solver with single precision (SP) arithmetic. The total execution time (in seconds) and the parallel efficiency considering the average total execution times are displayed for both master-worker and a simulated Superman agglomerations.

DOFs		**BLR** ϵ	**it**	**Master-Worker**			**Superman**		
fine	*coarse*			*proc.*	*total*	*par. eff.*	*proc.*	*total*	*par. eff.*
$5.4 \cdot 10^9$	$9.2 \cdot 10^4$	Full Rank	15	1 920	1169	1.00	1 920 + 40	1167	1.00
		10^{-3} + SP	15		1139	1.03		1137	1.03
$4.3 \cdot 10^{10}$	$7.0 \cdot 10^5$	Full Rank	13	15 360	1120	0.90	15 360 + 160	1084	0.93
		10^{-3} + SP	13		1091	0.93		1069	0.95
$1.2 \cdot 10^{11}$	$1.9 \cdot 10^6$	Full Rank	14	43 200	1382	0.79	43 200 + 225	1339	0.81
		10^{-5} + SP	14		1282	0.85		1235	0.88

This improvement is summarised in Figure I.13, where for each problem size, coarse grid solver and agglomeration technique, we plot the run-time for fine, coarse, analysis+factorisation, and data transfer. As expected, the total run-time stays relatively robust in the weak scaling and we observe that the gain from using MUMPS increases with the size of the problem because the scalability of the proposed solution is better. Also, the run-times for coarse, i.e. MUMPS solve phase, and data transfer are negligible on larger scales.

Concluding remarks

In this chapter, we studied the impact of advanced techniques for the solution of large linear systems, and in particular solvers applied to the coarsest level of a multigrid scheme. To increase the granularity of computations on the coarse grid, we propose the use of an agglomeration technique such that the coarse level solver is executed on only a fraction of the processors intended for the fine grid. By doing so, we significantly reduce the

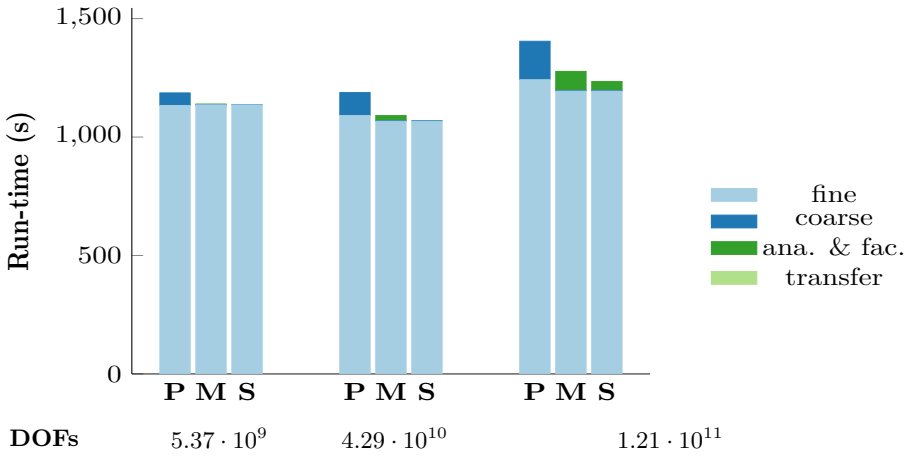


Figure I.13 Comparison of 3 coarse grid solvers in HHG for the three different sizes of problem (I.5): PMINRES (P); MUMPS using BLR and single precision with master-worker agglomeration (M) and Superman agglomeration (S).

time for communication and memory access, thus reducing the overall run-time of the coarse level solver. To increase the speed-up, we employ single precision arithmetic and the block low-rank approximation feature of the MUMPS parallel sparse direct solver that improves compute times at the cost of a reliably controlled loss of accuracy in the solution of the coarsest mesh. The efficiency of the solver is tested on the petascale supercomputer Hazel Hen on up to 43 200 processes and compared to Krylov space solvers for the coarsest grid level. For a 3D Stokes problem, the new solver achieves a 9% points overall parallel efficiency improvement compared to a simple Krylov solver by reducing the run-time of the coarsest level solver. This scalability is achieved thanks to the fact that the factorisation of the coarsest grid matrix need only be computed once and that the factorisation is then re-used in all later V-cycle iterations.

This is an essential improvement e.g. for Earth Mantle simulation scenarios.
This contribution is at the intersection of several domains which we reintroduced: PDE problems and their discretization, direct and iterative linear algebra solvers, efficient hybrid methods for large scale scientific computing. In the next chapters, we introduce other hybrid direct-iterative methods for the solution of very large unsymmetric linear systems based on the block Cimmino iterations.

Chapter II

On numerical solution of full rank linear systems

> Algebra is but written geometry and
> geometry is but figured algebra.
>
> Sophie Germain a.k.a. Antoine
> Auguste Le Blanc

Among iterative methods with a high potential for parallelism, projection methods have received a lot of attention [108]. One interesting feature is that these projection methods can be accelerated with the use of *Conjugate Gradient* (CG) [75] which guarantees convergence to the solution for *Symmetrizable* systems in a finite number of iterations. We extend the block Cimmino iterations [52], accelerated with a stabilised block-CG to solve full rank systems. This iterative method was developed for square unsymmetric problems [11], using a partitioning of the matrix into blocks of rows. An alternative pseudo-direct method was also introduced [47] which guarantees a convergence in 1 iteration through an augmentation of the matrix in a larger space. We here extend these two methods to the minimum norm solution of underdetermined systems and, using a corresponding column partitioning, to the solution of least-squares problems [50].
We start in section II.1 with the introduction of the basic projection techniques Cimmino [34] and Kaczmarz [88]. In section II.2, we then adapt the ideas of the accelerated block Cimmino method [112] applied to underdetermined systems, and least-squares problems. Finally, in section II.3, we introduce the augmented block Cimmino scheme [133] with complete theoretical justification of its properties.
The content of this section is an extension of the work submitted to SIAM's Journal on

Scientific Computing in June 2020 *"On numerical solution of full rank linear systems"*, A. Dumitraşc, P. Leleux, C. Popa, D. Ruiz, & U. Ruede, SIAM's SISC, Copper Mountain Special Issue.

II.1 Block projection methods for full rank systems

Here, we consider the solution to the minimisation problem of the form

$$\min \|x\|_2 \text{ when } x \in \{x; \|b - Ax\|_2 = min!\}. \tag{II.1}$$

where $A \in \mathbb{R}^{m \times n}$ is a *full-rank* matrix, x is vector of size n and b a vector of size m. For simplification, we consider that all matrices are real, although the results can be generalised to complex matrices. The *minimum norm solution* (m.n.s.) to (II.1) (see e.g. [108]) is given by

$$x = A^+ b, \tag{II.2}$$

where A^+ is the Moore-Penrose pseudo inverse of the matrix A.
For any matrix $K \in \mathbb{R}^{m \times n}$, if K is full row rank ($rank(K) = m$) then $K^+ = K^T(KK^T)^{-1}$, and if K is full column rank ($rank(K) = n$) then $K^+ = (K^TK)^{-1}K^T$. We also define the orthogonal projections on $\mathcal{R}(K)$ and $\mathcal{R}(K^T)$, the *range* of K and K^T, as

$$\mathcal{P}_{\mathcal{R}(K^T)} = K^+K \text{ and } \mathcal{P}_{\mathcal{R}(K)} = KK^+, \tag{II.3}$$

as well as the projectors on $\mathcal{N}(K)$ and $\mathcal{N}(K^T)$, the *nullspace* of K and K^T, as

$$\mathcal{P}_{\mathcal{N}(K^T)} = I_m - \mathcal{P}_{\mathcal{R}(K)} \text{ and } \mathcal{P}_{\mathcal{N}(K)} = I_n - \mathcal{P}_{\mathcal{R}(K^T)}. \tag{II.4}$$

II.1.1 The Cimmino and Kaczmarz methods

We assume that we have a partitioning of the matrix A into p blocks. For a full row rank matrix, we partition the matrix into row blocks A_i, and b is partitioned accordingly

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_p \end{bmatrix}, \text{ and } b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix}, \tag{II.5}$$

with $A_1, \ldots, A_p$ row blocks from the partitioning $A_i \in \mathbb{R}^{m_i \times n}$, $m_1 + \cdots + m_p = m$. For a full column rank matrix, we partition the matrix into column blocks A^i, and x is

partitioned accordingly

$$A = \begin{bmatrix} A^1 & A^2 & \dots & A^p \end{bmatrix}, \text{ and } x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}, \tag{II.6}$$

with $A^1, \dots, A^p$ column blocks from the partitioning $A^i \in \mathbb{R}^{m \times n_i}$, $n_1 + \dots + n_p = n$.

Block projection methods are iterative algorithms which compute the solution of the linear system (II.1) through successive projections of the current iterate $x^{(k)}$ onto the range of the partition blocks [52]. We distinguish 2 approaches: the multiplicative methods in which an iterate is obtained with a product of projections, and the additive methods in which the iterate is computed with a sum of projections. The 2 approaches are respectively related to the application of *Successive Over-Relaxation* (SOR), and *block Jacobi* (BJ) on the normal equations of the matrix, in the sense that the iteration matrices are equivalent. When the partitions are reduced to single rows, these methods are respectively called the *Kaczmarz* algorithm [88], and the *Cimmino* algorithm [34]. The row and column *block Kaczmarz* (BK) and *block Cimmino* (BC) methods are then generalisations with row or column partitions of arbitrary sizes [52].

Let's consider that A is underdetermined ($m \leq n$) and has full row rank ($rank(A) = m$). The starting vector $x^{(0)}$ is arbitrary. In the *row block Cimmino* method, we obtain the next iterate $x^{(k+1)}$ through a sum of projections of the current iterate $x^{(k)}$ onto the range of A_i^T, $i = 1, \dots, p$. Let's note ω a relaxation parameter, and $\delta^{(i)}$ the projection on $\mathcal{R}(A_i^T)$. We get the iteration

$$\begin{aligned} \delta^{(i)} &= A_i^+(b_i - A_i x^{(k)}) = A_i^+ b_i - \mathcal{P}_{\mathcal{R}(A_i^T)}(x^{(k)}),\ i = 1, \dots, p, \\ x^{(k+1)} &= x^{(k)} + \omega \sum_{i=1}^{p} \delta^{(i)}. \end{aligned} \tag{II.7}$$

In the row block Kaczmarz method, on the contrary, the new projections are used as soon as available

$$\begin{aligned} \delta^{(1)} &= x^{(k)}, \\ \delta^{(i+1)} &= \delta^{(i)} + \omega A_i^+(b_i - A_i \delta^{(i)}) = \delta^{(i)} + \omega A_i^+ b_i - \mathcal{P}_{\mathcal{R}(A_i^T)}(\delta^{(i)}),\ i = 1, \dots, p \\ x^{(k+1)} &= \delta^{(p+1)}. \end{aligned}$$

We illustrate both methods in Figure II.1 on a 2D case, where each equation corresponds to a line in space and finding the solution is finding the intersection between those spaces.

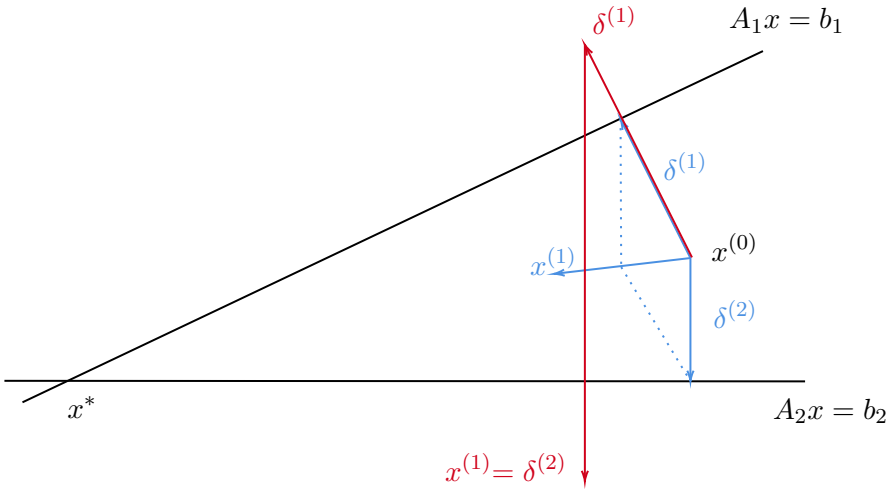


Figure II.1 2D visualisation of an iteration for *(Blue)* the BC method, and *(Red)* the block Kaczmarz method, applied to an initial iterate $x^{(0)}$.

Similarly, we consider now that A is overdetermined ($m \geq n$) and has full column rank ($rank(A) = n$), with A partitioned into column blocks A^i, $i = 1, \ldots, p$. The starting vector $x^{(0)}$ is arbitrary. Let's denote by ω a relaxation parameter, $\delta^{(i)}$ the projection, and $r^{(k,i)}$ the residual corresponding to partition i. The *column block Cimmino* method updates each part of the current iterate separately. Starting from an initial residual $r^{(0)} = b - Ax^{(0)}$, the iteration reads

$$\begin{aligned} \delta^{(i)} &= A^i A^{i^+} r^{(k)} = \mathcal{P}_{\mathcal{R}(A^i)}(r^{(k)}), \\ x_i^{(k+1)} &= x_i^{(k)} + \omega A^{i^+} r^{(k)}, \\ r^{(k+1)} &= r^{(k)} - \omega \sum_{i=1}^{p} \delta^{(i)}, \ i = 1, \ldots, p, \end{aligned} \tag{II.8}$$

The column block Kaczmarz method counterpart is given by

$$\begin{aligned} \delta^{(k,1)} &= r^{(k)} = b - Ax^{(k)}, \\ \delta^{(k,i+1)} &= \delta^{(k,i)} - \omega A^i A^{i^+} \delta^{(k,i)} = \delta^{(k,i)} - \omega \mathcal{P}_{\mathcal{R}(A^i)}(\delta^{(k,i)}), \\ x_i^{(k+1)} &= x_i^{(k)} + \omega A^{i^+} \delta^{(k,i)}, \ i = 1, \ldots, p. \end{aligned}$$

The Cimmino and Kaczmarz methods are very similar. We consider primarily the block

Cimmino method for parallel computing reasons: at each iteration, the projections are computed independently, thus unveiling a large potential for parallelism.

II.1.2 The block Cimmino method

In this section, we detail the block Cimmino method applied to full rank matrices, and its interpretation. But first, in an attempt to unify the approaches based on row and column partitioning, we are interested in the relation between the solution of an underdetermined system and a least-squares problem for special choices of right hand side.

Property 1. *Assuming the matrix $A \in \mathbb{R}^{m\times n}$ is underdetermined ($m \leq n$), has full row rank, and is partitioned into blocks of rows A_i as in* (II.5). *$A^T \in \mathbb{R}^{n\times m}$ is overdetermined ($m \geq n$), has full column rank, and is partitioned into blocks of columns A^i as in* (II.6), *with $\mathbf{A^i = A_i^T}$.*
Then, considering the vectors b and $\widetilde{b}$ s.t. $\mathbf{b = A\widetilde{b}}$, the unique solution x_{ls} of the least-squares *problem*

$$\min_{\widetilde{x}} \left\| \widetilde{b} - A^T \widetilde{x} \right\|_2, \tag{II.9}$$

and the minimum norm solution *(m.n.s.) x_{mns} of the problem*

$$\min \|x\|_2 \ \ \textit{with } Ax = b, \tag{II.10}$$

are linked by the relation $x_{mns} = A^T x_{ls}$.

Proof Considering we have $b = A\widetilde{b}$, the proof is simple.

x_{ls} is the unique solution of the normal equations $AA^T\widetilde{x} = A\widetilde{b}$.
x_{mns} is the unique solution of the system

$$\begin{aligned} &\begin{cases} AA^T y &= b = A\widetilde{b}, \\ x &= A^T y, \end{cases} \\ \Longleftrightarrow &\begin{cases} y &= \widetilde{x}, \\ x &= A^T \widetilde{x}, \end{cases} \end{aligned} \tag{II.11}$$

which completes the proof. The 2-steps process in (II.11) is implicit in the row block Cimmino method, which can also be seen as a *ω-damped block Jacobi* method (ω-BJ) applied to the normal equations, where the diagonal blocks correspond to the row partitions [52]. As we apply the same ω-BJ iterations to solve both (II.9) and (II.10), the iterates $x^{(k+1)}$ and $\widetilde{x}^{(k+1)}$ from row and column block Cimmino are linked by $x^{(k+1)} = A^T\widetilde{x}^{(k+1)}$ whenever $b = A\widetilde{b}$.

To find $\widetilde{b}$ such that $b = A\widetilde{b}$ is to find a solution to the system without considering the minimum norm constraint. This is not the goal. The previous property is only introduced to greatly simplify further developments by writing $b = A\widetilde{b}$ and always working with an underdetermined matrix A. In the rest of the thesis, we solve the underdetermined system (II.10), based on matrix A and b, using the row block Cimmino method, or the least-squares problem (II.9), based on the matrix A^T and $\widetilde{b}$, using the column block Cimmino method, with $b = A\widetilde{b}$.

II.1.2.a The iteration matrices

We introduce the block diagonal matrix

$$\begin{aligned} D &= blkdiag(D_1, \ldots, D_p), \\ D_i &= A_i A_i^T,\ i = 1, \ldots, p. \end{aligned}$$

As detailed in [47], we rewrite the iteration of the row block Cimmino method (II.7) as

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \omega \sum_{i=1}^{p} {A_i}^+ (b_i - A_i x^{(k)}),\ i = 1, \ldots, p \\ &= Q_\omega^{row} x^{(k)} + \omega k^{row}, \end{aligned} \tag{II.12}$$

with

$$\begin{cases} Q_\omega^{row} &= I_n - \omega H^{row} \text{ is the iteration matrix,} \\ H^{row} &= \sum\limits_{i=1}^{p} A_i^+ A_i = A^T D^{-1} A, \\ k^{row} &= \sum\limits_{i=1}^{p} A_i^+ b_i = A^T D^{-1} b. \end{cases} \tag{II.13}$$

Using previous notation, we also rewrite the iteration of the column block Cimmino (II.8) as

$$\begin{aligned} \widetilde{x}_i^{(k+1)} &= \widetilde{x}_i^{(k)} + \omega {A^i}^+ (\widetilde{b} - A^T \widetilde{x}^{(k)}),\ i = 1, \ldots, p \\ \iff \widetilde{x}^{(k+1)} &= Q_\omega^{col} \widetilde{x}^{(k)} + \omega k^{col}, \end{aligned} \tag{II.14}$$

with

$$\begin{cases} Q_\omega^{col} &= I_m - \omega H^{col} \text{ is the iteration matrix,} \\ H^{col} &= D^{-1} A A^T, \\ k^{col} &= D^{-1} A \widetilde{b}. \end{cases} \tag{II.15}$$

As H^{row} and H^{col} share several properties, in the following H is used to refer to

either matrices. The same kind of notation is used for Q and k.

It is shown in [52] that both row and column block Cimmino methods converge if and only if $0 < \omega < 2/\rho(H)$. The remaining question is how to compute the various pseudo-inverses composing H and k.

II.1.2.b Computing the projections

As detailed in [133], there are several approaches to compute these pseudo-inverses.

- *Normal equations:* The normal equations of each partition $A_i A_i^T$ are directly factorised. However, additionally to being expensive in flops and memory because of fill-in effects, this approach can also be numerically unstable because the conditioning of the original partition A_i is squared.
- *QR factorisation:* another alternative is to use the QR factorisation $A_i^T = Q_i R_i$, so that $A_i^+ = Q_i R_i^{-T}$ and $A_i^+ A_i = Q_i Q_i^T$.
- *Semi-normal equations:* using the same QR factorisation, $A_i^+ v$ is computed in 2 steps. Solving $R_i R_i^T w = v$ is a simple elimination, cheap and stable as R is upper triangular. Then we directly obtain $A_i^+ v = A_i^T w$. The issue is that QR factorisation is expensive compared to Gaussian elimination.
- *Augmented systems:* using the partitions A_i, the augmented system [19]

$$\begin{bmatrix} I_n & A_i^T \\ A_i & 0_{m_i} \end{bmatrix} \begin{bmatrix} u_i \\ v_i \end{bmatrix} = \begin{bmatrix} r \\ z_i \end{bmatrix}, \tag{II.16}$$

  is solved with $r \in \mathbb{R}^{n\times 1}$ and $z_i \in \mathbb{R}^{m_i \times 1}$, with m_i the size of the partition A_i. We obtain the solutions

$$\begin{aligned} v_i &= (A_i A_i^T)^{-1}(A_i r - z_i), \\ u_i &= r - A_i^T (A_i A_i^T)^{-1}(A_i r - z_i). \end{aligned}$$

  Solving (II.16) at each iteration, with one independent system for each partition, we obtain the required elements for the block Cimmino method.

The best option in our case is to use the augmented systems, that we call *projection systems*, and solve these systems with a parallel direct solver. This approach was proven to be more stable than the normal equations in [10].

II.1.2.c Block Cimmino as a domain decomposition method

In this section, we highlight the interpretation of BC as a *Domain Decomposition Method* (DDM) As stated above, the block Cimmino iterations (row or column methods) are

equivalent to applying a damped block-Jacobi algorithm to the normal equations of the system [75], where the size of the diagonal blocks corresponds to the size of the corresponding partition. The damped-Jacobi iterations are shown to be a form of additive Schwarz method with minimum overlap, as introduced in I.2.2, applied on the normal equations system. A proof is given in Appendix A.1 inspired from Section 1.2 of [40]. This interpretation gives a wider interpretation on how systems are solved with block Cimmino iterations. Also, this naturally points to the solution of discretized PDEs, together with domain decomposition. As a matter of fact, the partitioning of the matrix can be performed using the geometry of the PDE problem, with the usual purpose of DDM to balance the workload inside partitions – subdomains – while minimising the links between them – interfaces. We develop in more details these issues with respect to partitioning techniques in Section III.1.2.

II.2 Conjugate gradient acceleration for the block Cimmino method

The convergence of the projection methods introduced above is known to be slow, even with an optimal choice for the relaxation parameter ω [52]. From (II.12) and (II.14), the iteration of the block Cimmino method can be written in the form

$$x^{(k+1)} = Qx^{(k)} + k, \tag{II.17}$$

where the iteration matrix $Q = I - \omega H$, H and k depends on the row or column version. Equation (II.17) is the classical form of a fixed-point iterative scheme such as Jacobi or SOR, as seen in Section I.2.1. In [75], polynomial accelerations using either *Chebyshev polynomials* or the *Conjugate Gradient* algorithm (CG) were introduced to improve the convergence of these *symmetrisable* iterative schemes. Among these 2, CG acceleration is particularly interesting because it does not require any parameter estimation. On the opposite, Chebyshev polynomials require an estimation of the largest and smallest eigenvalues of the iteration matrix, but will not involve any dot-products.
The CG algorithm is known to converge in a finite number of steps, in the absence of round-off errors, when applied to *symmetric positive definite* (SPD) matrices [79]. For this kind of matrices, CG is the iterative scheme usually chosen. In particular, CG acceleration was studied for the Cimmino and Kaczmarz methods in [22]. The authors showed the potential of such accelerated methods to solve large unsymmetric linear systems, that exhibit a nice parallelism and high robustness. In this context, the CG

acceleration of symmetrisable schemes [75] was observed to converge faster than other classical iterative methods designed for unsymmetric systems, such as e.g. the *Generalised Minimal Residual method* (GMRES) or the *Conjugate Gradient method on the Normal Equations* (CGNE).

In this section, we recall the details relative to the Conjugate Gradient acceleration [75] of the block Cimmino method [112]. Looking at the fixed point of the iterations x^*, we rewrite the system as

$$\begin{aligned} x^* &= (I - \omega H)x^* + \omega k \\ \iff \omega H x^* &= \omega k \\ \iff H x^* &= k \ (\omega \neq 0). \end{aligned}$$

where $(I - \omega H)$ is the iteration matrix of the block Cimmino iterations. This equation is independent from the relaxation parameter ($\omega \neq 0$). We end up with the new system to solve

$$H x^* = k. \tag{II.18}$$

As an abuse of language in the following, we also call H the iteration matrix of the block Cimmino method. This change of system can be viewed as a left preconditioning of the original system inspired from *Cimmino*.

II.2.1 Solving underdetermined systems

In this section, we remind the results detailed in [112] and [133] for square full rank systems. We also prove that these results are directly applicable to the *minimum norm solution* (m.n.s.) of full rank underdetermined systems, and which can thus be solved using the accelerated row block Cimmino method.

Assuming we are computing the m.n.s. of the underdetermined system (II.10). With the row-partitioned block Cimmino method, the iteration matrix H^{row}, defined in (II.13) is a sum of orthogonal projections, and is *symmetric semi positive definite* (SPD). Also, the system (II.18) is consistent, and we can then use a CG to solve the equation (II.18), thus accelerating the row block Cimmino method as described in [75].

II.2.1.a Stabilised block-CG

The convergence of the CG acceleration can still be slow depending on the distribution of the eigenvalues of the matrix H^{row}. In particular, if the spectrum of H^{row} contains clusters

of small eigenvalues together with some ill-conditioning, we may observe long plateaus in the convergence. To reduce these plateaus, it was proposed in [68] to combine the CG acceleration with Chebyshev filtering to target directly these clusters of eigenvalues. Such filters are of particular interest in the case of the block Cimmino iteration matrix H^{row}, whose eigenvalues are clustered around 1, and small clusters of eigenvalues remain at the extreme of the spectrum. The authors showed that this approach can be beneficial when solving linear systems with multiple right-hand sides.
In [11] and [112], *a block version of the CG acceleration* (block-CG) is used to reduce the long plateaus in the convergence. The right hand side b is then augmented as $B \in \mathbb{R}^{m \times s}$, using additional random vectors. Using the block-CG we can also naturally solve with multiple right-hand sides. The algorithm is additionally stabilised through a re-orthonormalization of the residual and conjugate directions to avoid numerical issues at superlinear convergence. The complete algorithm is detailed in Algo. 1, which we shall simply call *row-BC* in the following.

Algorithm 1 Stabilised block-CG acceleration of the row block Cimmino iterations (row-BC).

Input: $H^{row} \in \mathbb{R}^{n \times n}$, $K^{row} \in \mathbb{R}^{m \times s}$.
Output: solution $X \in \mathbb{R}^{n \times s}$.
1: $X^{(0)}$ is arbitrary, $R^{(0)} = K - H^{row}X^{(0)}$,
2: $\overline{R}^{(0)} = R^{(0)}\gamma_0^{-1}$ such that ${\overline{R}^{(0)}}^T \overline{R}^{(0)} = I_s$,
3: $\overline{P}^{(0)} = \overline{R}^{(0)}\beta_0^{-1}$ such that ${\overline{P}^{(0)}}^T H^{row} \overline{P}^{(0)} = I_s$,
4: **for** $j = 0, 1, 2, \ldots$ until convergence **do**
5: $\quad \lambda_j = \beta_j^{-T}$,
6: $\quad X^{(j+1)} = X^{(j)} + \overline{P}^{(j)} \lambda_j \left(\prod_{i=j}^{0} \gamma_i \right)$,
7: $\quad \overline{R}^{(j+1)} = (\overline{R}^{(j)} - H^{row}\overline{P}^{(j)}\lambda_j)\gamma_{j+1}^{-1}$ such that ${\overline{R}^{(j+1)}}^T \overline{R}^{(j+1)} = I_s$,
8: $\quad \alpha_j = \beta_j \gamma_{j+1}^T$,
9: $\quad \overline{P}^{(j+1)} = (\overline{R}^{(j+1)} + \overline{P}^{(j)}\alpha_j)\beta_{j+1}^{-1}$ such that ${\overline{P}^{(j+1)}}^T H^{row} \overline{P}^{(j+1)} = I_s$.
10: **end for**

II.2.1.b The stabilisation process

Through the iterations, stabilisation is necessary in the block-CG because the residuals $R^{(j)}$ and $R^{(j+1)}$ are naturally maintained orthonormal block-wise, i.e. ${R^{(j)}}^T R^{(j+1)} = 0$, but the column-vectors inside the blocks are not. As the algorithm converges, some vectors in the residual block get close to colinear. As a result, $R^{(j)}$ is asymptotically

ill-conditioned, and need to be orthonormalized. This orthonormalisation takes the form of an upper triangular matrix γ_j built such that $\overline{R}^{(j)^T}\overline{R}^{(j)} = I_s$ with $\overline{R}^{(j)} = R^{(j)}\gamma_j^{-1}$. For the same reasons, an upper triangular matrix β_j is constructed at each iteration in order to H^{row}-orthonormalize the conjugate directions within each block $P^{(j)}$. We obtain $\overline{P}^{(j)} = P^{(j)}\beta_j^{-1}$ such that $\overline{P}^{(j)^T} H^{row}\overline{P}^{(j)} = I_s$.

There are several choices to compute the orthonormalization matrices. In [112], the authors proposed to orthonormalize the residuals using a Cholesky decomposition of $R^{(j)^T}R^{(j)}$ and $P^{(j)^T}H^{row}P^{(j)}$. The same authors show that for large block sizes, the stability of this process may deteriorate. Again, this instability is due to an increasing ill-conditioning of the residuals when reaching superlinear convergence. An alternative was is to use an orthonormalization process based on a *modified Gram-Schmidt* (GMGS) [20], or a *generalised QR* (GQR) [69]. We choose to use the latter. In any case, the main complexity of the stabilisation process comes from a dense matrix operations on a matrix of dimension s. Finally, some of the solution vectors may converge earlier than others in the block-CG, then we could reduce the block-size along the block-CG iterations [106].

II.2.1.c Computation of the projections

In Algo. 1, H^{row} is never explicitly assembled, we directly apply a sum of independent projections to the set of s vectors. The algorithm to compute the sum of projections, *sumProject*, is detailed in Algo. 2. For each partition i, the projection $A_i^+A_i$ is computed independently via the solution of the system (II.16) by a parallel direct solver, see Algo. 3. The independence between the computation of each projection is a key stone to the hybrid parallelization scheme [133] of the block Cimmino method accelerated with the stabilised block-CG, see Section III.2.1. This function takes 2 arguments $r \in \mathbb{R}^{\ n}$ and $z_i \in \mathbb{R}^{\ m_i}$, thus in the block-CG algorithm, we compute the projections as

$$\begin{aligned} R^{(0)} &= -sumProject(X^{(0)}, B), \\ HP^{(j)} &= sumProject(P^{(j)}, 0). \end{aligned}$$

The stabilised block-CG algorithm written in the form of Algo. 1 has several advantages in terms of computational complexity as the resulting block iterative algorithm benefits from efficient dense matrix operations together with the reduction of plateaus in the convergence. We detail these effects with experiments in Section II.4.1.

Algorithm 2 Sum of Projections (*sumProject*).

Input: r and z.

Output: $\sigma = \sum_{i=1}^{p} A_i^+(z_i - A_i r)$.

1: $\sigma = \begin{bmatrix} 0 & \dots & 0 \end{bmatrix}^T$,
2: **for** $i = 1 \to p$ **do**
3: $z_i = z$ restricted to the rows of partition i,
4: $\begin{bmatrix} u_i \\ v_i \end{bmatrix} = solve\_A_i(r,\ z_i)$,
5: $\sigma = \sigma + u_i$.
6: **end for**

Algorithm 3 Solution of the projection system for A_i (*solve_*A_i).

Input: r and z_i.

Output: $u_i = A_i^+(A_i r - z_i)$ and $v_i = (A_i A_i^T)^{-1}(A_i r - z_i)$.

1: $\begin{bmatrix} u_i \\ v_i \end{bmatrix} = \text{direct\_solve}(\begin{pmatrix} I_n & A_i^T \\ A_i & O_{m_i} \end{pmatrix} \begin{pmatrix} u_i \\ v_i \end{pmatrix} = \begin{pmatrix} r \\ z_i \end{pmatrix}$,
2: $u_i = r - u_i$.

II.2.2 Solving least-squares problems

We now extend the previous approach to the solution of the full column rank least squares problem (II.9) with the column block Cimmino method, accelerated with a block-CG. Thanks to the equivalence shown in Section II.1.2, most of the elements used to accelerate the row block Cimmino stay applicable for the column block Cimmino.

As opposed to the case with the row block approach, matrix H^{col} from (II.15) is not symmetric in itself, but since $D^{\frac{1}{2}} H^{col} D^{-\frac{1}{2}} = D^{-\frac{1}{2}} A A^T D^{-\frac{1}{2}}$, it is similar to an SPD matrix given A has full row rank. To develop a *block Conjugate Gradient* (block-CG) acceleration of the column block Cimmino iterations (II.8), we consider the solution of the equivalent linear system

$$My = f, \tag{II.19}$$

in which $M = D^{-\frac{1}{2}} A A^T D^{-\frac{1}{2}}$, $f = D^{-\frac{1}{2}} A \widetilde{B}$, and $y = D^{\frac{1}{2}} \widetilde{x}$ is a change of variables in (II.18). $\widetilde{B}$ is the right-hand side $\widetilde{b}$ completed to the block size s with additional random columns. Since matrix M is SPD, provided A has full row rank, we use the stabilised block-CG algorithm to solve this latter system. We recall the basics from [11] applied to (II.19), and summarised in Algo. 4 below. In this algorithm, matrices γ_j and β_j are upper triangular, and result from an orthonormalization process, which is managed as before with GQR for instance.

Algorithm 4 Stabilised Block Conjugate Gradient to solve $MY = F$.

Input: $M \in \mathbb{R}^{m\times m}$, $F \in \mathbb{R}^{m\times s}$.
Output: solution $Y \in \mathbb{R}^{n\times s}$.

1: $Y^{(0)}$ is arbitrary, $Z^{(0)} = F - MY^{(0)}$,
2: $\overline{Z}^{(0)} = Z^{(0)}\gamma_0^{-1}$ such that $(\overline{Z}^{(0)^T}\overline{Z}^{(0)}) = I_s$,
3: $\overline{W}^{(0)} = \overline{Z}^{(0)}\beta_0^{-1}$ such that $(\overline{W}^{(0)^T} M\overline{W}^{(0)}) = I_s$,
4: **for** $j = 0, 1, 2, \ldots$ until convergence **do**
5: $\quad\lambda_j = \beta_j^{-T}$,
6: $\quad Y^{(j+1)} = Y^{(j)} + \overline{W}^{(j)}\lambda_j \left(\prod_{i=j}^{0}\gamma_i\right)$,
7: $\quad\overline{Z}^{(j+1)} = \left(\overline{Z}^{(j)} - M\overline{W}^{(j)}\lambda_j\right)\gamma_{j+1}^{-1}$ such that $(\overline{Z}^{(j+1)^T}\overline{Z}^{(j+1)}) = I_s$,
8: $\quad\alpha_j = \beta_j\gamma_{j+1}^T$,
9: $\quad\overline{W}^{(j+1)} = \left(\overline{Z}^{(j+1)} + \overline{W}^{(j)}\alpha_j\right)\beta_{j+1}^{-1}$ such that $(\overline{W}^{(j+1)^T} M\overline{W}^{(j+1)}) = I_s$,
10: **end for**

The derivation, from Algo. 4, of the block-CG acceleration of the column block Cimmino, is rather straightforward. We introduce the starting guess $\widetilde{X}^{(0)}$, so that $Y^{(0)} = D^{\frac{1}{2}}\widetilde{X}^{(0)}$, and the associated residual $R^{(0)} = \widetilde{B} - A^T\widetilde{X}^{(0)}$, for which we get

$$Z^{(0)} = F - MY^{(0)} = D^{-\frac{1}{2}}AR^{(0)}.$$

From step 2 in Algo. 4, we then introduce $\overline{R}^{(0)} = R^{(0)}\gamma_0^{-1}$, so that

$$\overline{R}^{(0)^T}(A^TD^{-1}A)\overline{R}^{(0)} = \overline{Z}^{(0)^T}\overline{Z}^{(0)} = I.$$

The matrix γ_0 enforces the orthonormality of the residuals $R^{(0)}$ with respect to the inner product associated with matrix $A^TD^{-1}A$. Note also, that this matrix corresponds to the sum of the projectors onto the subspaces spanned by the partitions, and thus to H^{row}, the iteration matrix from the row block Cimmino in (II.13). In step 3, we set the upper triangular matrix β_0 so that $\beta_0^{-T}\overline{Z}^{(0)^T}M\overline{Z}^{(0)}\beta_0^{-1} = I$, which is also equivalent to

$$\beta_0^{-T}\overline{R}^{(0)^T}(H^{row})^2\overline{R}^{(0)}\beta_0^{-1} = I.$$

Therefore, we introduce the new set of vectors $\overline{Q}^{(0)} = H^{row}\overline{R}^{(0)}\beta_0^{-1}$, for which we then have $\overline{Q}^{(0)^T}\overline{Q}^{(0)} = I$, and matrix β_0 is built in order to orthonormalize the set of vectors in $H^{row}\overline{R}^{(0)}$. We then verify the propagation of this initial setting, by recurrence, and

exchange step 7 with

$$\overline{R}^{(j+1)} = \left(\overline{R}^{(j)} - \overline{Q}^{(j)}\lambda_j\right)\gamma_{j+1}^{-1} \text{ such that } (\overline{R}^{(j+1)^T} H^{row} \overline{R}^{(j+1)}) = I,$$

together with step 9 replaced by

$$\overline{Q}^{(j+1)} = \left(H^{row}\overline{R}^{(j+1)} + \overline{Q}^{(j)}\alpha_j\right)\beta_{j+1}^{-1} \text{ such that } (\overline{Q}^{(j+1)^T} \overline{Q}^{(j+1)}) = I.$$

Strictly speaking, the iterations are equivalent, yielding the same iterates α_j, β_j, γ_j, and λ_j, except that we update different vectors, for which we have the relations $\overline{Z}^{(j)} = D^{-\frac{1}{2}} A \overline{R}^{(j)}$ and $\overline{Q}^{(j)} = A^T D^{-\frac{1}{2}} \overline{W}^{(j)}$, together with an iteration matrix H^{row} that corresponds to the sum of projections defined by the partitioning of A. The final step is to recover the iterates $\widetilde{X}^{(j)} = D^{-\frac{1}{2}} Y^{(j)}$. From step 6 in Algo. 4, we maintain iteratively the new updates $\overline{P}^{(j)} = D^{-\frac{1}{2}} \overline{W}^{(j)}$. This is resumed in steps 4 and 11 in Algo. 5.

All in all, we obtain the following Stabilised Block-CG acceleration of the column-block Cimmino method, detailed in Algo. 5, which we simply call *column-BC* in the following. In this algorithm, the set of vectors $\overline{Q}^{(j)}$ and $\overline{P}^{(j)}$ depend respectively on

Algorithm 5 Stabilised Block Conjugate Gradient acceleration of the column-BC.

Input: $A^T \in \mathbb{R}^{n\times m}$, $\widetilde{B} \in \mathbb{R}^{n\times s}$, $H^{row} \in \mathbb{R}^{n\times n}$.
Output: solution $\widetilde{X} \in \mathbb{R}^{m\times s}$.

1: $\widetilde{X}^{(0)}$ is arbitrary, $R^{(0)} = \widetilde{B} - A^T \widetilde{X}^{(0)}$,
2: $\overline{R}^{(0)} = R^{(0)}\gamma_0^{-1}$ such that $(\overline{R}^{(0)^T} H^{row} \overline{R}^{(0)}) = I_s$,
3: $\overline{Q}^{(0)} = H^{row}\overline{R}^{(0)}\beta_0^{-1}$ such that $(\overline{Q}^{(0)^T} \overline{Q}^{(0)}) = I_s$,
4: $\overline{P}^{(0)} = D^{-1}A\overline{R}^{(0)}\beta_0^{-1}$,
5: **for** $j = 0, 1, 2, \ldots$ until convergence **do**
6: $\quad \lambda_j = \beta_j^{-T}$,
7: $\quad \widetilde{X}^{(j+1)} = \widetilde{X}^{(j)} + \overline{P}^{(j)}\lambda_j(\prod_{i=j}^{0}\gamma_i)$,
8: $\quad \overline{R}^{(j+1)} = (\overline{R}^{(j)} - \overline{Q}^{(j)}\lambda_j)\gamma_{j+1}^{-1}$ such that $(\overline{R}^{(j+1)^T} H^{row} \overline{R}^{(j+1)}) = I_s$,
9: $\quad \alpha_j = \beta_j \gamma_{j+1}^T$,
10: $\quad \overline{Q}^{(j+1)} = (H^{row}\overline{R}^{(j+1)} + \overline{Q}^{(j)}\alpha_j)\beta_j^{-1}$ such that $(\overline{Q}^{(j+1)^T} \overline{Q}^{(j+1)}) = I_s$,
11: $\quad \overline{P}^{(j+1)} = (D^{-1}A\overline{R}^{(j+1)} + \overline{P}^{(j)}\alpha_j)\beta_{j+1}^{-1}$.
12: **end for**

$H^{row}\overline{R}^{(j)}$ and $D^{-1}A\overline{R}^{(j)}$. The first results from a sum of projection and is computed using $sumProject(R^{(j)}, 0)$. The second element is a concatenation of pseudo-inverse solutions, which can be computed together with a direct solver again from the projection

systems (II.16), using $solve\_A_i$ (Algo. 3). Steps 4 and 11 are computed as

$$D^{-1}A\overline{R}^{(j)} = gatherProject(R^{(j)}, 0),$$

with the corresponding Algo. given in Algo. 6. In practice, the results from *sumPro-*

Algorithm 6 Gather Projections ($gatherProject$).

Input: r and z.
Output: $\gamma = D^{-1}(Ar - z)$.
1: $\gamma = \begin{bmatrix} 0 & \dots & 0 \end{bmatrix}^T$,
2: **for** $i = 1 \to p$ **do**
3: $z_i = z$ restricted to the rows of partition i,
4: $\begin{bmatrix} u_i \\ v_i \end{bmatrix} = solve\_A_i(r,\ z_i)$,
5: $\gamma_i = v_i$.
6: **end for**

ject and *gatherProject* are obtained with a single solution of the projection system for each partition. Doing so, we spare half of the calls to the direct solver to compute the projections, which is the most expensive part of the algorithm. The computational advantages of Algo. 5 are the same as for Algo. 1. Again, the independence between these subsystems is a keystone for the hybrid parallel implementation of the column-BC method accelerated with a stabilised block-PCG. We detail this parallelisation in the Section III.2.1, together with implementation details for the row and column methods as they are implemented in the ABCD-Solver [1] library.

Summary We have developed an acceleration of the block Cimmino method with a stabilised block-CG, designed both for the m.n.s. of underdetermined and the solution of overdetermined linear systems. Henceforth, in this thesis, we simply refer to the accelerated methods as row-BC and column-BC as opposed to the classical iterations. In section II.4, we show the sequential efficiency of the method for some sparse matrices and in particular the effect of increasing the block size within the block-CG acceleration. As explained in the forthcoming Section III.1.2, which details the construction of the partitions, the spectrum of the iteration matrix (row and column) is affected by the principal angles between the subspaces spanned by the partitions, and this is problem dependent. The convergence behaviour is thus difficult to predict in general with a convergence profile displaying either long plateaus or linear convergence. To overcome

[1] `http://abcd.enseeiht.fr/`

these issues with the convergence of such iterative methods, we propose in the next section an alternative approach that can achieve convergence in 1 iteration.

II.3 The augmented block Cimmino method

We introduce now an alternative approach proposed in [47]. The principle of the *Augmented block Cimmino* method (ABCD) is to augment the matrix to enforce the orthogonality between subspaces spanned by the partition blocks, so as to achieve a convergence of the block Cimmino method in 1 iteration. The result is a pseudo-direct method in which the construction and solution of a smaller symmetric positive system is central. In this section, we extend the approach, originally designed for full rank square unsymmetric systems, to full rank rectangular systems.

II.3.1 Enforcing the orthogonality between spaces

We consider again that we have a full rank underdetermined matrix $A \in \mathbb{R}^{m \times n}$, partitioned in blocks of rows as in (II.5), and we are computing either the m.n.s. of the underdetermined system (II.10), based on A and b, or the solution for the least-squares problem (II.9), based on A^T and $\widetilde{b}$ together with $b = A\widetilde{b}$. Most of the results for the augmentation process are identical for both kind of systems, and we thus do not make any difference at first. We shall just detail how to compute the final solution for each case.

For a better understanding of the augmentation process, we use the example of a block-tridiagonal matrix of the form

$$\begin{array}{c} A_1 \\ A_2 \\ A_3 \end{array} \begin{bmatrix} A_{1,1} & A_{1,2} & & & \\ & A_{2,1} & A_{2,2} & A_{2,3} & \\ & & & A_{3,2} & A_{3,3} \end{bmatrix}. \tag{II.20}$$

Such a form is not mandatory but simplifies the illustration below, and the approach is valid for any general underdetermined matrix. In this block tridiagonal form, the scalar product of 2 partitions is reduced to

$$\forall\ i, j = 1, \ldots, p,\ i \neq j,\ A_i^T A_j = \begin{cases} 0, & \text{if } j \neq i \pm 1 \\ A_{i,j}^T A_{j,i}, & \text{if } j = i \pm 1 \end{cases},$$

where $A_{i,j}$ and $A_{j,i}$ are the submatrices within A_i and A_j respectively corresponding to

the overlapping columns between these two blocks.

II.3.1.a The augmentation scheme

The essence of the method is to enforce the numerical orthogonality between partitions with the help of additional variables and constraints in the system. First, we augment the matrix A into $\overline{A}$, viz.

$$\overline{A} = \begin{bmatrix} A & C \end{bmatrix}, \tag{II.21}$$

with $C \in \mathbb{R}^{m \times q}$, and $\overline{A} \in \mathbb{R}^{m \times \overline{n}}$, $\overline{n} = n + q$. This augmentation block is built so that the augmented partitions $\overline{A}_i$ are mutually orthogonal, i.e.

$$\forall\ i, j = 1, \ldots, p,\ i \neq j,\ \overline{A}_i \overline{A}_j^T = A_i A_j^T + C_i C_j^T = 0.$$

Thus, the general property that must be satisfied by the augmentation block C is

$$\forall\ i, j = 1, \ldots, p,\ i \neq j,\ C_i C_j^T = -A_i A_j^T.$$

Since A has full row rank, the extended matrix $\overline{A} \in \mathbb{R}^{m \times \overline{n}}$ has also full row rank, and the has block structure

$$\overline{A} = \begin{bmatrix} \overline{A}_1 \\ \vdots \\ \overline{A}_p \end{bmatrix},\ \text{with}\ \ C = \begin{bmatrix} C_1 \\ \vdots \\ C_p \end{bmatrix}\ \text{and}\ \ \overline{A}_i = \begin{bmatrix} A_i & C_i \end{bmatrix},\ i = 1, \ldots, p,$$

composed of orthogonal partitions.

In the following, we denote as *interconnections* the non-zero overlapping columns between several partitions. In the example (II.20), the two sub-blocks $A_{1,2}$ and $A_{2,1}$, as well as $A_{2,3}$ and $A_{3,2}$, correspond to interconnections. The simplest augmentation is then to consider each couple of interconnections, and duplicate them with a minus sign for the second row partition. Using our block tridiagonal example, the augmented matrix then becomes

$$\begin{array}{c} \overline{A}_1 \\ \overline{A}_2 \\ \overline{A}_3 \end{array} \left[\begin{array}{ccccc|cc} A_{1,1} & A_{1,2} & & & & A_{1,2} & \\ & A_{2,1} & A_{2,2} & A_{2,3} & & -A_{2,1} & A_{2,3} \\ & & & A_{3,2} & A_{3,3} & & -A_{3,2} \end{array}\right].$$

The scalar product between each pair of augmented partitions ($\overline{A}_i,\ i = 1, \ldots, 3$) is now

zero as

$$A_i A_j^T = \begin{cases} 0, & \text{if } j \neq i \pm 1 \\ A_{i,j} A_{j,i}^T - A_{i,j} A_{j,i}^T = 0, & \text{if } j = i \pm 1 \end{cases},$$

There are several ways to form the augmentation block C, which do not lead to the same size, nor density [47]. We detail several techniques in Section III.1.3, including a novel approach. Note that, since the augmented partitions $\overline{A}_i$ are mutually orthogonal, the orthogonal projector onto the range of $\overline{A}$ corresponds to a sum of projectors onto orthogonal subspaces, i.e.

$$\overline{P} = \mathcal{P}_{\mathcal{R}(\overline{A}^T)} = \mathcal{P}_{\bigoplus_{i=1}^{p} \mathcal{R}(\overline{A}_i^T)} = \sum_{i=1}^{p} \mathcal{P}_{\mathcal{R}(\overline{A}_i^T)} = \sum_{i=1}^{p} \overline{A}_i^+ \overline{A}_i. \tag{II.22}$$

In a second step, a set of constraints $Y \in \mathbb{R}^{q \times \overline{n}}$ is introduced to set

$$\begin{bmatrix} \overline{A} \\ Y \end{bmatrix} = \begin{bmatrix} A & C \\ 0 & I_q \end{bmatrix}. \tag{II.23}$$

The augmented matrix is now of size $\overline{m} \times \overline{n}$ with $\overline{m} = m + q$ and $\overline{n} = n + q$. In the case of the underdetermined system, the new block $Y = \begin{bmatrix} 0 & I_q \end{bmatrix}$ is there to ensure that the solution to the augmented system remains the solution of the original problem by enforcing the additional variables to 0, i.e.

$$\begin{bmatrix} A & C \\ 0 & I_q \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}.$$

Now the partitions $\overline{A} = \begin{bmatrix} A & C \end{bmatrix}$ and $Y = \begin{bmatrix} 0 & I_q \end{bmatrix}$ are no longer orthogonal. We enforce the orthogonality through the projection of Y^T onto the orthogonal complement of $\overline{A}$. Using (II.4), this projector is rewritten $\mathcal{P}_{\mathcal{N}(\overline{A})} = I_{\overline{n}} - \mathcal{P}_{\mathcal{R}(\overline{A}^T)}$. In the end, Y is replaced by the block W defined as

$$W = \begin{bmatrix} B & S \end{bmatrix} = Y(I_{\overline{n}} - \mathcal{P}_{R(\overline{A}^T)}), \tag{II.24}$$

in which $B \in \mathbb{R}^{n \times q}$, $S \in \mathbb{R}^{q \times q}$. Then, $W \in \mathbb{R}^{q \times \overline{n}}$ is such that

$$\overline{A} W^T = 0.$$

The mutual orthogonality between $\overline{A}_i$ and W is what guarantees convergence in exactly 1 iteration, see Figure II.2.

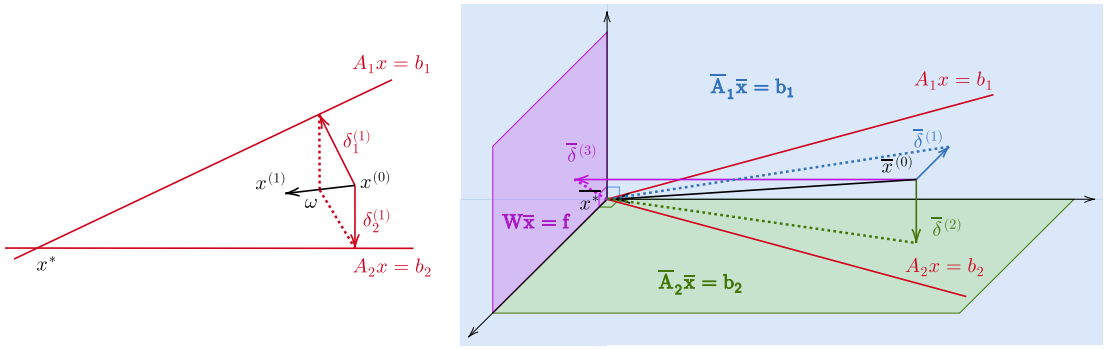


Figure II.2 *(Left)* An iteration of the BC method in 2D. *(Right)* The space has been augmented with one dimension, lines are included into orthogonal planes. The closure equations $W\overline{x} = f$ correspond to a new plane orthogonal to the 2 other ones. One iteration of the BC method gets directly to the solution of the system.

Finally, we get the fully augmented matrix

$$\begin{bmatrix} A & C \\ B & S \end{bmatrix}. \tag{II.25}$$

Depending on the problem, underdetermined or overdetermined, we then obtain a set of augmented variables as well as an augmented right-hand side. These are specified in Section II.3.2, where the choice of the additional right hand side is crucial to ensure that we can recover the solutions x_{mns} or x_{ls} of the original problems. Before computing the actual solution to our systems, we first show some general properties on the augmented matrix.

II.3.1.b Properties of the augmented matrix

In this section, we introduce some properties of the block $W = \begin{bmatrix} B & S \end{bmatrix}$ and the pseudo-inverses of the augmented partitions. They prove useful for the computation of the solution to the augmented system of equations.

Property 2. *We consider the augmented matrix from* (II.25) *with* W *as in* (II.24)*. Due to the mutual orthogonality between partitions* $\overline{A}_i$*, we define*

$$\overline{D} = \overline{A}\,\overline{A}^T = blkdiag(\overline{D_1}, \ldots, \overline{D_p}) \text{ and } \overline{D}_i = \overline{A}_i\overline{A}_i^T. \tag{II.26}$$

Then we have the following properties

(i) using (II.22)*, the symmetric matrix S is SPD, and it is written*

$$\begin{aligned} S = YW^T &= Y(I_{\overline{n}} - \overline{P})Y^T \\ &= I_q - C^T\,\overline{D}^{-1}C = I_q - \sum_{i=1}^{p} C_i^T\,\overline{D}_i^{-1}C_i. \end{aligned} \tag{II.27}$$

(ii) the normal equations of W respect the property

$$WW^T = BB^T + S^2 = S. \tag{II.28}$$

(iii) the block B is written

$$B = -C^T\,\overline{D}^{-1}A = -\sum_{i=1}^{p} C_i^T\,\overline{D}_i^{-1}A_i. \tag{II.29}$$

(iv) the normal equations of $\overline{A}$ are defined as

$$\begin{bmatrix} \overline{A} \\ W \end{bmatrix} \begin{bmatrix} \overline{A} \\ W \end{bmatrix}^T = \begin{bmatrix} \overline{D} & 0 \\ 0 & S \end{bmatrix}. \tag{II.30}$$

Proof (i): First, from the definition of W in (II.24), as $Y = \begin{bmatrix} 0 & I_q \end{bmatrix}$ is a restriction matrix on the augmentation space, S is given by

$$S = WY^T = Y(I_{\overline{n}} - \overline{P})Y^T.$$

Since $(I - \overline{P})$ is a symmetric orthogonal projector, its eigenvalues are 0 or 1. As S is a restriction of this projector, the eigenvalues of S lie in the interval $[0, 1]$, and thus S is symmetric semi positive definite.

(ii): Then, using the definition (II.24) of W, and the fact that $(I - \overline{P})$ is an orthogonal projector, as in (II.22), we have

$$\begin{aligned} WW^T = Y(I_{\overline{n}} - \overline{P})(I_{\overline{n}} - \overline{P})^T Y^T &= Y(I_{\overline{n}} - \overline{P})^2 Y^T \\ &= Y(I_{\overline{n}} - \overline{P})Y^T \\ &= S. \end{aligned}$$

Additionally, we rewrite the normal equations of W in terms of B and the symmetric

matrix S, and obtain

$$\begin{aligned} WW^T = \begin{bmatrix} B & S \end{bmatrix} \begin{bmatrix} B & S \end{bmatrix}^T &= BB^T + SS^T \\ &= BB^T + S^2. \end{aligned}$$

(i) and (iii): As A has full row rank, so does $\overline{A}$ and we know from (II.3) that

$$\overline{P} = \overline{A}^+ \overline{A} = \overline{A}^T (\overline{A}\,\overline{A}^T)^{-1} \overline{A} = \overline{A}^T \overline{D}^{-1} \overline{A}. \tag{II.31}$$

Then, using the orthogonality between $\overline{A}_i$, we express the block W in terms of the matrices A and C as

$$\begin{aligned} W = Y(I - \overline{P}) &= Y(I_{\overline{n}} - \overline{A}^T \overline{D}^{-1} \overline{A}) \\ &= Y(I_{\overline{n}} - \begin{bmatrix} A^T \\ C^T \end{bmatrix} \overline{D}^{-1} \begin{bmatrix} A & C \end{bmatrix}) \\ &= \begin{bmatrix} 0 & I_q \end{bmatrix} \begin{bmatrix} I_n - A^T \overline{D}^{-1} A & -A^T \overline{D}^{-1} C \\ -C^T \overline{D}^{-1} A & I_q - C^T \overline{D}^{-1} C \end{bmatrix} \\ &= \begin{bmatrix} -C^T \overline{D}^{-1} A & I_q - C^T \overline{D}^{-1} C \end{bmatrix} \\ &= \begin{bmatrix} -\sum_{i=1}^{p} C_i^T \overline{D_i}^{-1} A_i & I_q - \sum_{i=1}^{p} C_i^T \overline{D_i}^{-1} C_i \end{bmatrix}. \end{aligned} \tag{II.32}$$

(iv): Finally, using the orthogonality between $\overline{A}$ and W, (II.26) and (II.28), the normal equations of the augmented matrix are

$$\begin{aligned} \begin{bmatrix} \overline{A} \\ W \end{bmatrix} \begin{bmatrix} \overline{A} \\ W \end{bmatrix}^T &= \begin{bmatrix} \overline{A}\,\overline{A}^T & 0 \\ 0 & WW^T \end{bmatrix} \\ &= \begin{bmatrix} \overline{D} & 0 \\ 0 & S \end{bmatrix}. \end{aligned}$$

The pseudo-direct computation of the solution using the ABCD approach makes the critical assumption that S is invertible. The next result gives a sufficient condition for this.

Property 3. *If the matrix A has full row rank, then S is invertible and*

$$S^{-1} = I + C^T (AA^T)^{-1} C. \tag{II.33}$$

Proof We present here a proof for the invertibility of S based on algebraic arguments.

In [50], another proof based on geometric arguments is given.

From the equality $S = WW^T$ (II.28), we get the invertibility of the matrix S if and only if the matrix W^T has full column rank. In this respect, let us suppose that $W^T z = 0$, for some $z \in \mathbb{R}^{\,q}$. From (II.32), we obtain

$$W^T z = 0 \Leftrightarrow \begin{cases} A^T \overline{D}^{-1} C z = 0, \\ z - C^T \overline{D}^{-1} C z = 0. \end{cases} \tag{II.34}$$

But, from our hypothesis, the matrix A^T has full column rank, and thus from the first equation in (II.34), we must have $Cz = 0$, which gives us $z = 0$ from the second equation, and completes the proof.

Finally, in order to get an algebraic expression for the inverse of S, we remind the *Sherman-Morrison Woodbury formula* (SMW)

$$(M + U\Sigma V)^{-1} = M^{-1} - M^{-1}U(\Sigma^{-1} + VM^{-1}U)^{-1}VM^{-1}, \tag{II.35}$$

where M and Σ are invertible matrices. Since $\overline{A}\,\overline{A}^T = AA^T + CC^T$, we rewrite S as

$$S = I_q - C^T(AA^T + CC^T)^{-1}C$$

and using SMW (II.35) with $M = I_q$, $U = C^T$, $\Sigma = (AA^T)^{-1}$, and V=C, we get (II.33), which completes the proof.

Finally, we need to compute the pseudo inverse of W and $\overline{A}$ to obtain the solutions of the underdetermined and overdetermined systems respectively, and this is detailed in sections II.3.2.a and II.3.2.b.

Property 4. *If we consider W from* (II.24)*, we express its pseudo-inverse as*

$$W^+ = (I - \overline{P})Y^T S^{-1}. \tag{II.36}$$

Also, since $\overline{A}$ from (II.21) *is made of orthogonal partitions, we have the following properties*

$$\begin{aligned} \overline{A}^+ &= \begin{bmatrix} \overline{A}_1^+ & \dots & \overline{A}_p^+ \end{bmatrix}, \\ \begin{bmatrix} \overline{A} \\ W \end{bmatrix}^+ &= \begin{bmatrix} \overline{A}^+ & W^+ \end{bmatrix}. \end{aligned} \tag{II.37}$$

Proof From the previous proof II.3.1.b, we know W is full row rank. From the equality (II.28), we derive the pseudo-inverse of W

$$\begin{aligned} W^+ = W^T(WW^T)^{-1} &= W^T S^{-1} \\ &= (I_{\overline{n}} - \overline{P})Y^T S^{-1}. \end{aligned}$$

As for the second set of properties, this can be directly derived from the mutual orthogonality between the partitions $\overline{A}_i$ and the block W, and the fact that $\begin{bmatrix} \overline{A} \\ W \end{bmatrix} \begin{bmatrix} \overline{A}^T & W^T \end{bmatrix}$ is a block diagonal matrix.

II.3.1.c Decoupling subdomains through a Schur complement

Like the BC method, an interpretation of the ABCD method as a *Domain Decomposition Method* (DDM) is possible. As seen in Section II.1.2, to solve either the underdetermined system (II.10), or the least-squares problem (II.9), we must solve the normal equations

$$AA^T\widetilde{x} = b.$$

Consider that we have the augmented system (II.23) where the partitions $\overline{A}_i$ are numerically orthogonal, but not Y. The normal equations become

$$\begin{bmatrix} A & C \\ 0 & I_q \end{bmatrix} \begin{bmatrix} A & C \\ 0 & I_q \end{bmatrix}^T = \begin{bmatrix} \overline{A}_1\overline{A}_1^T & & & C_1 \\ & \ddots & & \vdots \\ & & \overline{A}_p\overline{A}_p^T & C_p \\ C_1^T & \dots & C_p^T & I_q \end{bmatrix}. \tag{II.38}$$

This bordered block diagonal form is commonly found in DDM. We can compute the Schur complement for the normal equations of the augmented matrix (II.38), which condenses all the information of the matrix on the interface variables. The expression of the Schur matrix is

$$\begin{aligned} Schur &= I_q - \sum_{i=1}^{p} C_i(\overline{A}_i\overline{A}_i^T)^{-1}C_i^T \\ &= S. \end{aligned} \tag{II.39}$$

The Schur complement of the intermediary augmented normal equations system is equal to the block S from the complete ABCD approach. In fact, projecting the augmentation block Y^T onto the nullspace of $\overline{A}$, amounts to condensing all the information of

the system onto the interfaces between subdomains.

An illustrative example: 1D Poisson

To have a better understanding of how the ABCD method works, we use a 1D Poisson discretized with finite differences as the toy problem. We use here a row partitioning and the corresponding augmented block Cimmino method. Consider the Poisson equation in 1D with homogeneous Dirichlet boundary conditions

$$\begin{cases} \nabla u = f, \text{ on } \Omega =]0,1[, \\ u = 0, \text{ on } \partial\Omega = \{0,1\}. \end{cases} \tag{II.40}$$

We discretize this equation using finite differences on a structured grid of 6 points with interval of size h, looking for the function u discretized as x_i with the forcing term f giving the right hand side elements $b_i = h^2 f_i$, see Figure II.3. We obtain the system

$$\begin{cases} -x_{i-1} + 2x_i - x_{i+1} = b_i & , i \in \{1,..,4\}, \\ x_i = 0 & , i \in \{0,5\}. \end{cases} \tag{II.41}$$

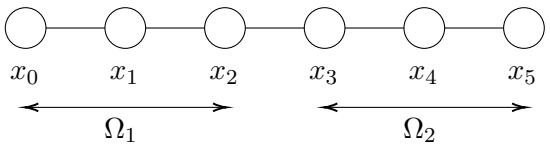


Figure II.3 1D grid with 2 domains corresponding to the partitioning of the matrix for block Cimmino.

As a result, we have to solve the classical system

$$\begin{bmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & -1 & 2 & -1 \\ & & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}.$$

As we would like to solve this system with ABCD, we partition the matrix in 2 blocks of rows of equal size 2. Basically, we are splitting the grid in 2 equal parts, also illustrated in the Figure II.3. Then we augment the system with new variables to enforce

the orthogonality. The corresponding *underdetermined problem* is

$$\begin{bmatrix}\overline{A}\\ Y\end{bmatrix}\begin{bmatrix}x\\ y\end{bmatrix} = \left[\begin{array}{cccc|cc} 2 & -1 & & & -1 & \\ -1 & 2 & -1 & & 2 & -1 \\ & -1 & 2 & -1 & 1 & -2 \\ & & -1 & 2 & & 1 \end{array}\right] \left[\begin{array}{c} x_1\\ x_2\\ x_3\\ x_4\\ \hline y_2\\ y_3 \end{array}\right] = \begin{bmatrix} b_1\\ b_2\\ b_3\\ b_4 \end{bmatrix}. \tag{II.42}$$

This system can be written as

$$\begin{cases} 2x_1 & -(x_2+y_2) & & & = b_1,\\ -x_1 & +2(x_2+y_2) & -(x_3+y_3) & & = b_2,\\ & -(x_2-y_2) & +2(x_3-y_3) & -x_4 & = b_3,\\ & & -(x_3-y_3) & +2x_4 & = b_4. \end{cases}$$

This is equivalent to decoupling completely the space Ω into 2 domains Ω_1 and Ω_2 which are augmented into domains $\overline{\Omega_1}$ and $\overline{\Omega_2}$, where each domain has its own copy of the local interface variables $(x_2^{(1)}, x_3^{(1)})$ and $(x_2^{(2)}, x_3^{(2)})$. A possible interpretation is to see the additional variables as a halo ghost where we have

$$\begin{cases} x_2^{(1)} = x_2 + y_2,\\ x_2^{(2)} = x_2 - y_2,\\ x_3^{(1)} = x_3 + y_3,\\ x_3^{(2)} = x_3 - y_3. \end{cases} \tag{II.43}$$

As illustrated in Figure II.4, $\overline{\Omega_1}$ and $\overline{\Omega_2}$ are recoupled using the Dirichlet compatibility conditions

$$\begin{cases} x_2^{(1)} = x_2^{(2)},\\ x_3^{(1)} = x_3^{(2)}. \end{cases} \tag{II.44}$$

Rewriting the system (II.42) to include the new variables and compatibility conditions

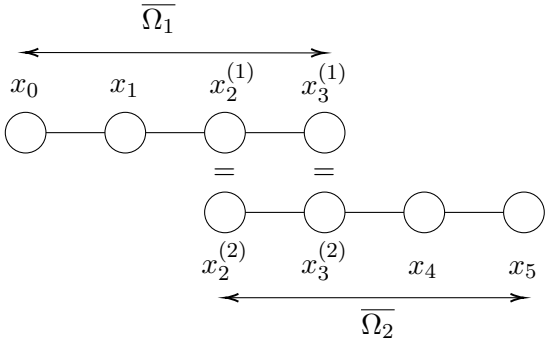


Figure II.4 1D grid with the 2 domains decoupled then re-coupled with compatibility conditions on the interface points.

(II.44), we obtain

$$\left[\begin{array}{cccc|cc} 2 & -1 & & & & \\ -1 & 2 & & & & -1 \\ \hline & & 2 & -1 & -1 & \\ & & -1 & 2 & & \\ \hline & 1 & & & -1 & \\ & & 1 & & & -1 \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2^{(1)} \\ x_3^{(2)} \\ x_4 \\ \hline x_2^{(2)} \\ x_3^{(1)} \end{array}\right] = \left[\begin{array}{c} b_1 \\ b_2 \\ b_3 \\ b_4 \\ \hline 0 \\ 0 \end{array}\right].$$

which corresponds to a stretching method applied on A [70]. This system is equivalent to the one from the ABCD method if we consider (II.43). We combine the 2 first equations and the 2 last equations in (II.43) to obtain

$$\begin{cases} 2 \times y_2 = x_2^{(1)} - x_2^{(2)}, \\ 2 \times y_3 = x_3^{(1)} - x_3^{(2)}. \end{cases}$$

The additional variables y_2 and y_3 are then interpreted as the gap between the decoupled variables in each domain. The compatibility equations become

$$\begin{cases} x_2^{(1)} = x_2^{(2)}, \\ x_3^{(1)} = x_3^{(2)}, \end{cases} \iff \begin{cases} y_2 = 0, \\ y_3 = 0. \end{cases}$$

Through this example we have illustrated the equivalence between the ABCD and a DDM, where subdomains are completely decoupled through additional variables, then recoupled using simple equality conditions, possibly leading to the solution of a Schur complement equation. While BC is an additive Schwartz method with minimal overlap,

ABCD introduces the corresponding Schur complement method when considering the transmission of information between subdomains as Dirichlet conditions at the interface.

II.3.2 Computing the pseudo-direct solution

Now, we have set the stage with a fully augmented matrix and proven properties on the augmentation blocks. In this section, we first show how to get the solution to the augmented problem. Secondly, we prove that these solutions are the same as the m.n.s. to the original underdetermined system (II.10), and the least-squares problem (II.9).

II.3.2.a Underdetermined systems

As a reminder, our goal is to compute the m.n.s. of the full rank underdetermined system (II.10), i.e. $\min \|x\|_2$ with $Ax = b$. After augmenting the matrix A as in (II.25), we have to solve the new underdetermined system

$$\begin{bmatrix} \overline{A} \\ W \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} A & C \\ B & S \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ f \end{bmatrix}. \tag{II.45}$$

In this system, the augmented partitions $\overline{A}_i$ and the additional constraints W are mutually orthogonal. To keep the solution to the original system (II.10), the right hand side f is chosen in order to enforce the additional variables y to be 0. Let's consider x_{mns} is the solution of the original system and $\begin{bmatrix} x_{mns} \\ 0 \end{bmatrix}$ is the solution of the augmented system (II.45). Using (II.22) and (II.31), the right-hand side f must satisfy

$$\begin{aligned} f = W \begin{bmatrix} x_{mns} \\ 0 \end{bmatrix} &= Y(I_{\overline{n}} - \overline{P}) \begin{bmatrix} x_{mns} \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & I_q \end{bmatrix} \begin{bmatrix} x_{mns} \\ 0 \end{bmatrix} - Y\overline{A}^{+}\overline{A} \begin{bmatrix} x_{mns} \\ 0 \end{bmatrix} \\ &= -Y\overline{A}^{+}b \\ &= -Y \sum_{i=1}^{p} \overline{A}_i^{+} b_i. \end{aligned} \tag{II.46}$$

We now prove the following property, which establishes the equivalence between the m.n.s. solutions of the systems (II.10) and (II.45).

Property 5. *Let's consider the matrix S is invertible and f is defined by* (II.46)*. Then,*

(i) if x is the m.n.s. of the system (II.10)*, the vector* $\begin{bmatrix} x \\ 0 \end{bmatrix}$ *is the m.n.s. of* (II.45)*.*

(ii) if $\begin{bmatrix} x \\ y \end{bmatrix}$ *is the m.n.s. of the system* (II.45)*, then $y = 0$ and x is the m.n.s. of* (II.10)*.*

Proof The authors in [47] proved this property for square unsymmetric systems. To extend the theory to underdetermined systems, we need to prove that the solutions have minimum norm, on top of satisfying $Ax = b$.

If x is the m.n.s. of (II.10), using (II.46), we have

$$\begin{bmatrix} \overline{A} \\ W \end{bmatrix} \begin{bmatrix} x_{mns} \\ 0 \end{bmatrix} = \begin{bmatrix} Ax_{mns} \\ W \begin{bmatrix} x_{mns} \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} b \\ f \end{bmatrix},$$

yielding the fact that $\begin{bmatrix} x_{mns} \\ 0 \end{bmatrix}$ is a solution of (II.45), which gives the first part of (i).

Let now $\begin{bmatrix} x \\ y \end{bmatrix}$ be the m.n.s. of (II.45) with f from (II.46). Hence, from (II.37) and (II.45), we write

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \overline{A} \\ W \end{bmatrix}^{+} \begin{bmatrix} b \\ f \end{bmatrix} = \begin{bmatrix} \overline{A}^{+} & W^{+} \end{bmatrix} \begin{bmatrix} b \\ f \end{bmatrix} = \overline{A}^{+} b + W^{+} f. \tag{II.47}$$

As S is invertible, using Prop. (II.36) we have

$$W^{+} f = W^{T} S^{-1} f = \begin{bmatrix} B^{T} \\ S^{T} \end{bmatrix} S^{-1} f = \begin{bmatrix} B^{T} S^{-1} f \\ f \end{bmatrix}.$$

We then have

$$\begin{aligned} \overline{A}^{+} b + W^{+} f = \begin{bmatrix} I_n & 0 \\ 0 & I_q \end{bmatrix} \overline{A}^{+} b + W^{+} f &= \begin{bmatrix} \begin{bmatrix} I_n & 0 \\ 0 & I_q \end{bmatrix} \overline{A}^{+} b \end{bmatrix} + \begin{bmatrix} B^{T} S^{-1} f \\ f \end{bmatrix} \\ &= \begin{bmatrix} [I_n \ 0] \overline{A}^{+} b \\ Y \overline{A}^{+} b \end{bmatrix} + \begin{bmatrix} B^{T} S^{-1} f \\ -Y \overline{A}^{+} b \end{bmatrix} \\ &= \begin{bmatrix} [I_n \ 0] \overline{A}^{+} b + B^{T} S^{-1} f \\ 0 \end{bmatrix}. \end{aligned} \tag{II.48}$$

Finally, (II.47) and (II.48) yield $y = 0$, hence the m.n.s. of the (consistent) system (II.45) has the form $\begin{bmatrix} x \\ 0 \end{bmatrix}$, with $x = [I_n\ 0]\overline{A}^{+}b + B^T S^{-1} f$. In particular, from (II.45), we have

$$\begin{bmatrix} A & C \\ B & S \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} = \begin{bmatrix} Ax \\ Bx \end{bmatrix} = \begin{bmatrix} b \\ f \end{bmatrix},$$

i.e. $b = Ax$ which shows that x is a solution of (II.10). Now since $\left\| \begin{bmatrix} x \\ y \end{bmatrix} \right\| = \|x\|$ is minimum (as $y = 0$ necessarily), x must correspond to x_{mns}, the m.n.s. of (II.10), and vice-versa. This completes the second part of the proof for (i), and proves (ii) at the same time.

Thanks to the mutual orthogonality between partitions, the classical BC iterations is guaranteed to converge in one iteration. Using (II.36), (II.46), (II.47), and the mutual orthogonality between $\overline{A}_i$, the final solution is then computed as

$$\begin{aligned} \begin{bmatrix} x_{mns} \\ 0 \end{bmatrix} &= \overline{A}^{+}b + W^{+}f \\ &= \sum_{i=1}^{p} \overline{A}_i^{+} b_i - (I_{\overline{n}} - \overline{P}) Y^T S^{-1} Y \sum_{i=1}^{p} \overline{A}_i^{+} b_i. \end{aligned} \tag{II.49}$$

From [47], the final algorithm to compute this solution is detailed in Algo. 7.

Algorithm 7 Solve using *ABCD* for the m.n.s. of underdetermined systems.

Input: $\overline{A}$ and b.
Output: x.

1: Build $w = \overline{A}^{+}b$, using a sum of projections, then restrict $f = -Yw$,
2: Solve $Sz = f$ with a direct solver,
3: Expand $\overline{z} = Y^T z$ and project it, $u = (I_{\overline{n}} - \overline{P})\overline{z}$,
4: Then sum $w + u$ to get the solution $\begin{bmatrix} x_{mns} \\ 0 \end{bmatrix}$.

II.3.2.b Least-squares problems

Here, we compute the unique least-squares solution x_{ls} of problem (II.9), i.e. $\min_{\widetilde{x}} \left\| \widetilde{b} - A^T \widetilde{x} \right\|_2$. After augmenting the matrix A^T, so as to ensure orthogonality between the partitions,

we have to consider the solution of the new least-squares problem

$$\min_{\widetilde{x}} \left\| \begin{bmatrix} \widetilde{b} \\ \widetilde{f} \end{bmatrix} - \begin{bmatrix} \overline{A}^T & W^T \end{bmatrix} \begin{bmatrix} \widetilde{x} \\ \widetilde{y} \end{bmatrix} \right\|. \tag{II.50}$$

with $\overline{A} = \begin{bmatrix} A & C \end{bmatrix}$ and $W = \begin{bmatrix} B & S \end{bmatrix}$.

In this system, the augmented partitions $\overline{A}_i$, corresponding to a partitioning of A in blocks of rows, and the additional constraints W are mutually orthogonal. To keep the least-squares solution of the original system (II.9), we have to make the right choice for the right hand side $\widetilde{f}$. In the following lemma, we examine what is the form of a least-squares solution to the augmented system (II.50).

Lemma 1. *For any vector $\widetilde{f} \in \mathbb{R}^{\,q}$, the matrix of the problem* (II.50) *is overdetermined and has full column rank. Moreover, its (unique) least-squares solution is given by*

$$\begin{bmatrix} \widetilde{x} \\ \widetilde{y} \end{bmatrix} = \begin{bmatrix} \overline{D}^{-1}(A\widetilde{b} + C\widetilde{f}) \\ S^{-1}(B\widetilde{b} + S\widetilde{f}) \end{bmatrix}. \tag{II.51}$$

Proof The column blocks $\overline{A}^T$ and W^T have full column rank and are orthogonal, which tells us that matrix $\begin{bmatrix} \overline{A}^T & W^T \end{bmatrix}$ in (II.50) is overdetermined (has dimensions $\overline{n} \times \overline{m},\ \overline{n} \geq \overline{m}$), and has also full column rank. Hence, using (II.30), the least-squares solution of (II.51) is the unique solution of the associated normal equations

$$\begin{bmatrix} \overline{D} & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} \widetilde{x} \\ \widetilde{y} \end{bmatrix} = \begin{bmatrix} A\widetilde{b} + C\widetilde{f} \\ B\widetilde{b} + S\widetilde{f} \end{bmatrix}.$$

which gives us (II.51) and completes the proof.

In [50], the proof of equivalence between the least-squares solution of the original and the augmented least-squares problem is made with a choice of $\widetilde{f} = 0$. Here, we establish directly the equivalence using a non-zero appropriate choice for the right hand side.

Property 6. *Given matrix S is invertible, let us suppose that the vector $\widetilde{f}$ from* (II.50) *is given by*

$$\widetilde{f} = -S^{-1}B\widetilde{b}. \tag{II.52}$$

Then, $\begin{bmatrix} \widetilde{x} \\ \widetilde{y} \end{bmatrix}$ is the (unique) least-squares solution of problem (II.50)*, iff $\widetilde{y} = 0$ and $\widetilde{x}$ is the (unique) least-squares solution of problem* (II.9)*.*

Proof Since we know that matrix $\begin{bmatrix} \overline{A}^T \\ W^T \end{bmatrix}$ has full column rank, the least-squares

solution is unique of (II.50) is unique, and the components $\widetilde{x}, \widetilde{y}$ of the least-squares solution of problem (II.50) are given by (II.51). Using (II.52), we first obtain that $\widetilde{y} = 0$, as $\widetilde{f} = -S^{-1}B\widetilde{b}$, and we have

$$\begin{aligned} \widetilde{x} = \overline{D}^{-1}(A\widetilde{b} + C\widetilde{f}) &= \overline{D}^{-1}A\widetilde{b} - \overline{D}^{-1}CS^{-1}B\widetilde{b} \\ &= \overline{D}^{-1}A\widetilde{b} + \overline{D}^{-1}CS^{-1}C^T\overline{D}^{-1}A\widetilde{b}. \end{aligned} \tag{II.53}$$

The unique solution to the problem (II.9) must satisfy the normal equations and we verify this is the case for $\widetilde{x}$. Using (II.33) and (II.53), we have

$$\begin{aligned} AA^T\widetilde{x} &= AA^T(\overline{D}^{-1}A\widetilde{b} + \overline{D}^{-1}CS^{-1}C^T\overline{D}^{-1}A\widetilde{b}) \\ &= AA^T\left[\overline{D}^{-1}A\widetilde{b} + \overline{D}^{-1}C(I + C^T(AA^T)^{-1}C)C^T\overline{D}^{-1}A\widetilde{b}\right] \\ &= AA^T\,\overline{D}^{-1}A\widetilde{b} + AA^T\,\overline{D}^{-1}\left[C + CC^T(AA^T)^{-1}C\right]C^T\,\overline{D}^{-1}A\widetilde{b} \\ &= AA^T\,\overline{D}^{-1}A\widetilde{b} + AA^T\,\overline{D}^{-1}\left[I_m + CC^T(AA^T)^{-1}\right]CC^T\,\overline{D}^{-1}A\widetilde{b} \\ &= AA^T\,\overline{D}^{-1}A\widetilde{b} + AA^T\,\overline{D}^{-1}(AA^T + CC^T)(AA^T)^{-1}CC^T\,\overline{D}^{-1}A\widetilde{b}. \end{aligned}$$

Since $\overline{D} = AA^T + CC^T$, we get

$$\begin{aligned} AA^T\widetilde{x} &= (AA^T + CC^T)\overline{D}^{-1}A\widetilde{b} \\ &= A\widetilde{b}. \end{aligned}$$

This shows that $\widetilde{x}$ is also the unique least-squares solution of (II.9) and completes the proof. Since the least-squares solution of (II.9) and the least squares solution of (II.50) are unique, this completes the proof.

Due to the mutual orthogonality between the augmented partitions, the column BC iteration is guaranteed to converge in one iteration to the solution (II.53). In order to rewrite this solution for an efficient computation, we use the following identities

$$C = \overline{A}Y^T,\ A\widetilde{b} = \overline{A}\begin{bmatrix}\widetilde{b}\\0\end{bmatrix},\ \overline{D}^{-1}\overline{A}\begin{bmatrix}\widetilde{b}\\0\end{bmatrix} = \begin{bmatrix}\overline{D}_1^{-1}\overline{A}_1\begin{bmatrix}\widetilde{b}\\0\end{bmatrix}\\ \vdots \\ \overline{D}_p^{-1}\overline{A}_p\begin{bmatrix}\widetilde{b}\\0\end{bmatrix}\end{bmatrix},\ \overline{A}^T\,\overline{D}^{-1}\overline{A}\begin{bmatrix}\widetilde{b}\\0\end{bmatrix} = \overline{H}^{row}\begin{bmatrix}\widetilde{b}\\0\end{bmatrix},$$

with $\overline{H}^{row} = \sum_{i=1}^{p}\overline{A}_i^{+}\overline{A}_i = \sum_{i=1}^{p}\mathcal{P}_{\mathcal{R}(\overline{A}_i^T)}$ All of these identities simply come from (II.26), and the fact that $\overline{A}_i$ has full row rank. From (II.53), the final solution is then given by

$\widetilde{x} = \overline{D}^{-1}A\widetilde{b} + \overline{D}^{-1}CS^{-1}C^T\overline{D}^{-1}A\widetilde{b}$, and is decomposed by blocks as

$$\widetilde{x} = \begin{bmatrix} \overline{D}_1^{-1}\overline{A}_1 \begin{bmatrix} \widetilde{b} \\ 0 \end{bmatrix} + \overline{D}_1^{-1}\overline{A}_1 Y^T S^{-1} Y \overline{H}^{row} \begin{bmatrix} \widetilde{b} \\ 0 \end{bmatrix} \\ \vdots \\ \overline{D}_p^{-1}\overline{A}_p \begin{bmatrix} \widetilde{b} \\ 0 \end{bmatrix} + \overline{D}_p^{-1}\overline{A}_p Y^T S^{-1} Y \overline{H}^{row} \begin{bmatrix} \widetilde{b} \\ 0 \end{bmatrix} \end{bmatrix}. \tag{II.54}$$

We can observe that each sub-part $\widetilde{x}_k$ of the solution, corresponding to the partition A_i, can be computed independently with

$$\widetilde{x}_i = \overline{D}_i^{-1}\overline{A}_i \begin{bmatrix} \widetilde{b} \\ 0 \end{bmatrix} + \overline{D}_i^{-1}\overline{A}_i Y^T S^{-1} Y \overline{H}^{row} \begin{bmatrix} \widetilde{b} \\ 0 \end{bmatrix}.$$

This independence between all the solution parts is again a key point of the parallelisation scheme, each part being computed on separate cores. The final algorithm to compute this solution is detailed in Algo. 8.

Algorithm 8 Solve using *ABCD* for least-squares problems.

Input: $\overline{A}$ and $\widetilde{b}$.
Output: $\widetilde{x}$.

1: Build $w = \sum_{i=1}^{p} \overline{A}_i^{+}\overline{A}_i \begin{bmatrix} \widetilde{b} \\ 0 \end{bmatrix}$ and $v_k = \overline{D}_k^{-1}\overline{A}_k \begin{bmatrix} \widetilde{b} \\ 0 \end{bmatrix}$, $k = 1, \ldots, p$, using projections,
2: Restrict $f = Yw$ and solve $Sz = f$ with a direct solver,
3: Expand $\overline{z} = Y^T z$ and compute the projection $u_k = \overline{D}_k^{-1}\overline{A}_k^T\overline{z}$,
4: Then sum and gather all the parts $x = \begin{bmatrix} u_1 + v_1 \\ \vdots \\ u_p + v_p \end{bmatrix}$.

Remark: As seen in Prop. 1 of Section II.1.2, the solutions of (II.9) and (II.10) are linked when considering $b = A\widetilde{b}$. Taking (II.49) and (II.54), we can again show in such case that $x_{mns} = A^T x_{ls}$.

Summary

We have developed the theory to establish a new pseudo-direct augmented approach, namely the ABCD method (standing for the Augmented Block Cimmino Distributed method), to solve full rank linear systems, either underdetermined or overdetermined. In [133], using the ABCD approach showed good results for some problems when compared to either the iterative BC or a more classical direct solver, e.g. MUMPS. We emphasise that

the augmented scheme is based on ingredients coming from block projection techniques, but yields a pseudo-direct method. For this reason, we denote this method as an hybrid direct-iterative scheme. The efficiency of this method depends entirely on the solution with the SPD matrix S (the matrix B is useless for the computations of the solution). As a result, the ABCD approach is efficient when the augmentation remains small (typically $< 5\%.m$), as well as when the density and then conditioning of S stays reasonnable. This will be illustrated in the experiments.

II.4 Numerical Experiments

We have extended the *CG-accelerated block Cimmino method* (BC), as well as the *augmented block Cimmino approach* (ABCD), to the solution of rectangular systems (both m.n.s. and L.S.). Here, we are interested in the computational efficiency of both methods. In particular, we demonstrate a behaviour of the methods on rectangular matrices similar to previous studies on unsymmetric square matrices [8, 47].

In order to assess these results, we apply the BC and ABCD methods on underdetermined matrices and least-squares problems extracted from the SuiteSparse Matrix Collection[2] [39] with characteristics detailed in Table II.1. The number of partitions in these cases is chosen so as to get blocks of a dimension in the order of 10 000. The partitioner used is again GRIP [126] for BC, and PaToH [32] for ABCD, see Section III.1.2 for more details.

Table II.1 Characteristics of the test matrices. m and n: the dimensions of the matrix, "elts per row": the number of nonzero values in the matrix.

Matrix	**m** ($\times 10^6$)	**n** ($\times 10^6$)	**elts per row**	**Problem**	**#Parts**
`deltaX`	0.07	0.02	3.61	Counter Example	4
`LargeRegFile`	2.11	0.80	2.34	Circuit Simulation	64
`sctap1-2r`	0.03	0.06	6.46	Linear Programming	8
`stat96v3`	0.03	1.11	98.04	Linear Programming	4
`TSOPF_RS_b39_c30`	0.06	0.06	17.97	Power Network	8

II.4.1 Accelerated block Cimmino iterations

As introduced in Section I.2.1, we use as stopping criterion for the iterative scheme a threshold of 10^{-12} for the normwise backward error. For consistent systems, either square

[2] `https://sparse.tamu.edu/`

or rectangular and underdetermined, the classical backward error used [9] is

$$\omega_k = \frac{\left\|Ax^{(k)} - b\right\|_\infty}{\|A\|_\infty \|x\|_1 + \|b\|_\infty}.$$

In the case of least squares problems, the equivalence between the system (II.9) with solution $\widetilde{x}$ and the system (II.10) when taking $b = A\widetilde{b}$ and $x = A^T\widetilde{x}$, allows us to use the same backward error with

$$\begin{aligned}\omega_k^{LS} &= \frac{\left\|Ax^{(k)} - b\right\|_\infty}{\|A\|_\infty \|x\|_1 + \|b\|_\infty} \\ &= \frac{\left\|A(A^T\widetilde{x}^{(k)} - \widetilde{b})\right\|_\infty}{\|A\|_\infty \|A^T\widetilde{x}\|_1 + \left\|A\widetilde{b}\right\|_\infty}.\end{aligned}$$

which is implicitly based on the normal equations A^TA. Note that to compute $x = A^T\widetilde{x}$ in the BC method Algo. 5, it is not necessary to compute explicitly the matrix-vector product $A^T\widetilde{x}$ at each iteration. Instead, it suffices to use the relation $\overline{Q}^{(j)} = A^T\overline{P}^{(j)}$, and introduce the additional update

$$X^{(j+1)} = A^T\widetilde{X}^{(j)} = X^{(j)} + \overline{Q}^{(j)}\lambda_j(\prod_{i=j}^{0}\gamma_i),$$

to monitor the convergence.

In [133], using the row partitioned BC algorithm from II.2.1, the authors showed that increasing reasonably the block size has a positive effect on the convergence of the method as well as on the execution time, reducing the long plateaus in the convergence profile and improving the ratio of computations vs. memory access. For the non-square matrices in Table II.1, we apply in sequential the appropriate BC method, i.e. Algo. 1 for underdetermined systems and Algo. 5 for overdetermined systems, with a block-CG size s varying in powers of 2 from 1 to 256.

We first focus on the case of the overdetermined matrix `deltaX`. Figure II.5 displays the convergence profile in terms of backward error for all block-CG sizes. When applying a simple CG acceleration, i.e. using a block size of 1, the convergence is characterised by plateaus corresponding to clusters of small eigenvalues in the iteration matrix. Increasing the block size of the block-CG algorithm can take care of several eigenvalues at once, thus breaking plateaus and accelerating the convergence.

Table II.2 gives for each block size the total execution time of the block-CG algorithm

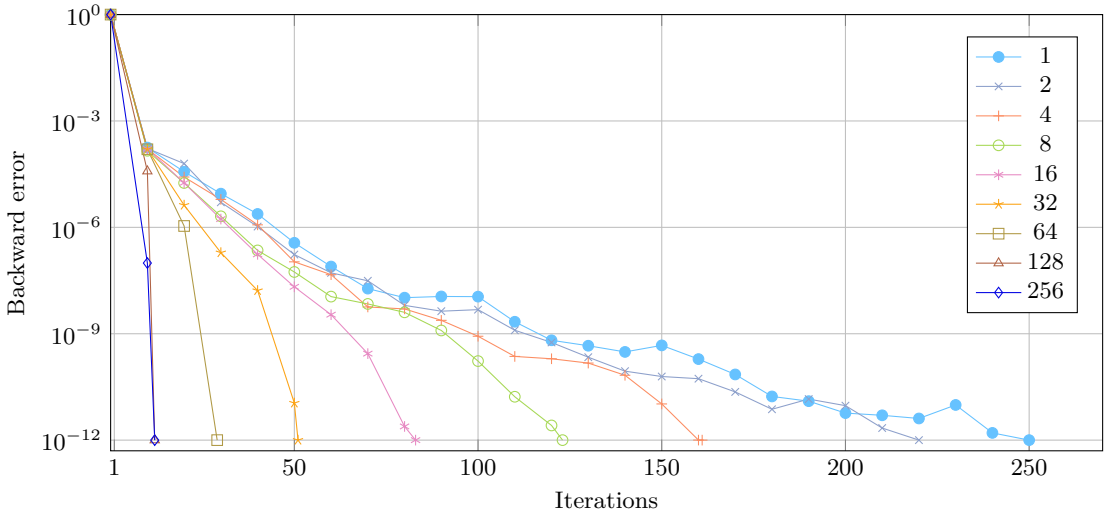


Figure II.5 Convergence of the block Cimmino method applied to the matrix `deltaX` with 16 partitions. The block-CG size varies by power of 2 from 1 to 256.

for each rectangular matrix. In all cases, there is a positive effect in using a block size higher than 1 up to a certain threshold. The optimal choice remains problem dependent. Two effects are combined which are synthesised with Figure II.6.

Table II.2 Sequential execution time in seconds of the block-CG algorithm for rectangular matrices, with an increasing block-CG size in power of 2 starting from 1 to 256.

Matrix	**1**	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**
`deltaX`	65	59	49	45	40	**39**	47	62	84
`LargeRegFile`	501	**388**	420	467	595	753	1420	2790	6100
`sctap1-2r`	99	67	37	18	8	**7**	8	12	61
`stat96v3`	188	160	148	**136**	139	171	233	317	1050

The first effect is the *reduction in iterations* in itself. However, while the convergence is apparently faster, the complexity of each iteration is increased accordingly. Using a block-CG is computationally similar to applying multiple iterations of a CG at once. If we define k_1 and k_s the iteration at convergence for block sizes 1 and s, we ideally get $k_s \ll k_1 \times s$ for an efficient reduction of the plateaus using a block-CG. We thus consider the notion of *equivalent iterations* $it^s_{eq} = s \times k_s$, which is just a better indicator for the actual computational complexity. In Figure II.6 plain lines (with left scale) represent the relative evolution of equivalent iterations, i.e. it^1_{eq}/it^s_{eq}. This ratio rises up to around 1.6

in the case of `sctap1-2r`, meaning that increasing the block size up to 16 or 32 is 60% more effective than the equivalent number of CG iterations. The ratio then lowers as the reduction in the iteration counts starts to stall.

But what explains the speed-up for the other matrices for which the ratio it^1_{eq}/it^s_{eq} deteriorates ? Through an efficient use of instruction level parallelism, as introduced in I.2.1, *the BLAS3 effect* is a phenomenon which makes matrix-matrix operations more efficient than the corresponding series of matrix-vector operations computed separately [80]. The BLAS3 effect stays advantageous as long as 1 equivalent iteration with the block size is faster than a single CG iteration. In Figure II.6, dashed lines (with right scale) display the ratio between the time for one iteration of the CG ($t^1_{it_{eq}}$) and the time for 1 equivalent iteration of the block-CG ($t^s_{it_{eq}}$). We observe a benefit from the BLAS3 effect for all considered block sizes. The execution time of an equivalent iteration is consistently reduced up to a block size of 32, a machine dependent threshold, from which the efficiency decreases and the computational load gets too high.

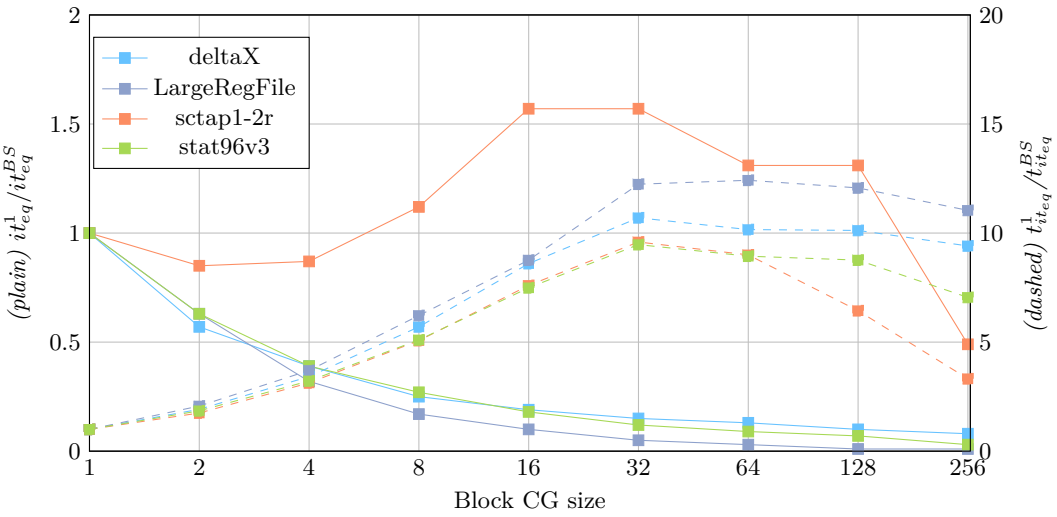


Figure II.6 *(Plain)* Equivalent number of iterations relative to the number of CG iterations. *(Dashed)* Timing for an equivalent iteration relative to the timing of a CG iteration.

In summary, it is the combination of a faster convergence and more efficient computations thanks to an appropriate choice of the block-CG size, which characterises the efficiency of the BC method.

II.4.2 Augmented block Cimmino

While choosing an appropriate block-CG size for the BC method may improve its convergence and execution time, the convergence still stays unpredictable from one test case to the other. The ABCD method enforces a convergence in only 1 iteration at the expense of additional variables and the direct solution of a Schur complement S. Using again the example matrix `deltaX`, we display in Figure II.7 the structure of the augmented matrix and the corresponding S. Even if the size of the augmentation is relatively small, S can be quite dense which introduces a difficulty for the direct solver used to solve it. For matrix `deltaX`, the augmentation size is around 27% of the original number of columns, but S has 2 614 elements per row which is rather dense compared to the original matrix. Additionally, the conditioning of this matrix can be high, and here $\kappa(S) = \mathcal{O}(10^{14})$. In Section III.1.1, we introduce a scaling method for the ABCD method to improve this conditioning.

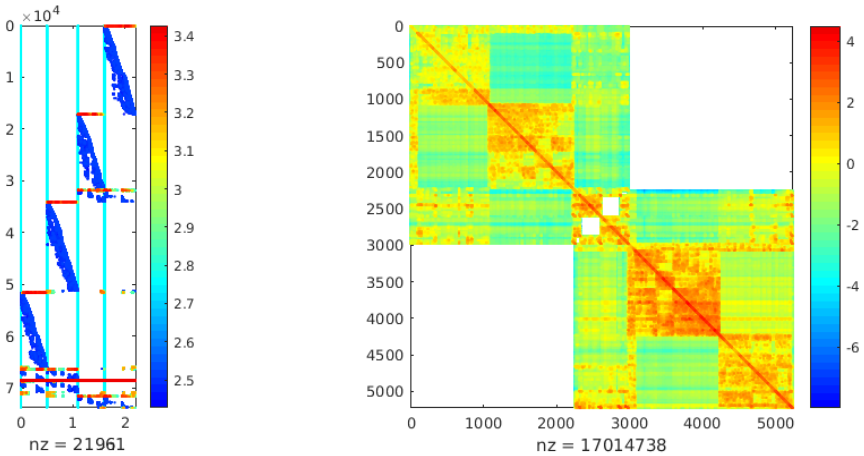


Figure II.7 *(Left)* Structure of the augmented matrix `deltaX` with a partitioning in 4 blocks of columns, and *(Right)* structure of the resulting Schur complement S.

We apply ABCD on the matrices from Table II.1 using for each case the augmentation method giving smallest augmentation among the methods introduced in [47]. Table II.3 displays the resulting augmentation size, number of entries in S, execution time, memory and accuracy. The memory requirement is split into the memory used for the factorisation of the projection systems, and the factorisation of S. We also include for reference the memory required by the iterative BC method as well as the execution time and accuracy.

We first observe that we obtain a backward error of the order of machine precision

Table II.3 Application of the ABCD method on the test matrices. Displayed are the size/density of S, and the memory required for its factorisation. Also, we give for BC and ABCD the memory requirement for the factorisation of the projection systems, as well as the execution time and accuracy.

Matrix	ABCD						BC		
	S		Memory (MB)		Time (s)	Bwd. err.	Memory	Time (s)	Bwd. err.
	m	*elts per row*	*Facto*	*Facto S*			*Facto (MB)*		
`deltaX`	5 226	2 614	129	273	21	$2 \cdot 10^{-18}$	118	44	$1 \cdot 10^{-11}$
`LargeRegFile`	214 422	Memory requirements too high					1315	1272	$4 \cdot 10^{-16}$
`sctap1-2r`	952	239	190	21	5	$1 \cdot 10^{-16}$	59	8	$3 \cdot 10^{-14}$
`stat96v3`	1 026	534	459	13	65	$2 \cdot 10^{-20}$	458	152	$4 \cdot 10^{-13}$

with ABCD in any case, compared to the more limited accuracy of block Cimmino in general. In terms of memory, the amount used for the factorisation of the projection systems is systematically higher in ABCD compared to BC, which is natural as the matrix is then embedded in a larger space. The determining factor is then the memory used for the factorisation of S in ABCD, which is directly linked to the potentially high size and density of the Schur complement. Taking the example of `deltaX`, the memory required for S is around 2.1 times larger than the memory for the projections themselves in ABCD. Worst case is the matrix `LargeRegFile` for which the memory requirements induced by its large augmentation is too high for the machine we use. In Section III.1.3, we introduce a new augmentation technique in order to decrease the size of the augmentation in general. In the case of `sctap1-2r` and `stat93v3`, the complexity of solving S is not high, and these are the cases where the hybrid direct-iterative solver can be of great interest especially when compared to other solvers.

Finally, thanks to the convergence in one iteration and the direct solver used to compute the projections and solve the Schur complement, we observe here a total execution time lower in all cases when using ABCD compared to BC. This is no generality though. In Section III.2, we present a thorough comparison between BC and ABCD as implemented in parallel inside the ABCD-Solver[3], as well as a comparison with the state-of-the-art direct solvers MUMPS[4] for square matrices, and QR-MUMPS[5] for rectangular matrices.

[3] `http://abcd.enseeiht.fr/`
[4] `http://mumps-solver.org/`
[5] `http://buttari.perso.enseeiht.fr/qr_mumps/`

II.4.3 Handling unsymmetric square matrices

The approaches presented in this chapter are applicable to any full rank system, and thus also to unsymmetric square matrices as was the purpose of the original augmented block Cimmino method [47]. The advantage is that we now have the choice between partitioning the matrix in blocks of rows or blocks of columns, depending on the problem.

To observe the potential impact of choosing the orientation of the partitioning, we focus on the unsymmetric matrix `TSOPF_RS_b39_c30` from Table II.1. Applying the block Cimmino methods with either a row or a column partitioning on this matrix gives two very different behaviour. The left part of Figure II.8 displays the convergence of the BC method for both type of partitioning. While we get a convergence in 141 iterations with super linear convergence using a row partitioning, the column partitioning gives a convergence in 432 iterations with long plateaus. In the case of the ABCD method, a row partitioning gives an augmentation of size 140, while the column partitioning induces an augmentation of size 4 731 with an ill-conditioned Schur complement S, see the right part of Figure II.8.

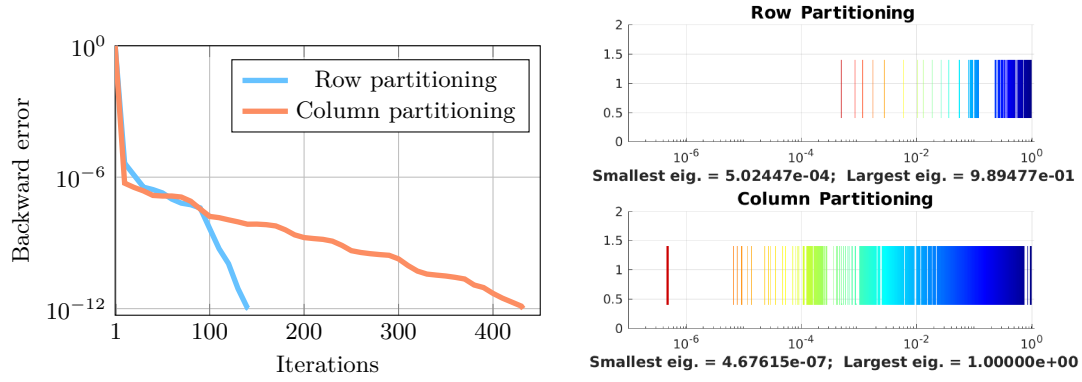


Figure II.8 The BC and ABCD methods applied to `TSOPF_RS_b39_c30` partitioned in 8 blocks of rows or 8 blocks of columns. *(Left)* Convergence of the iterative method. *(Right)* Spectrum of the Schur complement S.

While in this specific case a row partitioning is preferable, the choice for the partitioning strategy is completely problem dependent. We now apply the iterative and augmented method on all square unsymmetric matrices from the SuiteSparse Matrix Collection with more than 1 000 rows and columns. After removing the matrices for which the ABCD-Solver could not finish, either because the system is ultimately (numerically) rank deficient or because of too high memory requirements, around 700 matrices were

kept. Additionally, matrices from the same family were merged, e.g. adder_dcop_01 to adder_dcop_48, and the mean value taken for iterations and augmentation sizes. In total, 240 classes of problems remain, with 158 problems for BC and 238 problems for ABCD. The partitioner used is again GRIP [126] for BC, and PaToH [32] for ABCD, see Section III.1.2 for more details. We use for BC a block-CG size of 4, and for ABCD the technique from [47] giving smallest augmentation for each matrix. Table II.4 gives the number of classes where each partitioner is best in term of iterations for convergence and augmentation sizes. These results seem to indicate that it is better to use a column partitioning for the iterative BC method, and a row partitioning for better efficiency when considering the pseudo-direct ABCD method, but this is not systematic. In Appendix A.3, Figure A.1 represents the difference in number of iterations vs. the difference in augmentation size from the use of a row and a column partitioning applied on each test case.

Obviously, the convergence and augmentation size are problem dependent (sparse structure, numerical values, ...), but it is good to have the opportunity to choose between the row or column partitioning alternatives, to better exploit the particularities of the given sparse matrix.

Table II.4 For each method (BC or ABCD), number of classes of problems where the partitioning giving best result is row, column, or both. The choice is decided in terms of smallest number of iterations or augmentation size.

Best part.	Row	Equal	Column
BC (it.)	40	40	**78**
ABCD (aug. size.)	**114**	42	82

Concluding remark

In this chapter, we introduced the extended CG-acceleration of the block Cimmino iterations for the minimum-norm solution of underdetermined systems and the solution of least squares problems. The latter performs a partitioning of the columns instead of the rows. We show the sequential efficiency of the method for some sparse matrices and in particular the effect of increasing the block size which reduces the number of iterations for convergence while taking benefit from the BLAS3 effect.

While the memory requirement is low, the efficiency of the method stays problem dependent with an unpredictable convergence characterised by a convergence profile possibly displaying either long plateaus or superlinear convergence. The augmented

block Cimmino method is then a pseudo-direct alternative which we also extended to the solution of full rank rectangular systems. To compute the final solution, a relatively smaller symmetric positive definite system is built and solved. The major drawback of the augmented block Cimmino method is then its dependence on the numerical properties of this smaller (but denser) SPD matrix S. In cases where the matrix S is too big or dense, the memory and computation required to solve it may become prohibitive.

In the next chapter, we introduce preprocessing methods to improve the behaviour of the iterative and augmented approaches through scaling methods, a good construction of partitions, and a new augmentation method to reduce the size of S. The advantage of the augmented block Cimmino method is a decomposition of the matrix into subproblems, which breaks the complexity in computation and memory of the direct solver, internally used for the computation of projections. Due to the independence between projections, a natural parallelism appears. In section III.2, we detail the parallel scheme for these methods as they were implemented in the ABCD-Solver. We then demonstrate the parallel efficiency compared to state-of-the-art direct solvers.

Additionally, thanks to elements from the multigrid methods, we introduce in chapter IV a new approach to control the size of S via approximate orthogonality between partitions to obtain an iterative method with linear convergence.

Chapter III

Parallel implementation

At this point, we have developed 2 approaches for the solution of the problem (II.1). An iterative method [11, 96, 112], based on a stabilised block-CG (BC), and a pseudo-direct method [47, 51, 133], which embeds the original matrix into an augmented space (ABCD), to accelerate the classical block Cimmino iterations [52].
The convergence and overall robustness of these 2 approaches are highly dependent on the numerical properties of the original problem and the partitioning of the system into blocks. To make sure we get the best performance from our methods, we need to preprocess these matrices which display various block structures, densities, and numerical properties. In Section III.1, we introduce preprocessing techniques aiming at the improvement of these properties.
Both the iterative or the pseudo-direct methods rely on the partitioning of the original matrix A into blocks of rows, on which independent projections are computed. This independence is a key point of the parallelism we manage in distributed memory environments. In section III.2, we present the parallelism scheme we use for the parallel implementation of the ABCD-Solver.
In particular, we improve the scalability of our parallel implementation through a combination of load balancing methods and communication reducing techniques. Finally, we demonstrate the parallel efficiency of the solver on a distributed memory architecture by solving several problems from the SuiteSparse Matrix Collection.

III.1 Preprocessing

> Science, my lad, is made up of mistakes, but they are mistakes which it is useful to make, because they lead little by little to the truth.
>
> *A Journey to the Center of the Earth*, Jules Verne

In this section, we study practical techniques to improve the numerical properties of the original system, as well as the augmented problem in the case of the pseudo-direct approach. Here, we only need to focus on the row partitioning method, indeed the preprocessing performed on an underdetermined A, independent from the right hand side, is also relevant to the overdetermined matrix A^T partitioned in blocks of columns. First, we will consider the problem of scaling the matrix, original or augmented. Then we will introduce several approaches to partition and augment the matrix.

III.1.1 Scaling

Generally, we assume that the matrix A and the solution x are well-scaled, however norms are largely affected by this scaling. In particular, the condition number $\mathcal{K}(A) = \|A\| \, \|A^{-1}\|$ often becomes unnecessarily large because of a poorly scaled matrix [46]. The best scaling option is to build a well scaled matrix to begin with, by having consistent units in data and variables, and making consistent modelling assumptions. We turn to automatic scaling when this is not possible. The standard scaling methods aim at building diagonal matrices D_r and D_c, called *scaling matrices*, such that the matrix $\widetilde{A} = D_r A D_c$ is well scaled. However, it is difficult to construct a generic matrix scaling, see the discussion in [59], because when dealing with various real problems it is difficult to define what data is significant in a general sense.

We can consider, for instance, normalising both rows and columns close to 1, so as to get a doubly stochastic matrix. In [92, 113], the authors achieve this through an algorithm which scales iteratively the rows and columns by the square roots of their norm. In some cases, this algorithm gives a scaling with optimal condition number in the sense of [13], and at least never increases the condition number. This scaling was implemented in sequential in *Harwell Subroutine Library* (HSL)HSL as MC77, and a parallel version

[31] is available in the direct solver MUMPS[1] [6], using 1-norm or ∞-norm. However, this approach is feasible only when considering square matrices with total support.

In the more general case, when using the scaled matrix $\widetilde{A}$ instead of A, the underdetermined system (II.10) becomes $min\,\|y\|$ s.t. $D_rAD_cy = D_rb$, with $y = D_c^{-1}x$. However, $min\,\|x\|$ from the original system is not equivalent to min $D_c^{-1}x$. As for the overdetermined system (II.9), the problem is reduced to $min\left\|D_c(A^TD_r\widetilde{y} - \widetilde{b})\right\|$ which is not equivalent to $min\left\|A^T\widetilde{x} - \widetilde{b}\right\|$. In summary, for rectangular systems, only the row scaling can be applied in general, as opposed to square matrices for which solving $Ax = b$ is equivalent to $D_rAD_cx = D_rb$ with $y = D_c^{-1}x$.

III.1.1.a Iterative block Cimmino

In the ABCD-Solver, our default choice is to apply 3 successive MC77 scaling steps: 5 iterations in the ∞-norm, 20 in the 1-norm, and 10 in the ∞-norm, using the parallel implementation of MC77 in MUMPS. We also add, as a final step, an optional normalisation of the rows in the 2-norm. In the case of rectangular matrices, only this last step is applied so as not to change the minimisation problem itself.

The scaling of the rows has no impact on the convergence of the BC algorithm as shown in [52, 112]. However, the row scaling affects the numerical pivoting in the direct solver when solving the projection systems (II.16). Thus the row scaling should improve the quality of the factorisation of these projection systems, see e.g. [7].

The column scaling of the matrix, on the contrary, affects the convergence properties of the method. In [112], the author uses ellipsoidal norms for the projections, implicitly using a column scaling on interface variables, to accelerate the convergence of BC.

III.1.1.b Augmented block Cimmino

In the case of the pseudo-direct ABCD method, it is not possible to scale the augmented system after its construction as we may lose the convergence in 1 iteration, either with row or column scaling, if this is not done in a coherent manner so as to maintain orthogonality between the augmented partitions. There is a simple way to maintain this orthogonality. First scale the matrix A as D_rAD_c, as was done for the iterative BC, then construct C and W from this scaled matrix. In this case, the column scaling for the block C directly comes from D_c restricted to the interconnected columns, and its row scaling is D_r. Also in this context, W is constructed with the classical projection of $Y = \begin{bmatrix}0 & I_q\end{bmatrix}$ on the

[1] `http://mumps-solver.org/`

nullspace of the scaled $\overline{A}$, see Section II.3.1. Finally, we may choose a row scaling for W arbitrarily, the convergence staying independent from any row scaling.

In order to observe the impact of the scaling on the ABCD method, we extract full rank square unsymmetric and rectangular matrices from the SuiteSparse Matrix Collection[2] [39], see Table III.1. The number of partitions is set so that in these cases, the size of a partition is of the order of 10 000. We use to build these partitions the PaToH partitioner [32] combined with one of the augmentation techniques introduced in [47], chosen so as to give the smallest augmentation size. Given the number of partitions, we exploit the 2 methods for permutation and partitioning described in Section III.1.2. For each matrix, we use for the iterative BC solver the partitioning inducing the smallest number of iterations, and for the pseudo-direct ABCD solver, we use the combination of partitioning and augmentation technique giving the smallest augmentation size. After

Table III.1 Characteristics of the test matrices. n: the order of the matrix, nnz: the number of nonzero values in the matrix.

Matrix	**m** ($\times 10^6$)	**n** ($\times 10^6$)	**elts per row**	**#Parts**	**Problem**
`deltaX`	0.07	0.02	3.61	4	Counter Example
`sctap1-2r`	0.03	0.06	6.46	8	Linear Programming
`bayer01`	0.06	0.06	4.76	8	Chemical Process

scaling the matrices `deltaX`, `sctap1-2r`, and `bayer01` with the default chosen approach, the augmentation technique is applied. The conditioning of the corresponding Schur complement S and the projection systems are shown in Table III.2 for the 3 test matrices with and without scaling. The scaling of rows and columns is very efficient in the case of `bayer01` to decrease the conditioning of both S and the projection systems.

For the rectangular cases `deltaX` and `sctap1-2r`, when applying the augmented block Cimmino method on the scaled system $\widetilde{A} = D_r A$, thus $\widetilde{C} = D_r C$ the Schur complement becomes

$$\begin{aligned} \widetilde{S} &= I_q - \widetilde{C}^T(\widetilde{A}\widetilde{A}^T + \widetilde{C}\widetilde{C}^T)^{-1}\widetilde{C} \\ &= I_q - CD_r(D_r AA^T D_r + D_r CC^T D_r)^{-1} D_r C \\ &= I_q - C(AA^T + CC^T)^{-1}C = S. \end{aligned} \tag{III.1}$$

However, we have the possibility to scale the rows of S in 2-norm. In the case of `bayer01`, there is a large benefit from this scaling which shifts even further the spectrum of S

[2] `https://sparse.tamu.edu/`

towards 1. In the case of the rectangular matrices however, the effect is not so beneficial.

Table III.2 Effect of a scaling applied to the matrix before the application of the augmented block Cimmino method. The conditioning of the Schur complement S and the average conditioning of the projection systems are given.

Matrix	**scaling**	$\kappa(\mathbf{S})$	**mean** $\kappa(\mathbf{proj})$
`deltaX`	No scaling	$2.95 \cdot 10^{16}$	$1.63 \cdot 10^{4}$
	2-norm S	$3.15 \cdot 10^{15}$	$1.65 \cdot 10^{4}$
`sctap1-2r`	No scaling	$5.24 \cdot 10^{5}$	$3.27 \cdot 10^{4}$
	2-norm S	$5.31 \cdot 10^{5}$	$3.27 \cdot 10^{4}$
`bayer01`	No scaling	$1.45 \cdot 10^{11}$	$8.90 \cdot 10^{30}$
	Scaling	$1.89 \cdot 10^{7}$	$2.74 \cdot 10^{10}$
	Scaling + 2-norm S	$4.48 \cdot 10^{3}$	$2.74 \cdot 10^{10}$

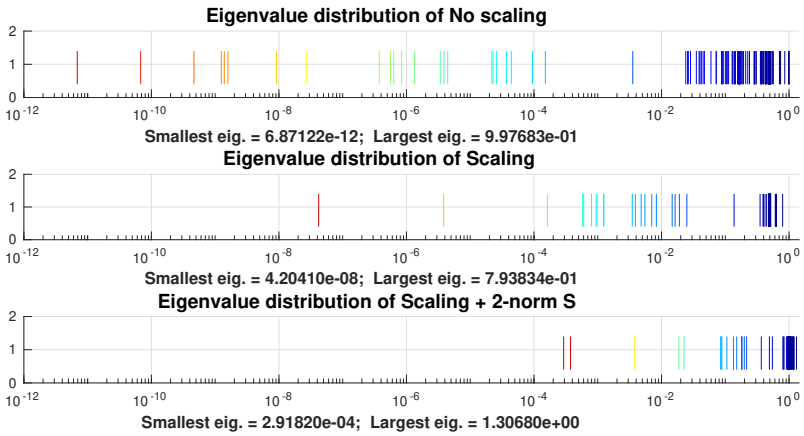


Figure III.1 Spectrum of the Schur complement matrix obtained using the augmented block Cimmino method applied to the matrix `bayer01` partitioned in 8 blocks of rows.

III.1.2 Partitioning

In this section, we are interested in the construction of the partitions (II.5) for the block Cimmino scheme. We first establish what we consider a good partitioning strategy. Then, we focus on 2 different types of partitioning methods: partitioning based on bandwidth reducing techniques, and partitioning based on graph-partitioning techniques.

III.1.2.a Targeting the angles with partitioning

Although with the block Cimmino iterative scheme, the computation of the solution is reduced to projections on subdomains, the convergence stays very problem dependent. If the matrix A is ill-conditioned, this means that there exists a linear combination of the rows which is close to zero. To get a fast convergence, one must group these specific linear combinations inside a partition [133]. The burden of the ill-conditioning then lies in the computation of the projections handled by the direct solver. The convergence behaviour of the accelerated block Cimmino iterations is directly linked to the spectrum of the matrix H [112]. Even though the CG method is guaranteed to converge after m iterations, it is desirable to obtain convergence way before. For each partition A_i, $i = 1, \ldots, p$, let a QR decomposition be defined as $A_i^T = Q_i D_i^{\frac{1}{2}}$, where $Q_i \in \mathbb{R}^{m_i \times n}$ is an orthonormal matrix.Using their definition from (II.13) and (II.15), we show that the iteration matrices from row (H^{row}) and column (H^{col}) block Cimmino, share the same non-zero spectrum. Firstly, we symmetrize H^{col} with the matrix $D^{\frac{1}{2}}$ to get

$$\begin{aligned} D^{\frac{1}{2}} H^{col} D^{-\frac{1}{2}} &= (D^{-\frac{1}{2}} A^T)(A D^{-\frac{1}{2}}) \\ &= \begin{pmatrix} Q_1 & \ldots & Q_p \end{pmatrix}^T \begin{pmatrix} Q_1 & \ldots & Q_p \end{pmatrix} = \widetilde{H}. \end{aligned}$$

Secondly, concerning H^{row} we have

$$\begin{aligned} H^{row} = \sum_{i=1}^{p} A_i^T (A_i A_i^T)^{-1} A_i &= \sum_{i=1}^{p} Q_i Q_i^T \\ &= \begin{pmatrix} Q_1 & \ldots & Q_p \end{pmatrix} \begin{pmatrix} Q_1 & \ldots & Q_p \end{pmatrix}^T. \end{aligned}$$

Thus, from the theory of the singular value decomposition (see [67, 69]), the nonzero eigenvalues of H^{row} are also the nonzero eigenvalues of $\widetilde{H}$. All in all, H^{row} and H^{col} have the same nonzero spectrum as that of the matrix

$$\widetilde{H} = \begin{pmatrix} I_{m_1} & Q_1^T Q_2 & \ldots & \ldots & Q_1^T Q_p \\ Q_2^T Q1 & & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & & Q_{p-1}^T Q_p \\ Q_p^T Q_1 & \ldots & \ldots & Q_p^T Q_{p-1} & I_{m_p} \end{pmatrix},$$

where the $Q_i^T Q_j$ are matrices whose singular values represent the cosines of the principal angles between the subspaces $\mathcal{R}(A_i)$ and $\mathcal{R}(A_j)$, as defined in [21]). These principal

angles (see [69, pages 584-585]) are also defined successively by

$$\cos(\Psi_k) = \max_{u\in\mathcal{R}(A_i^T)} \max_{v\in\mathcal{R}(A_i^T)} \frac{u^T v}{\|u\|\|v\|} \quad \text{subject to} \begin{cases} u^T u_p = 0, & p = 1,\dots,k-1, \\ v^T v_p = 0, & p = 1,\dots,k-1, \end{cases}$$
$$= \frac{u_k^T v_k}{\|u_k\|\|v_k\|},$$

with k varying from 1 to $m_{ij} = \min(\dim(\mathcal{R}({A_i}^T)), \dim(\mathcal{R}({A_j}^T)))$.
Note that the principal angles satisfy $0 \leq \Psi_1 \leq \ldots \leq \Psi_{m_{ij}} \leq \Pi/2$ and that having $\Psi_k = \Pi/2, \forall\ k = 1,\dots,m_{ij}$ is equivalent to $\mathcal{R}(A_i^T)$ being orthogonal to $\mathcal{R}(A_j^T)$. In the extreme case where all partitions are mutually orthogonal, the accelerated (and classical) BC converges in 1 iteration. This is the idea behind the ABCD method. Then the intuition is the following: the wider the principal angles between subspaces, the closer H is to the identity, and the faster the convergence of the iterative scheme should be.
However, knowing the principal angles between partitions is not enough to have a full grasp of the spectrum of H, except in special cases, e.g. a two-blocks partitioning strategy.

Following the example in [52], let's consider the original matrix $A \in \mathbb{R}^{m\times n}$ is partitioned with a *two-blocks partitioning* as introduced in [52]

$$A = \begin{pmatrix} B_1 \\ B_2 \end{pmatrix}.$$

Illustrated in Figure III.2, this type of partitioning is obtained when the normal equations are block tridiagonal. Then naturally, the even numbered blocks are mutually orthogonal, and so are the odd numbered blocks: $\forall i = 1,\dots,p-2, A_i A_{i+2}^T = 0$. We thus group the odd (resp. even) partitions in $B_1 \in \mathbb{R}^{m_1\times n}$ (resp. $B_2 \in \mathbb{R}^{m_2\times n}$). In this case, we have $H^{row} = \mathcal{P}_{\mathcal{R}(B_1)} + \mathcal{P}_{\mathcal{R}(B_2)}$, where each projector is a sum of independent projectors because the partitions within B_1 and B_2 are structurally mutually orthogonal. Then, considering $m_{min} = min(m_1, m_2)$, the authors in [52] showed that the spectrum of H^{row} is

$$\begin{aligned} \lambda_k &= 1 + \cos\Psi_k, & k &= 1,\dots,m_{min}, \\ \lambda_k &= 1 - \cos\Psi_{k-m_{min}}, & k &= m_{min}+1,\dots,2m_{min}, \\ \lambda_k &= 1, & k &= 2m_{min}+1,\dots,n, \end{aligned}$$

where Ψ_k, $k = 1,\dots,m_{min}$ are the principal angles between $\mathcal{R}({B_1}^T)$ and $\mathcal{R}({B_2}^T)$. With such two-blocks partitioning, we conclude that the CG algorithm would not take more than $2m_{min}$ iterations to converge in exact arithmetic. The smallest block of is called the *interface block*. With a very small interface between the partitions we would essentially

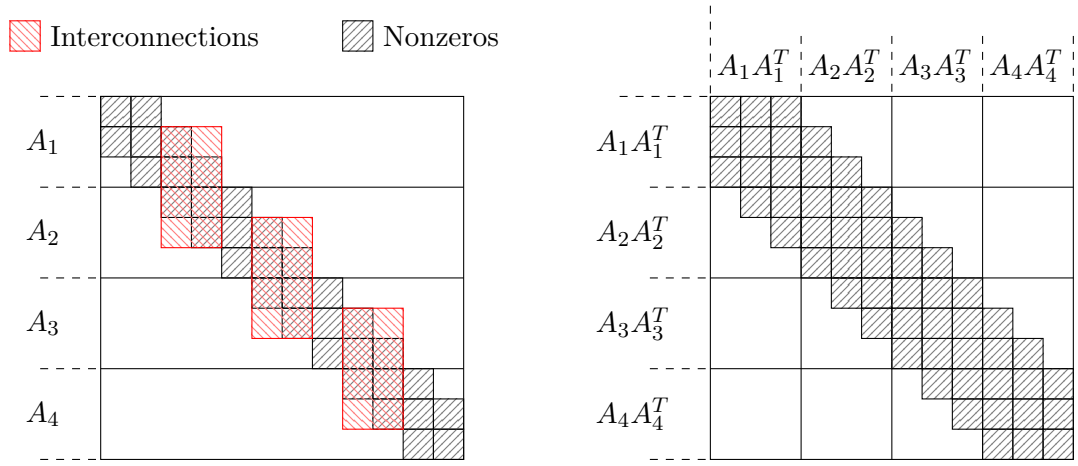


Figure III.2 Block tridiagonal matrix with a two-blocks partitioning and the corresponding normal equations, sharing also a block tridiagonal structure with respect to the chosen partitioning. Even numbered partitions are not interconnected, and neither are the odd numbered partitions.

reach a faster convergence in CG.

Based on the two discussions above, we derive 2 linked objectives for a well-defined partitioning

Objective 1: Minimise the principal angles between subspaces spanned by the partitions. Or phrased differently, reduce the ill-conditioning across partitions, while concentrating it inside partitions to be handled by a direct solver upon the computation of projections.

Objective 2: Minimise the size of the interface interconnecting the partitions.

In a parallel implementation, both criteria are combined with a global balance constraint between the workload needed to compute each projection. Concerning the ABCD scheme, the goal of a partitioning is to have a Schur complement S as small as possible to be solved by a direct solver, while keeping a certain balance between the computation of projections, as these are performed in parallel. The augmentation size in the ABCD scheme, and thus the size of S, is directly linked to the size of the structural interface between subdomains, which is exactly what *Objective 2* targets.

Generally speaking, partitioning the matrix can also imply a permutation of the rows in order to redistribute the entries between blocks and have more flexibility to achieve the two objectives above. We thus determine a permutation matrix P, and we consider

the row permuted problem to solve,

$$\min \|x\|_2 \text{ when } x \in \{x; \|P(b - Ax)\|_2 = min!\},$$

with a partitioning of PA. Note that orthonormal matrices (such as permutations) do not change the 2-norms, and thus the least squares problem remain the same.

A tremendous amount of reordering and partitioning methods have been studied over time in contexts as varied as

- Reordering to minimise fill-in for solvers based on Gaussian elimination [46],
- Building a coarsening in algebraic Multigrid methods [132],
- Image segmentation [119],
- Determination of subdomains in domain decomposition methods [40].

In the context of the BC and ABCD methods, several partitioners were considered [11, 48, 96, 112, 133] which we summarise in the 2 following Sections. We distinguish 2 class of partitioners: methods relying on bandwidth reducing techniques motivated by the good properties of the 2-blocks partitioning. Then, techniques from domain decomposition, particularly methods based on graph-partitioning techniques. Finally, we introduce a new partitioning technique relying on defining partitions which can overlap.

As stated in Section II.3, the BC and ABCD schemes are designed to tackle rectangular or unsymmetric square matrices. However, most reordering and partitioning methods target symmetric square matrices. The solution is to analyse symmetric matrices linked to the original matrix, such as AA^T, A^TA, $A+A^T$ or the bipartite graph which ultimately corresponds to the adjacency graph of the symmetric matrix $\begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}$ [46].

III.1.2.b Reduction of bandwidth for 2-blocks partitioning

The first considered approach was relying on bandwidth reducing methods to permute the matrix into block tri-diagonal form. In this form, a two-blocks partitioning is simple to obtain, see Figure III.2. For matrices coming from the discretization of a 1D PDE problem, this structure often naturally appears. Else, we need to apply an automatic ordering tool on the pre-existing matrix [46]. The traditional choice is the *Cuthill-McKee algorithm* [36] (CM), and its reverse version which was observed to often give better results in [64].

For the partitioning in block Cimmino, the authors in [112] propose to apply Cuthill-McKee to $A + A^T$, while in [42] the normal equations AA^T are reordered. In both cases, we can obtain a two-blocks row partitioning of the reordered matrix. If one considers,

for instance, only the sparsity structure of the normal equations AA^T, for permutation purposes, not the values in the matrix, only Objective 2 may be targeted (i.e. reducing the size of the interconnections between partitions). As shown in [133], the partitions obtained with this approach often raise very unbalanced sizes, leading to a poor degree of parallelism.

In [42], the authors propose to apply Cuthill-McKee on normalised normal equations in which small values are filtered out based on a threshold τ. The goal is to sparsify the pattern of the resulting approximate matrix AA^T, and to have more freedom to improve the balance between partition sizes. Considering that the rows of the original matrix were normalised in 2-norm, the entries in the normalised normal equations correspond to the cosines of the principal angles between every pair of rows, i.e. the degree of colinearity between rows. Filtering small entries then correspond to ignoring interconnections between already quasi-orthogonal pairs of rows. The result is a partitioning close to the strict two-blocks partitioning. The lower the threshold τ for filtering is, the closer to two-blocks partitioning we get. Additionally, as the normal equations get sparser after filtering, their bandwidth after reordering should be smaller, and the number of level-sets is higher than with the complete AA^T. This fact helps in getting better balanced workloads for the projections, and thus a higher parallelism.

III.1.2.c Graph partitioning

In a parallel context, finding the solution of a linear system typically implies the subdivision of the problem into p parts. As we introduced in Section I.2.2, this is particularly the case with DDM where sets of equations are assigned to "subdomains". A processor then holds a set of equations and the associated vector components. Here, the goal is to minimise communications, i.e. the size of data exchanged on the interfaces, while keeping balanced workloads for the subproblems over the processors.

For the discretization of PDE problems, the construction of these subsets is often based on the geometry for simple situations, e.g. structured grids. When the situation is more complex, an algebraic approach is required, commonly using graph partitioning techniques. These techniques are based on the dissection of the matrix graph. The principle is to find small subsets of edges or nodes in the graph, called *separators*, which cut the graph in smaller parts when removed. Reordering the matrix based on such cuts leads to a typical bordered block diagonal form [63]. Each smaller part can then be similarly dissected if needed, leading to a technique called nested dissection [62].

Nested dissection is the base of several well-known graph partitioners such as METIS[3] or SCOTCH[4]. These two partitioners use multilevel approaches where separators are constructed on coarsened graphs, then these separators are refined using the Fiduccia and Mattheyses algorithm [58].

Generally these partitioners use the number of edge-cuts, i.e. edges linking nodes from different subdomains, as a measure of interface reduction. However, this criterion is not enough, e.g. if a single node is linked to k nodes in another subdomain then k edge-cuts are artificially counted while only one variable is involved in an exchange between partitions in practice. This simple fact was used to devise the notion of *Hypergraph partitioners*, particularly usefull to partition sparse matrices.

PaToH: hypergraph partitioning The concept of hypergraph is a generalisation of the notion of graphs. A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ consists of a set of nodes $\mathcal{V}$ and a set of hyperedges $\mathcal{N}$, called nets, linking two or more nodes. Hypergraphs represent the structure of a matrix: each row is represented by a node, and each column is represented by a net linking the rows with a non-zero entry in this column. Figure III.3 shows a sparse matrix and its hypergraph representation.

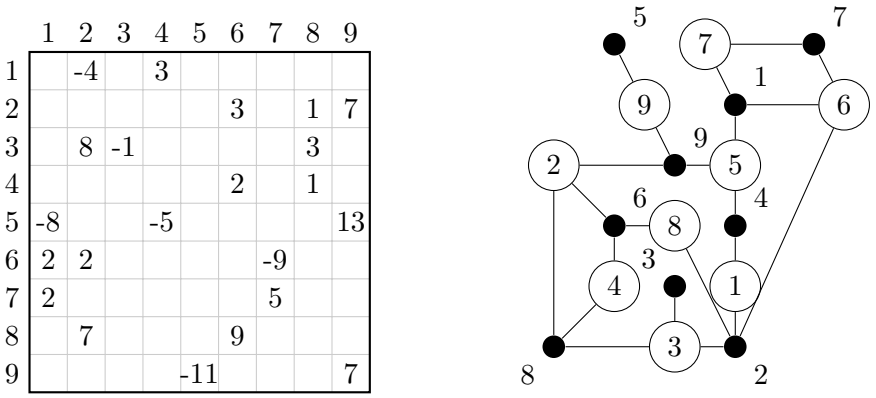


Figure III.3 A sparse matrix and its hypergraph representation. The white circles represent the nodes, i.e. the row, and the black circles are the nets, i.e. the interconnection columns.

A k-way partitioning of the hypergraph separates the graph into k disjoint subsets such that the interconnections between partitions, represented by the cut hyperedges, are minimised. This problem is NP-hard in reality [95]. In practice, heuristics are used to look for such a partitioning while enforcing a balance between the size of the constructed

[3] `http://glaros.dtc.umn.edu/gkhome/metis/metis/overview`
[4] `https://www.labri.fr/perso/pelegrin/scotch/`

subsets. In the ABCD-Solver, the hypergraph partitioner PaToH [32] is used to satisfy these criteria as introduced in [47]. In Figure III.4, we show the typical result of our example matrix partitioned uniformly *(Left)* and using a hypergraph partitioner *(Right)*. With the uniform partitioning, the partitions are interconnected on almost every columns, while when using the hypergraph partitioner, only 4 columns remain interconnected.

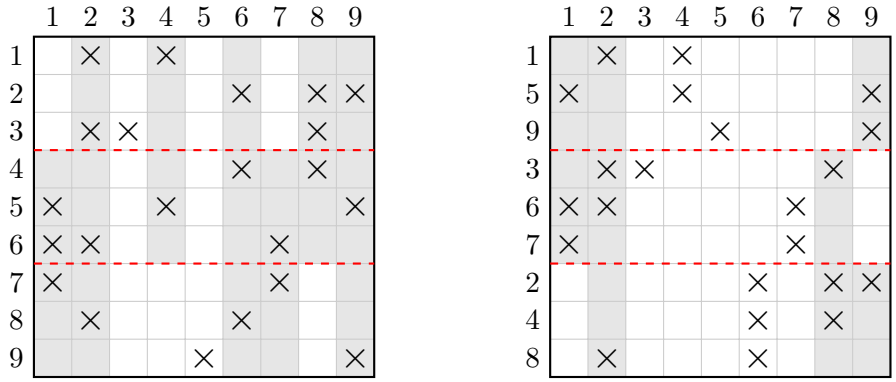


Figure III.4 A sparse matrix partitioned with *(Left)* a uniform and *(Right)* a hypergraph partitioner. Grey columns represent interconnections between partitions.

The target of the hypergraph partitioner is explicitly to reduce the number of interconnections between partitions, i.e. Objective 2. *As such, we expect this approach to give the smallest sizes of augmentation in ABCD.* However, the hypergraph partitioner only takes into account the structure of the matrix, the numerical properties should also be used for the iterative BC method.

GRIP: numerically aware partitioning In [126], the authors propose a graph partitioning method, called GRIP [126], which takes into account numerical information from the matrix in order to accelerate the convergence of the iterative BC. For this purpose, the row inner-product graph model of the matrix A is introduced. This graph, noted $\mathcal{G}_{RIP}(A) = (\mathcal{V}, \mathcal{E})$ is composed of the nodes $\mathcal{V}$ representing the rows of A (v_i for row r_i), and there is an edge (v_i, v_j) between node v_i and v_j if the inner product between rows r_i and r_j is nonzero. The nodes can be weighted, e.g. a weight of 1 per row or the number of nonzeros in the corresponding row, and each edge has a cost corresponding to the absolute value of the inner product between the linked rows

$$\forall v_i,\ w(v_i) = nnz(r_i),$$
$$\forall (v_i, v_j) \in \mathcal{E}, cost(v_i, v_j) = |r_i r_j^T|.$$

The method then uses the state-of-the-art partitioner METIS to perform a k-way partitioning of the weighted $\mathcal{G}_{RIP}$ to minimise the cutsize between partitions while maintaining a certain balance between weights of the partitions. The cutsize after partitioning corresponds to the sum of the inner products between rows belonging to separate partitions. The weight of a partition corresponds to the sum of the row weights inside the partition. E.g. if each row has a weight of 1 then the weight of the partition corresponds to the number of rows belonging to this partition in the matrix. After reordering according to the computed partitioning, we obtain normal equations were the large row inner products are close to the diagonal, see Figure III.5.

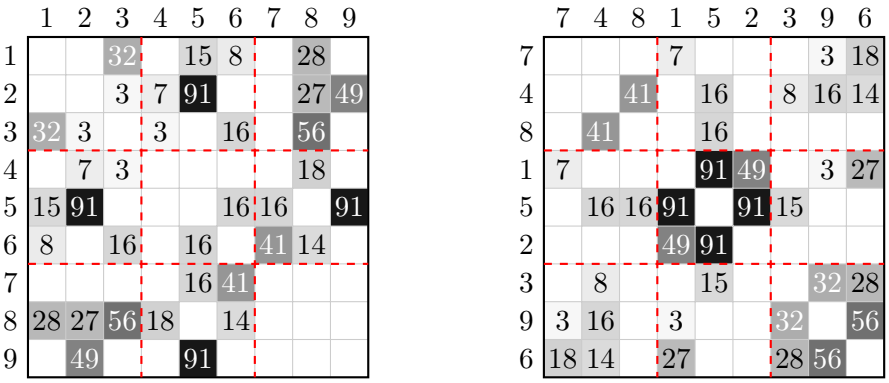


Figure III.5 Normal equations of a matrix partitioned with *(Left)* a uniform and *(Right)* the GRIP partitioner.

Furthermore, in the case where each row of the matrix A is scaled in 2-norm, the cost of each edge then corresponds to the cosines of the angle between the linked pair of rows. Therefore, minimising the cutsize after a k-way partitioning means minimising the sum of the cosines between partitions, and thus should help to explicitly reduce the angles between subspaces spanned by the partitions. As we have seen before, reducing these angles is ultimately expected to improve the spectrum of the iteration matrix H for block Cimmino. In [126], the authors show a reduced number of iterations with GRIP compared to other partitioners including PaToH. *Because this partitioning method explicitly targets the angles between subspaces spanned by the partitions, it is expected to give best convergence in the case of the iterative BC method.*

III.1.2.d Partitioning in practice

We now present numerical results from the application of the 2 partitioners PaToH and GRIP. Our goal here is not to study the detailed impact of the partitioning on the test

matrices. This has already been studied for unsymmetric square matrices before, see [47] and [126], and the conclusions are very similar for rectangular matrices. In appendix, results are given for the matrices `bayer01`, deltaX and sctap1-2r for both the iterative and augmented block Cimmino methods.

In this section, we have extracted all unsymmetric matrices from the SuiteSparse Matrix Collection with more than 1 000 rows. BC and ABCD are applied to each matrix with a number of partitions chosen such that each partition contains around 10 000 rows. Both the GRIP and the PaToH partitioners are used and we compare qualitatively the obtained results. Only matrices where the ABCD-Solver could finish computation were kept, and values from the same family of matrices were merged into the average value. In the end, we give results for

- *BC*: 28 Least-Squares, 143 underdetermined matrices, and 190 unsymmetric square,
- *ABCD*: 36 Least-Squares, 158 underdetermined matrices, and 235 unsymmetric square.

In Appendix B.1.1, we display the detailed difference between the 2 partitioners for each matrix in terms of number of iterations or augmentation size. In Table III.3, the number of cases where each partitioner is best, in terms of iterations for BC and augmentation size for ABCD, is given for each type of problem (least-squares underdetermined or square). These results confirm what we expected from the principle of each partitioner, i.e. the augmented block Cimmino method generally gives a smaller augmentation size with the PaToH partitioner, and the iterative block Cimmino method generally converges faster when using the GRIP partitioner.

Table III.3 Best partitioner in terms of iterations for the BC method and in terms of augmentation size for the ABCD method applied to resp. 361 (BC) and 429 (ABCD) classes of problems from the SuiteSparse Matrix Collection. The matrices are split in least-squares problems, and underdetermined or square matrices.

Best part.		**GRIP**	**Both**	**PaToH**
BC	LS	**14**	7	7
	under	**67**	29	47
	Square	**113**	45	32
ABCD	LS	11	0	**25**
	under	59	7	**92**
	Square	31	11	**193**

III.1.2.e Overlapping partitioning

Considering the iterative BC method as a DDM, a natural evolution for the construction of the subdomains is to introduce the notion of overlap. In our context, overlapping subdomains correspond to the construction of non-disjoint partitions.

In this approach, we start from an already existing disjoint partitioning of the sparse matrix, noted $A_i,\ i = 1, \ldots, p$. From this partitioning, we have the possibility to duplicate some rows from one partition to another, with the corresponding right hand side entries. Each row from the original matrix only appears once in each partition such that they keep full row rank. We focus here on BC for underdetermined systems of the form (II.10). After replication, we obtain the new system to solve

$$\widetilde{A}x = \begin{pmatrix} \widetilde{A}_1 \\ \widetilde{A}_2 \\ \vdots \\ \widetilde{A}_p \end{pmatrix} x = \begin{pmatrix} \widetilde{b}_1 \\ \widetilde{b}_2 \\ \vdots \\ \widetilde{b}_p \end{pmatrix}.$$

where $\widetilde{A}_i$ is a row-block which may intersect some rows with other partitions due to the replication. Each block of row $\widetilde{A}_i$ can be reordered into the form $\widetilde{A}_i = \begin{pmatrix} A_i \\ R_i \end{pmatrix}$ where A_i is the original partition and R_i is the set of duplicated rows.

Considering that the original matrix A has full row rank, so are the partitions $\widetilde{A}_i$ because no row can appear several times in a single partition. Also, the system stays consistent as the right hand side contains the duplications. These two conditions, and the right choice of a relaxation parameter, are sufficient for the BC classical iterations to converge to the solution of the original system (II.10), see [52]. As in Section II.2, in the case of the row version of BC, we then solve the system

$$\widetilde{H}x = \widetilde{\mathcal{K}} \text{ with } \begin{cases} \widetilde{H} &= \sum_{i=1}^{p} \mathcal{P}_{\mathcal{R}(\widetilde{A}_i^T)}, \\ \widetilde{\mathcal{K}} &= \sum_{i=1}^{p} \widetilde{A}_i^+ f_i. \end{cases}$$

We shall try now to understand intuitively the effect of duplicated rows between partitions on the spectrum of the iteration matrix $\widetilde{H}$. To explain this, let's consider the simple case of a matrix with 2 row-blocks. Let $\widetilde{A}$ be partitioned into 2 row-blocks $\widetilde{A_1}$ and $\widetilde{A_2}$ and let R be the replicated rows which are shared by both $\widetilde{A_1}$ and $\widetilde{A_2}$. The matrix is reordered as $\widetilde{A} = \begin{pmatrix} \widetilde{A}_1 \\ \widetilde{A}_2 \end{pmatrix}$ where $\widetilde{A}_1 = \begin{pmatrix} A_1 \\ R \end{pmatrix}$ and $\widetilde{A}_2 = \begin{pmatrix} R \\ A_2 \end{pmatrix}$. Let Q_i be an orthonormal

matrix whose columns span the range of $\widetilde{A}_i^T$. We define Q_i with first an orthogonalisation of the rows in the R set, followed by an orthogonalisation of the complementary rows A_i projected first onto the orthogonal complement of range of R^T. Then the two partitions can be represented by

$$Q_1 = \begin{pmatrix} W_1 & W_R \end{pmatrix} \text{ and } Q_2 = \begin{pmatrix} W_R & W_2 \end{pmatrix},$$

where W_R corresponds to the orthogonal columns spanning the range of R^T, and W_1, W_2 correspond to the orthogonal columns which complete the spanning of the range of $\widetilde{A}_i^T$. As explained before, the iteration matrix of the row-replicated Cimmino is given by

$$\widetilde{H} = \sum_{i=1}^{p} \widetilde{A}_i^+ \widetilde{A}_i = Q_1 Q_1^T + Q_2 Q_2^T, \tag{III.2}$$

and the non-zero eigenvalues of (III.2) are equal to the non-zero eigenvalues of matrix

$$\begin{pmatrix} Q_1 & Q_2 \end{pmatrix}^T \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} = \begin{pmatrix} I & Q_1^T Q_2 \\ Q_2^T Q_1 & I \end{pmatrix}, \tag{III.3}$$

where $Q_1^T Q_2$ are matrices whose singular values represent the cosines of the principal angles between the subspaces $\mathcal{R}(\widetilde{A}_1)$ and $\mathcal{R}(\widetilde{A}_2)$.

Considering the above representation of Q_i, we have

$$Q_1^T Q_2 = \begin{pmatrix} W_1^T \\ W_R^T \end{pmatrix} \begin{pmatrix} W_R & W_2 \end{pmatrix} = \begin{pmatrix} 0 & W_1^T W_2 \\ I_R & 0 \end{pmatrix}. \tag{III.4}$$

As the singular values of $Q_1^T Q_2$ are the cosines of the principal angles between $\mathcal{R}(\widetilde{A}_1)$ and $\mathcal{R}(\widetilde{A}_2)$, we see that as many cosines as the number of replicated rows are set to 1 using the replication technique. As in the case of a 2-blocks partitioning, the non-zero eigenvalues of the iteration matrix $\widetilde{H}$ correspond to the non-zero values of $1 \pm$ the singular values of $Q_1 Q_2^T$. For each replication, we thus ensure a non-zero eigenvalue of 2. The issue is then to know the content of the spectrum corresponding to what has not been replicated, i.e. the singular values of $W_1^T W_2$ in (III.4). This shifting property can be extended to a general case where the matrix has no special structure and partitions are interconnected in any way, which is what we generally encounter in practice. However, things are then getting more complex with respect to the spectral analysis of $\widetilde{H}$. The links between the partitions are like a spider web where a single row may connect more than 2 partitions. In this case, replicating a row still shifts eigenvalues, however it is

hard to understand (and control) the effect of what is left, and in particular implicit new links can be induced between previously non-linked partitions.

We now propose a method of replication based on the partitioning obtained using the previous GRIP partitioner. As a reminder, in this partitioner, we build a graph where the nodes are the rows, and where the edges are weighted with the absolute value of the inner product between 2 rows. The idea of our approach is to use the already existing graph $\mathcal{G}_{RIP}(A)$ for selecting the rows to be duplicated. Here, the expectation is that, when applying a replication technique targeting edges with a high inner-product weight, the gain from replication may be higher than the loss from the induced links between partitions. Several heuristics have been tested for this purpose. In practice, one of the most efficient at the moment, between these alternatives, is the *Duplication Method*, a very straight-forward approach.

In this method, we choose the edge with highest weight connecting 2 partitions. The rows at each end of this edge are then duplicated in the other partition. For example, in the graph with 3 partitions of Figure III.6, the edge (v_1, v_6) is the cut-edge with the highest weight. v_1 is then replicated into partition 2, and v_6 is replicated into partition 3. This does not however take into account any feedback from the induced links between partitions due to this replication. For example, v_6 is connected to v_3 inside the same partition, and to v_7 in a separate partition, but we do not consider an additional edge $(v_{6'}, v_3)$, $(v_{6'}, v_7)$. As a crude approximation of the effect from these replicated nodes, we only suppress the edge cost between v_1 and v_6. The algorithm then proceeds with the next edge of highest weight, and so on. Each replication is performed such that in every partition, each row can only appear once. Replication is stopped whenever all edges are under a certain threshold, or we have reached a maximum number of replications.

In Table III.5, we show the effect of an overlapping partitioning, obtained with the above described Duplication method, on the unsymmetric square matrices `memchip`, `bayer01` and `cage14`. The characteristics of these matrices, obtained from the SuiteSparse Matrix Collection[5] [39], are given in Table III.4. An initial partitioning of these matrices into 16 blocks of rows is computed using the GRIP partitioner [126]. The number of duplicated rows is bounded so as to increase the matrix by a maximum ratio of its size, and we vary this ratio from 0.1% to 20%. We observe the evolution of the number of iterations to convergence relatively to the disjoint partitioning case. In the case of `memchip` and `bayer01`, we observe a direct acceleration of the method with a decrease of the number of iterations by 20% with only 0.1% of duplicated rows. This decrease goes up to 77% for `memchip` with 1% of duplicated rows. In the case of `cage14`, the results

[5] `https://sparse.tamu.edu/`

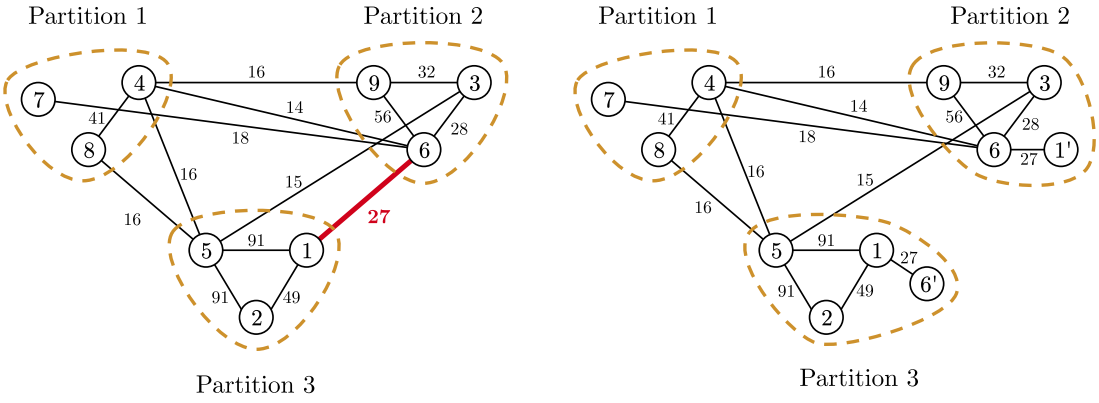


Figure III.6 Graph illustration of before and after replication of two nodes for a 3-way partitioning.

are negative as the number of iterations increases by up to 26% with 5% of duplicated rows. This might be explained by additional links between partitions induced by the duplicated rows which the heuristics does not consider at all. This work is still ongoing, with the hope to design a method taking into account these additional links that would enable a reduction in iterations as soon as rows are duplicated.

Table III.4 Characteristics of the test matrices. n: the order of the matrix, nnz: the number of nonzero values in the matrix.

Matrix	**m** ($\times 10^6$)	**n** ($\times 10^6$)	**elts per row**	**#Parts**	**Problem**
`bayer01`	0.06	0.06	4.76	8	Chemical Process
`cage14`	1.51	1.51	18.02	256	Directed Weighted Graph
`memchip`	2.71	2.71	4.93	512	Circuit Simulation

If we increase the number of duplications even further, we can observe however an acceleration of the convergence in all cases. Indeed, as the number of duplications increases, our method tend to the extreme case where each partition contains all rows and each projection computes the complete solution in 1 iteration. A trade-off must then be found between the right number of duplications to reduce sufficiently the set of small eigenvalues in the iteration matrix $\widetilde{H}$, together with the number of iterations and the computational complexity induced by the increase of the sizes of the partitions.

Table III.5 Impact of the Duplication Method on the number of iterations, relatively to the disjoint partitioning case. The number of duplications is bounded by a fixed percentage of the matrix size.

Matrix	**Original**	**Relative it. with DM replication**				
	it.	0.1%	1%	5%	10%	20%
`bayer01`	285	0.79	0.81	0.51	0.48	0.48
`cage14`	19	1.05	1.16	1.26	1.26	1.26
`memchip`	349	0.86	0.26	0.23	0.23	0.23

III.1.3 Alternative techniques for the augmentation

In Section II.3.1, we have introduced the augmented block Cimmino method. Starting from a matrix A partitioned in p blocks of rows A_i, an augmentation technique is applied to construct the augmentation block C so that the partitions $\overline{A}_i$ of the augmented matrix $\overline{A} = \begin{bmatrix} A & C \end{bmatrix}$ are mutually orthogonal, i.e

$$\forall i, j \in 1, .., p, i \neq j, \overline{A}_i \overline{A}_j^T = A_i A_j^T + C_i C_j^T = 0.$$

We define the augmentation procedure as the nonlinear function on matrices

$$\mathcal{F} \begin{array}{ccc} \mathbb{R}^{m \times n} & \longrightarrow & \mathbb{R}^{m \times \mathbb{N}^+} \\ A & \longrightarrow & C \end{array}, \tag{III.5}$$

Two main augmentation procedures have been developed in [47]. We first remind these 2 procedures, and then introduce a variant that can minimise the amount of extra columns introduced. To illustrate these techniques, we use the simple block tridiagonal example

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & & & \\ & A_{1,2} & A_{2,2} & A_{2,3} & \\ & & & A_{2,3} & A_{3,3} \end{bmatrix}. \tag{III.6}$$

III.1.3.a Normal equations duplication

The procedure $\mathcal{F}^{Cij}$ considers separately each couple of interconnected partitions. For each couple $A_i \in \mathbb{R}^{m_i \times n}$ and $A_j \in \mathbb{R}^{m_j \times n}$, the normal equations $C_{i,j} = A_i A_j^T \in \mathbb{R}^{m_i \times m_j}$ are computed. As some rows from the 2 partitions are often already orthogonal (mainly structurally), zero rows and/or columns appear in the normal equations. We filter out the zero columns from $C_{i,j}$, as well as in its transpose, to obtain $C_{i,j}^{filt}$ and $C_{j,i}^{filt}$. If $C_{i,j}^{filt}$

has the smallest number of column, the augmentation receives the blocks $C_{i,j}^{filt}/-I$, else we incorporate instead the blocks $-I/C_{j,i}^{filt}$ in the augmentation. Back to our example matrix, considering the filtered normal equations with smallest size are $C_{1,2}^{filt}$ and $C_{3,2}^{filt}$, then we obtain the augmentation block

$$\mathcal{F}^{Cij}(A) = \begin{bmatrix} C_{1,2}^{filt} & \\ -I & -I \\ & C_{3,2}^{filt} \end{bmatrix}.$$

Following this procedure, for each couple of interconnected partitions the augmentation gets a number of additional columns corresponding to the order of the smallest partition size. The size of the augmentation varies greatly depending on how many zero columns are filtered from the normal equations, and this is of importance not to increase too much the size of the matrix S, and to minimise the amount of extra computations in ABCD.

III.1.3.b Straight forward duplication

The procedure $\mathcal{F}^{Aij}$ consists in considering each couple of interconnected partitions separately. After finding the complete set of their interconnected columns, the corresponding interconnected blocks are duplicated with a minus sign for the second partition. On the example matrix (III.6), we obtain

$$\mathcal{F}^{Aij}(A) = \begin{bmatrix} A_{1,2} & \\ -A_{1,2} & A_{2,3} \\ & -A_{2,3} \end{bmatrix}.$$

With this procedure, for a single column in the original matrix interconnecting z partitions, the augmentation block receives $z(z-1)/2$ columns with a total of $z(z-1)$ non-empty sub-blocks inside. This is illustrated with a column $a = \begin{pmatrix} a_1 & a_2 & a_3 \end{pmatrix}^T$ with 3 interconnected partitions, giving the augmentation

$$\mathcal{F}^{Aij}(a) = \begin{bmatrix} a_1 & a_1 & \\ -a_2 & & a_2 \\ & -a_3 & -a_3 \end{bmatrix} = \begin{pmatrix} c^1 & c^2 & c^3 \end{pmatrix}. \tag{III.7}$$

Note that for both $\mathcal{F}^{Aij}$ and $\mathcal{F}^{Cij}$, we can use scaling matrices to better balance the numerical values in matrix C In the example (III.6), the part of the augmentation $\mathcal{F}^{Aij}(A)$ corresponding to the first 2 partitions could then be set to $\begin{bmatrix} A_{1,2}D \\ A_{2,1}D^{-1} \end{bmatrix}$ with D a

diagonal scaling matrix. This scaling is particularly important in the case of $\mathcal{F}^{Cij}$, as when using the normal equations, the scaling factors in the normal equations often vary by several orders of magnitude compared to the identity block added below in C. Of course, the diagonal matrix D can change for each couple of partitions, and also can be reduced to a simple scaling factor $\alpha \neq 0$ in some cases (such that $D = \alpha I$).

Note that none of the augmentation technique $\mathcal{F}$ requires a special structure for the matrix, and they all are directly applicable to more than 2 interconnected partitions. The main concern in the ABCD method is about the construction and solution of the Schur complement matrix S. The first concern is that S may be ill-conditioned. A good scaling and partitioning helps to improve this point. The second issue is with the size of S which may get large in some cases. Note that the augmentation block C is not guaranteed to be full rank, and this is actually not the case in general. Indeed, when more than 2 partitions are interconnected through a single column, then the added columns in C, when using $\mathcal{F}^{Aij}$ for instance, are linearly linked. In the above example with a column $a = \begin{pmatrix} a_1 & a_2 & a_3 \end{pmatrix}^T$ interconnecting 3 partitions, the augmentation C is as in (III.7). In this block, we can see that $c^1 + c^3 - c^2 = 0$, so that these augmentation columns are linearly dependent. It can be shown that the rank of the added block for the $\mathcal{F}^{\mathcal{A}_{)|}}$ augmentation technique is always equal to the number of interconnected partitions minus 1. As mentioned above, the size of the augmentation may be very large, and in some cases even larger than the size of the original system itself. Using the augmentation $\mathcal{F}^{Aij}$ on a column with z interconnected partitions induces a number of columns in the augmentation of the order of $\mathcal{O}(z^2)$.

III.1.3.c Full rank augmentation

Here we introduce a new augmentation procedure $\mathcal{F}^{FR}$, that ensures that, for each interconnected column a, the added block $\mathcal{F}^{FR}(a)$ has full column rank and the number of added columns is of the order $\mathcal{O}(z)$. In the case of 2 interconnected partitions, $\mathcal{F}^{FR}$ gives the same augmentation as $\mathcal{F}^{Aij}$, the difference appears with more interconnections. To illustrate the differences between the three augmentation techniques, we use the matrix

$$A = \begin{bmatrix} A_{1,1} & A_{1,2}^{(1)} & & & & A_{1,2,3}^{(1)} \\ & A_{1,2}^{(2)} & A_{2,2} & A_{2,3}^{(2)} & & A_{1,2,3}^{(2)} \\ & & & A_{2,3}^{(3)} & A_{3,3} & A_{1,2,3}^{(3)} \end{bmatrix}. \tag{III.8}$$

The principle of $\mathcal{F}^{FR}$ relies on the construction of a lower block tridiagonal augmentation which prevents the use of redundant information. When building the augmentation for a given column with z interconnected partitions, the first column in C receives the

duplicates of the parts corresponding to *all* the interconnected partitions with a minus sign for all partitions except the first. Then another column is added, incorporating again the parts corresponding to all partitions *except the first*, and with a minus on all except for the second partition, which now gets a scaling factor of 2. And so on, until there are only 2 partitions left, in which case the first gets the factor $(z-1)$ and the other gets a minus sign. The complete algorithm for $\mathcal{F}^{FR}$ is detailed in Algo. 9. For a single column with z interconnected partitions, the number of columns in the augmentation is $z-1$ and the number of non-empty blocks is $\frac{(z+2)(z-1)}{2}$. Thus, in all cases, we have $size(\mathcal{F}^{FR}) \leq size(\mathcal{F}^{A_{ij}})$. Additionally, this new augmentation technique ensures that, for each interconnected column, the corresponding block in the augmentation C is full rank, thus its name.

Algorithm 9 Algorithm to compute $\mathcal{F}^{FR}(A)$

Input: $A \in \mathbb{R}^{m \times n}$ partitioned in p row blocks.

$Adj(p,c) = \begin{cases} 1, & \text{if } c \text{ is a nonzero column of partition } A_p \\ 0, & \text{else} \end{cases}$,

$Interc(c) = \sum_{i=1}^{p} Adj(i,c)$,

$Count = zeros(n)$,

for all $i \in \{1,\ldots,p\}$ **do**

  for all $c \in \{1,\ldots,n\}$ s.t. $Adj(i,c) \neq 0$ and $Interc(c) - Count(c) > 1$ **do**

    $Count(c)$++,

    $scale = zeros(m)$, $scale(A_i) = Count(c)$,

    for all $j \in \{i+1,\ldots,p\}$ s.t. $Adj(j,c) \neq 0$ **do**

      $scale(A_j) = -1$,

    end for

    $C = diag(scale)A(:,c)$.

  end for

end for

Taking the example matrix (III.8), when using the 2 augmentations $\mathcal{F}^{A_{ij}}$ and $\mathcal{F}^{FR}$, we get the complete augmentation blocks

$$\mathcal{F}^{Aij} = \begin{bmatrix} A^{(1)}_{1,2} & & A^{(1)}_{1,2,3} & A^{(1)}_{1,2,3} & \\ -A^{(2)}_{2,1} & A^{(2)}_{2,3} & -A^{(2)}_{1,2,3} & & A^{(2)}_{1,2,3} \\ & -A^{(3)}_{2,3} & & -A^{(3)}_{1,2,3} & -A^{(3)}_{1,2,3} \end{bmatrix}, \quad \mathcal{F}^{FR} = \begin{bmatrix} A^{(1)}_{1,2} & & A^{(1)}_{1,2,3} & \\ -A^{(2)}_{2,1} & A^{(2)}_{2,3} & -A^{(2)}_{1,2,3} & 2 \cdot A^{(2)}_{1,2,3} \\ & -A^{(3)}_{2,3} & -A^{(3)}_{1,2,3} & -A^{(3)}_{1,2,3} \end{bmatrix}.$$

As was done for the other two augmentation procedures, it is also possible to scale in some way the augmentation blocks. However, partitions are no longer augmented by pairs of blocks, and setting appropriate scaling factors so as to maintain the orthogonality between the augmented partitions is more complicated. For the first interconnected block

in the augmentation, a scaling matrix D is applied to the right, then on the i-th added column we apply to the right of the first non-zero block the matrix $D + (i-1)D^{-1}$. As for the minus signs they all have to be replaced by $-D^{-1}$ on the right of the blocks. Taking the example of a column

$$a = \begin{bmatrix} a_1 & \dots & a_p \end{bmatrix}^T,$$

interconnecting all p partitions. The augmented column scaled with the diagonal matrix D, would then be

$$\left[\begin{array}{c|cccc} a & \mathcal{F}^{FR}(a) & & & \end{array}\right] = \left[\begin{array}{c|cccc} a_1 & a_1.D & & & \\ a_2 & -a_2.D^{-1} & a_2.(D+D^{-1}) & & \\ a_3 & -a_3.D^{-1} & -a_3.D^{-1} & \ddots & \\ \vdots & \vdots & \vdots & & \\ a_{p-1} & -a_{p-1}.D^{-1} & -a_{p-1}.D^{-1} & \dots & a_{p-1}.[D+(p-1)D^{-1}] \\ a_p & -a_p.D^{-1} & -a_p.D^{-1} & \dots & -a_p.D^{-1} \end{array}\right].$$

This type of augmentation respects the orthogonality between partitions. $\forall i,\ j \in \{1,\dots,p\},\ i<j$, we call $\overline{a}_i$ and $\overline{a}_j$ the augmented partitions, then we have

$$\begin{aligned} \overline{a}_i\overline{a}_j^T &= a_i a_j^T + (i-1)a_i D^{-1}D^{-1}a_j^T - a_i[D+(i-1)D^{-1}]D^{-1}a_j^T \\ &= (1-1)a_i I a_j^T + [(i-1)-(i-1)]a_i D^{-2}a_j^T = 0. \end{aligned} \tag{III.9}$$

III.1.3.d Augmentation in practice

As in [47], we investigate the size of the augmentation using ABCD on a wide range of matrices. For this purpose, we have considered all the matrices with more than 1 000 rows or columns from the SuiteSparse Matrix Collection[6]. We distinguish between overdetermined, underdetermined and unsymmetric square matrices. We apply the augmentation method giving the smallest augmentation in each case. For all matrices, the size of the augmentation blocks relatively to the larger dimension of the original matrix, is displayed in Figure III.7 sorted by increasing order. Matrices from the same family, e.g. the combinatorial problems `GL7d11` to `GL7d26`, gave similar relative augmentation sizes, and were merged together in one class with the average augmentation size. In the end, 38 classes of least squares problems, 153 classes of underdetermined systems, and 241 classes of unsymmetric square matrices are displayed. We observe similar behaviours

[6] `https://sparse.tamu.edu/`

for the 3 types of problems. A third of the matrices shows augmentation sizes below 5% of the dimension of the original problem, which confirms for full rank matrices the results from [47], where the an augmentation below 5% is considered to be reasonnable. More than 70% over all classes show an augmentation size under 20% of the original size. In this case, whenever the constructed S is still sparse, the augmentation size is potentially still acceptable for ABCD when compared to a direct solver, but this has to be verified in practice.

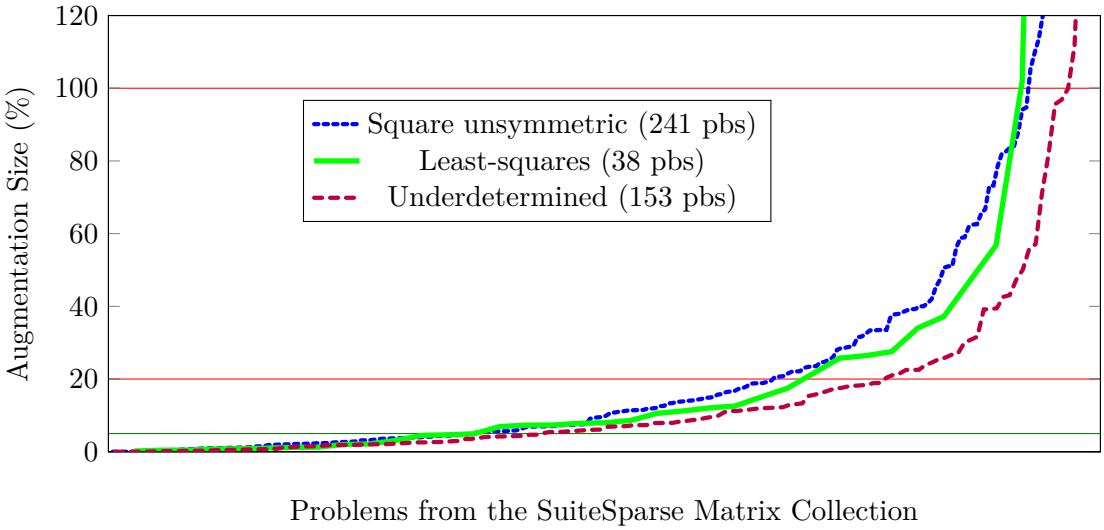


Figure III.7 Distribution of the augmentation size, relative to the size of the original system, for 38 least squares *(green plain line)*, 153 underdetermined *(dashed red line)* and 241 unsymmetric square *(dotted blue line)* problems, sorted by increasing augmentation size. The 5%, 20% and 100% thresholds are drawn.

In order to compare the new augmentation $\mathcal{F}^{FR}$ with the previous approach $\mathcal{F}^{C_{ij}}$, we apply these augmentations on the same classes of matrices. The augmentation $\mathcal{F}^{A_{ij}}$ is not used as it is guaranteed to give an augmentation larger or equal to $\mathcal{F}^{FR}$. In table III.6, we give the number of matrices which gave the smallest augmentation size using $\mathcal{F}^{FR}$ or $\mathcal{F}^{C_{ij}}$. We distinguish the type of problems as least-squares, underdetermined, and unsymmetric square. In the case of rectangular matrices, $\mathcal{F}^{C_{ij}}$ generally gives smaller augmentations, while $\mathcal{F}^{FR}$ is best for the unsymmetric square matrices.

The best augmentation method depends on the structure of the matrix. As illustrated in Figure III.8, $\mathcal{F}^{C_{ij}}$ is more suited for dense rows, which induce large interconnection blocks, while $\mathcal{F}^{FR}$ is more suited for dense columns, where the normal equations fill-in

Table III.6 Best augmentation method in terms of size for different classes of problems from the SuiteSparse Matrix Collection: 36 Least-Squares, 161 underdetermined and 241 unsymmetric square matrices.

Smallest aug.	$\mathcal{F}^{C_{ij}}$	**Both**	$\mathcal{F}^{FR}$
LS	29	1	7
under	83	11	57
Square	22	25	193

and the number of interconnected partitions is large. In reality, matrices display a mix of various structures. In Appendix B.1.2, we give more details on these 2 extreme cases and also give results from the application of ABCD for a set of matrices from the SuiteSparse Matrix Collection depending on the used augmentation technique. In the following sections, we use the augmentation method with smallest size for each problem.

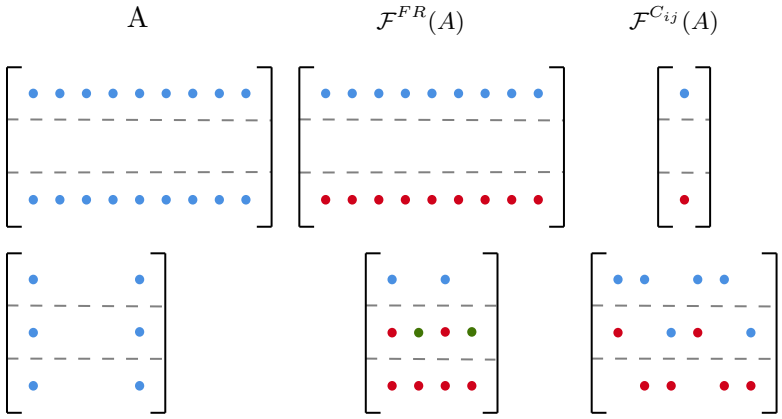


Figure III.8 Two matrices partitioned in 3 blocks of rows, with respectively dense rows and dense columns. The corresponding augmentation blocks obtained with $\mathcal{F}^{FR}$ and $\mathcal{F}^{C_{ij}}$ are displayed.

III.2 Parallel implementation scheme

> I do not fear computers. I fear the lack of them.
>
> Isaac Asimov

With preprocessing, the stage is set for the actual parallel implementation. In this section, we first introduce all practical aspects of the parallelism scheme we use, then the implementation choices made in order to decrease the complexity in computation, communication, and memory. All of these 3 aspects are essential to get an efficient solver in modern supercomputers. Finally, putting everything together, we demonstrate the efficiency of the ABCD-Solver through numerical experiments.
The three following sections and the corresponding numerical experiments are derived from the paper [45], published in the proceedings of the Parallel Computing 2019 conference. In this work, we combine methods to get a good trade-off between an overall load-balancing amongst the distributed tasks, together with a reduced communication.

III.2.1 Hybrid parallelism

The parallelism of the ABCD-Solver is based on MPI [71] and OpenMP [37]. Both the BC and the ABCD methods perform the same preprocessing steps following the principles presented in the beginning of this chapter. This preprocessing step is applied to the underdetermined matrix A. Firstly, after scaling the system, we partition the matrix. Secondly, the basic idea is to distribute each partition A_i to one process, called *master*, which builds the corresponding projection system (II.16). Thirdly, these symmetric systems are solved using the sparse direct solver MUMPS[7] [6] so as to be able to compute the projection on the subspace spanned by the block of rows in the partition. This direct solver uses the well-known multifrontal method and performs three steps: analysis (preprocessing, estimation of workload and memory), LDL^T factorisation, and finally solve (forward elimination and backward substitution), see Section I.3.2. Analysis and factorisation must only be performed once, while one solve is performed each time we need to compute the projection at every iterations. These local projections are then summed through non-blocking point-to-point communications between masters. The amount of data communicated between 2 masters is equal to the number of shared columns, i.e. *interconnections* [133]. Additionally in ABCD, the matrix S is built in an embarrassingly parallel way, see Section 5.4 in [133] for the detailed description. We

[7] `http://mumps-solver.org/`

use the fact that S is the restriction of W, and each column of W corresponds to the independent projections of canonical vectors, thus implying only a small subset of the masters. Then S is given in distributed form directly to MUMPS for a parallel solve on the global communicator (see [47] and [133] for the details of the construction and solution of S). S being identical for both row and column partitionings, no modification to this computation was needed compared to the previously published versions.

The ABCD Solver is a *hybrid scheme*, in the sense that the method is block-iterative, with the use of a direct solver for each subproblem defined by the partition, but also in the sense that it offers a pseudo-direct approach, with convergence in 1 iteration, through the augmentation technique. The solver also implements a *hybrid parallelism* with several levels of parallelism exploited at the same time

1. the projections are independent and computed in parallel,
2. the MUMPS solver introduces two levels of parallelism: through the exploitation of its *elimination tree* and through the factorisation of large frontal matrices using parallel linear algebra *dense kernels*.

Figure III.9 depicts the parallelization scheme, including the fundamental steps of the algorithm, for the BC and ABCD algorithms.

In addition to the multiple levels of parallelism exploited in the ABCD-Solver, the management of data is also optimised as introduced in [133]. In particular, the possibility to compress the data maintained locally on each process is crucial, particularly to reduce MUMPS complexity via a reduction of the size of the projection systems. Also important, is the possibility to update only the part of the solution corresponding to the local partitions in the column version of the block Cimmino methods. In the row version, the same is true concerning the residuals.

III.2.2 Improving the scalability of the ABCD-Solver

The possibility to give exactly one partition to each process is only the basic idea. Depending on the number of masters and the number of partitions, there are various possibilities for scheduling the computations. In the following sections of the article, we propose and study two different approaches. In the first setting, the number of partitions is higher than the number of masters, and a master process owns a group of partitions. In Section III.2.2.a, we propose a new algorithm that aims to group partitions on each master so as to minimise the overhead of communication between masters, and at the same time maintain the load balance across masters. In the second setting, there are more processes than masters. We use the extra processes, not yet assigned, to act as workers to help the master MPI processes to parallelize the computation further. In

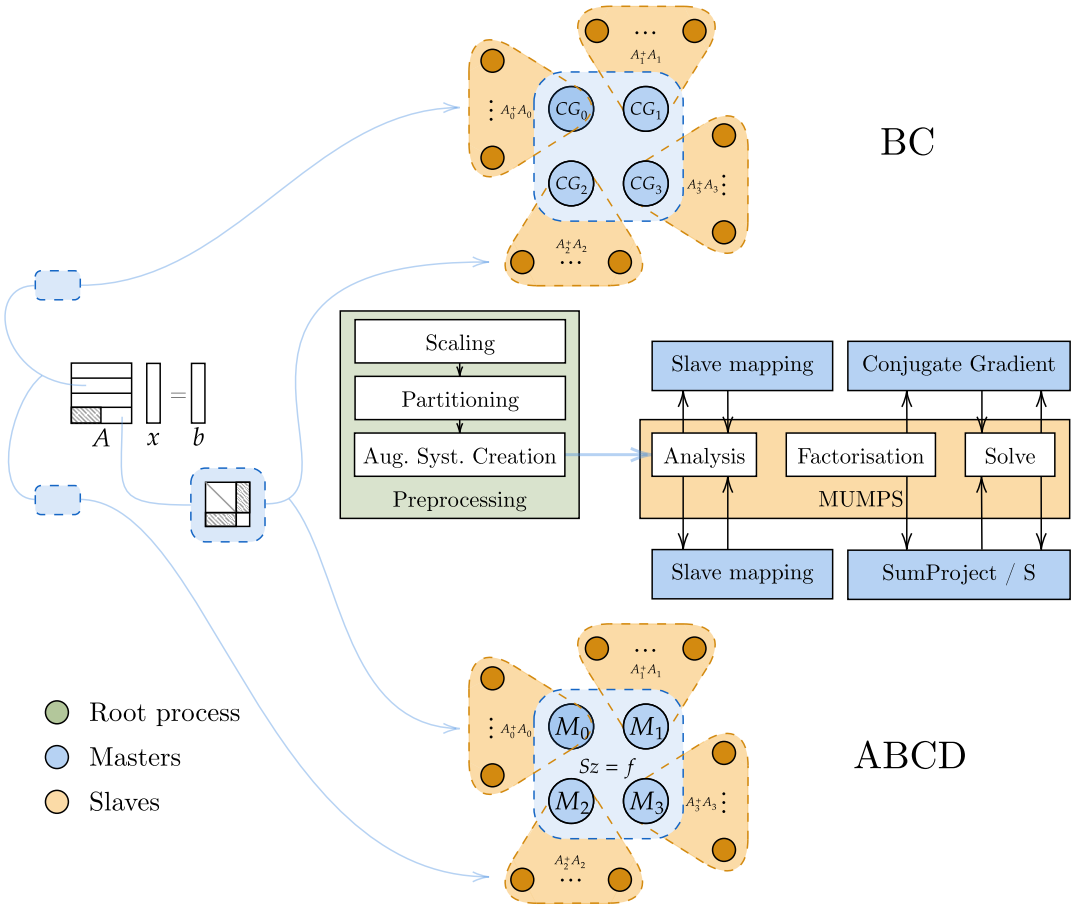


Figure III.9 Hybrid parallelisation scheme of the ABCD-Solver.

Section III.2.2.b, we introduce a new method to assign the processes, masters and workers, in the physical computing resources to decrease the communication overhead both within and between master-workers groups.

In the ABCD Solver, we distinguish three types of communications [133]: the *intra-communication* within master-workers group which only occurs when computing a projection using MUMPS; the *inter-communication* between masters which occurs when summing the projections; finally in ABCD, the *global communication* used at first when solving the system based on S with all available processes.

III.2.2.a Load balancing: distribution of partitions

In the first setting, the number of masters is assumed to be less than the total number of partitions. In such a case, the idea is to assign groups of partitions to the masters, who construct one single block diagonal system, made with the projection systems from the various partitions. This block diagonal system is then solved using MUMPS as before. Here, the goal is to distribute the partitions to the masters with the right trade-off between balancing the weight of the local groups of partitions over all master processes, and minimising the overhead in communication between masters.

Remark: At extreme scales, this scheme can be further enforced by picking a number of masters much lower than the number of partitions, and also lower than the total number of available processes, together with the idea to exploit the extra processes as workers assigned in clusters to the masters. This can be of interest depending on the architecture of the target machine. The distribution of partitions then becomes a kind of data agglomeration in situations where the execution on a limited number of processes is more efficient, see Section I.3.1.

As proposed in [133], the first approach we consider is called "*Greedy*". This distribution results in an optimal distribution of the partitions over all masters, in terms of balancing the weight of the groups of partitions. Globally, balancing the weights of local sets of partitions is not the only concern, one should also consider the overhead from intercommunication between masters resulting from the distributed group of local projections. Point-to-point communication is then effectively performed on values corresponding to interconnections between different masters, see the details in [47]. Therefore, the best distribution of the partitions should find the right trade-off between minimising this communication, i.e. decreasing the number of interconnected columns between master processes, and balancing the workload over these masters in order to achieve minimum parallel execution time.

We propose a new approach, called "*Comm-μ*", which is based on this principle, see Algo. 10. The algorithm first creates a graph $\mathcal{G}$. The nodes of $\mathcal{G}$ are the partitions weighted by their respective size. There is an edge between two nodes if the corresponding partitions are interconnected, i.e. they share a nonzero column, and the cost of that edge equals the number of such columns. In the final step, we partition $\mathcal{G}$ using the multilevel graph partitioning tool METIS [89] to minimise the number of interconnections between the groups of partitions for each master, with a parameter μ that permits a certain imbalance in the accumulated weight over the groups of partitions.

Algorithm 10 Algorithm for the distribution of partitions minimising communications while keeping a balance over the weight of partitions.

Input: w: weight of the partitions
Input: $colIndex$: indices of the non-empty columns for each partition
Input: $nb\_masters$, nb_parts, μ: imbalance threshold
1: $AdjacencyMatrix = zeros(nb\_parts, n)$
2: $AdjacencyMatrix(p, colIndex(p)) = 1$
3: **for** $p_1, p_2 \in \{1..nb\_parts\}$ **do**
4: $\quad Interactions(p_1, p_2) = size\left(colIndex(p_1) \bigcup colIndex(p2)\right)$
5: **end for**
6: Create graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ using $AdjacencyMatrix$ and $interactions$
7: METIS($\mathcal{G}$, $nb\_masters$, μ)

III.2.2.b Placement of masters and workers

In the second setting, we assume more MPI processes than masters. Partitions, or groups of partitions, are assigned to the masters, and the extra processes with no partitions are associated with the masters in clusters, as worker processes that contribute to the parallel computations in MUMPS.

Hierarchy of the computing architectures

The ABCD Solver is designed to solve large systems on distributed memory architectures where the computing resources are hierarchically structured with nodes and cores, as introduced in Section I.2.1. When launching our distributed solver, we specify a certain number of MPI processes per node which are allocated and placed on the system architecture in a predetermined way by the batch system. Depending on the situation at runtime, we need to decide which processes are given the role of master or worker in order to minimise the total overhead of the communication between masters (inter-communication) on the one hand, and inside master-workers groups (intra-communication) on the other hand. This process is composed of three steps. Firstly, processes are chosen to be masters, and in a second phase we determine the number of workers for each master. In principle, the anticipated number of flops (given by the MUMPS analysis for each partition) is used to assign more workers to masters with higher workload. The assignment of workers optimally balance the average workload per process with a greedy algorithm. Thirdly, we choose the workers for each master depending on their respective placement in the architecture.

Two opposite approaches emerge for the first step. We can place masters close to each other to accelerate inter-communication (by exploiting the shared memory access

available on each node), and we refer to this approach as *Compact*. Alternatively, we can spread the masters over the nodes in order to fit the master-workers group together on a same node. This simultaneously improves intra-communication and decreases concurrent access to memory by masters. We refer to this approach as *Scatter*.

Architecture aware placement of masters and workers

The approach first implemented in the ABCD Solver is *Compact*, see [133]. In this approach, the first ranks of MPI define the masters and the rest of the processes are assigned in sequence to them as workers, depending on their rank. Although this approach minimises the inter-communication, both the intra-communication as well as the sequential calls to dense kernels, known to be memory-bound, are slowed down due to concurrent memory access among masters.

In this case, spreading masters over the nodes can be better, we call this approach *Scatter*. This was the placement privileged in Section I.3.3 when using MUMPS on the coarse grid of a multigrid scheme at extreme scale. Note that we propose here a "manual" implementation of this approach, but, in the future, this implementation could be replaced by existing architecture aware mechanisms [87]. We define two algorithms, respectively for the placement of the masters and the workers. The principle of these algorithms is simple

- To place the masters, we first gather information to know which node each process runs on. We then assign one master per non-filled node in a zig-zag fashion, starting from the largest node to the smallest then alternating.
- To place the workers, we first sort the masters in descending number of desired workers. Then for each master, we place its workers in the same node and, when the node is filled, we group the remaining workers in other nodes as close to each other as possible.

The pseudo-code of these algorithms is given in Appendix B.2. In Figure III.10, we illustrate the effect of the Compact and Scatter approaches on a toy example. We partition a matrix in 3 partitions solved using block Cimmino with 12 processes. We define 3 masters each with 3 workers and launch the solver on 3 nodes each with 4 processes.

In Section III.2.3, the ABCD-Solver is applied to various matrices. We demonstrate an improvement in parallel performance in both settings, first using the communication reducing distribution of partitions, then deciding masters and workers depending on the computing architecture.

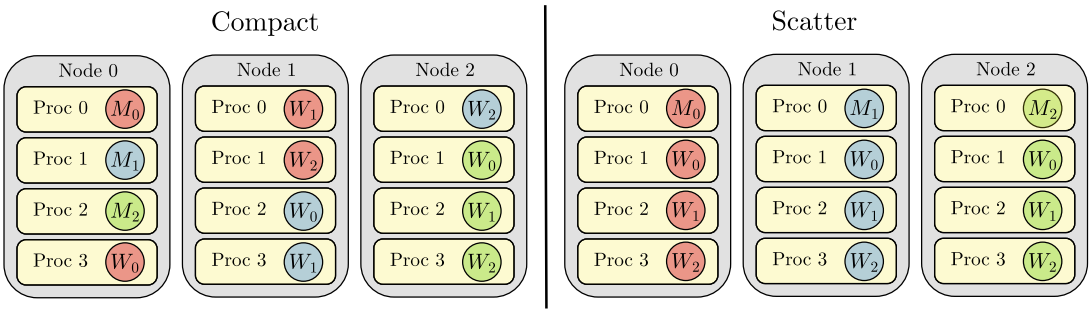


Figure III.10 3 nodes with 4 processes on each and we have 3 masters with 3 workers each. M_i corresponds to the master i and the S_j of the same colour is its worker j. *(Left)* Compact scheme, *(Right)* Scatter scheme.

III.2.3 Parallel experiments

In this section, we show parallel distributed runs for the BC and ABCD methods on the unsymmetric and rectangular matrices described in Table III.7, which were extracted from the SuiteSparse Matrix Collection[8] [39]. This set of sparse matrices is freely accessible and corresponds to problems from real-world applications. The number of partitions is set so that in these cases, the size of a partition is of the order of 10 000. Given the number of partitions, we exploit the 2 methods for permutation and partitioning described in Section III.1.2. For the iterative BC solver, we use the partitioning inducing the smallest number of iterations, and for the pseudo-direct ABCD solver, we use the combination of partitioning and augmentation technique giving the smallest augmentation size. Our experiments are carried on Meggie, a supercomputer at the RRZE at FAU in Erlangen, Germany[9]. Meggie is a Megware system with Broadwell Intel Xeon E5-2630 v4 processors at 2.2GHz. Each of its 728 compute nodes is a 2-socket system with 64 GB memory, where the 10 cores of each processor constitute a separate NUMA (non-uniform memory access) domain. Meggie uses the Intel OmniPath interconnect.

Firstly, we demonstrate parallel efficiency induced by the communication reducing load balancing introduced in Section III.2.3.a. Secondly, to have an efficient parallel ABCD method, we show that the Schur complement must be solved with a limited number of processes, as S is of small size whenever this approach is effective, in Section III.2.3.b. Thirdly, as presented in Section III.2.3.c, a good placement of the master and worker processes is used to decrease the execution time of the iterative and pseudo-direct methods.

[8] `https://sparse.tamu.edu/`
[9] https://www.anleitungen.rrze.fau.de/hpc/meggie-cluster/

Table III.7 Characteristics of the test matrices. n: the order of the matrix, nnz: the number of nonzero values in the matrix.

Matrix	m ($\times 10^6$)	n ($\times 10^6$)	elts per row	#Parts	Problem
deltaX	0.07	0.02	3.61	4	Counter Example
image_interp	0.24	0.12	2.97	16	Computer Graphics/Vision
LargeRegFile	2.11	0.80	2.34	64	Circuit Simulation
shar_te2-b2	0.20	0.02	3.00	32	Combinatorial Problem
neos	0.48	0.52	3.19	32	Linear Programming
sctap1-2r	0.03	0.06	6.46	8	Linear Programming
stat96v3	0.03	1.11	98.04	4	Linear Programming
bayer01	0.06	0.06	4.76	4	Chemical Process
cage14	1.51	1.51	18.02	256	Directed Weighted Graph
memchip	2.71	2.71	4.93	256	Circuit Simulation

The experiments in Sections III.2.3.a and III.2.3.c use the same settings as those presented in [45] which details are recalled. Finally, in Section III.2.3.d, the parallel scalability is studied on larger test cases with an extensive comparison with the state-of-the-art direct solvers MUMPS, and QR-MUMPS.

III.2.3.a Communication reducing distribution of partitions

Here we use the distribution technique introduced in Section III.2.2.a to apply the BC method on the square matrices `bayer01`, `cage14`, and `memchip`. We apply BC in parallel using 128 MPI processes spread over 16 distributed nodes. Each MPI process is attached to 2 OpenMP threads. Matrix `bayer01` is partitioned in blocks of rows, and matrices Hamrle3 and `memchip` are partitioned in blocks of columns, using the partitioning method giving the smallest number of iterations for convergence.

In the first setting, each matrix is partitioned into 1 024 blocks and partitions must be grouped before being distributed to the master processes. Note that this number of partitions would be considered too high for a normal use of the BC method on matrices of this size, see Table III.7. Here, because we want to analyse the communication scheme, we set the number of partitions to an exaggerated fixed size such that partitions are highly interconnected in general, and thus the amount of communications required for the sum of projections between masters is increased. We use the three matrices `bayer01`, `cage14`, and `memchip` because they illustrate the main behaviours in terms of computation and communications.

The experiments are then conducted using 3 distribution of partitions. First, the greedy algorithm (*Greedy*) is used, which goal is to only balance the workload over the

groups of partitions. Then, the communication reducing algorithm is applied, which takes into account the interconnections between partitions to reduce communications while keeping a balance ratio μ between workloads associated to the sets of partitions. We use 2 values for this parameter, $\mu = 1\%$ (*Comm-1*) and $\mu = 10\%$ (*Comm-10*). Results are reported in Table III.8. The column 'Com. col%' of Table III.8 reports the total communication volume, equal to the number of interconnected columns, normalised with respect to the greedy method. The table also reports execution times for the factorisation as well as the imbalance ratio between the slowest and average factorisation times over all masters. Finally, the table gives the BCG execution time and iterations for the BC method. As seen in the table, for BC, the proposed methods *Comm-1* and *Comm-10* achieve around 89%, 58%, and 60% reduction in the total number of exchanged columns for `bayer01`, `cage14`, and `memchip` respectively. This improvement in turn leads to faster parallel execution of BCG for the matrices `bayer01`, and `memchip`. On the other hand, for `cage14`, despite the reduction in the amount of communications, the execution time increases because the overhead of load imbalance on the factorisation absorbs the gain from the minimisation of communication. This is partly explained because of the fast convergence in this case, since the amount of communication (performed once per iteration) is then low. In the ABCD-Solver, the workload induced by each partition is crudely estimated by its number of rows for the distribution of partitions, which explains why the imbalance does not increase exactly according to the imbalance ratio μ. Additionally, our experiments tend to show that a larger value for the parameter μ has a limited effect on the reduction of the total size of communication and of the execution time.

The overall execution time of the BC benefits from the use of the communication reducing distribution of partitions with a reasonnable parameter, e.g. $\mu = 1$, in cases where the time spent in the direct solver is not predominant compared to communication. In the case of the ABCD method, only one iteration is required for the convergence, thus the communication between masters to compute the sum of projections is only performed once. Also, with such a high number of partitions, the size of the Schur complement would explode and most of the execution time would be spent on its solution. Applying a special technique for the distribution of partitions for the ABCD method does not make sense.

III.2.3.b Parallel solution of the Schur complement

The block Cimmino approach is a hybrid scheme where we use a direct solver to solve subsystems defined by the partitions, with a divided complexity in terms of computation

Table III.8 Impact of the distribution of partitions on the execution times. All runs were performed with 1024 partitions and 128 MPI processes with 2 threads on 16 nodes. (Com. col %: Normalised column reduction values with respect to the Greedy algorithm. tot: Total time in seconds. Fact. imb. %: ratio of maximum over average factorisation times. BCG it: Number of iterations required for convergence. Sol. time: Total solution time in seconds)

Matrix	**Algo.**	**Com.**	**Fact.**		**BCG**	
		col.%	*tot.*	*imb.*	*tot.*	*it.*
`bayer01`	Greedy	100	0.85	120	18.80	3647
	Comm-1	11	0.87	122	**16.20**	3571
	Comm-10	11	0.78	105	16.80	3570
`cage14`	Greedy	100	2.14	126	**5.26**	17
	Comm-1	42	3.44	192	8.26	17
	Comm-10	42	3.09	202	7.42	17
`memchip`	Greedy	100	1.21	145	89.40	537
	Comm-1	39	0.96	119	**89.00**	536
	Comm-10	40	1.04	133	91.00	537

as well as in terms of memory. However, there is a downside to this effect. As the size of the subsystems decrease, so does their granularity and this is a limit for parallel scalability. Once a certain limit of granularity is reached there is often positive effect on execution times in limiting the number of processes used by a direct solver. We have shown this effect in Section I.3, in the case of a direct solver used for the solution of the coarse grid problem in a multigrid scheme, for example.

The ABCD method shows a good behaviour in cases where the matrix S is not too large, nor dense. When the direct solver is applied to solve this system with a large number of processes, the granularity of the subproblems considered by the direct solver crumbles. In Table III.9, we show the execution times corresponding to the solution of the Schur complement system for ABCD applied to the matrices from Table III.7 (with the corresponding number of partitions), depending on the number of processes used. The number of processes varies from half to 32 times the number of partitions, with each process attached to 2 threads. We observe that the fastest runs are always obtained with a number of processes equal or lower to the number of partitions. For example in the case of the matrix `neos` (split in 64 partitions), if we use 2 048 processes instead of 64, the solution time is increased by a factor 6. Thus, in the rest of this chapter, while the matrix S is built in an embarrassingly parallel way involving all processes [133], we solve

the Schur complement system only with the master processes, which number is upper bounded by the number of partitions.

Table III.9 Execution time for the solution of the Schur complement system depending on the number of processes used. The number of processes is displayed relatively to the number of partitions used for each matrix.

Matrix	**#Parts**	Aug.	$\frac{\#MPI}{\#Parts}$	$\frac{1}{4}$	$\frac{1}{2}$	1	2	4	8	16	32
`deltaX`	4	5 154	solve time (s)	1.39	1.15	**1.09**	1.10	1.14	1.08	1.09	1.07
`image_interp`	16	4 406		0.42	**0.27**	0.33	0.40	0.47	0.86	0.86	3.22
`neos`	32	42 677		46.30	53.03	**40.62**	51.39	76.11	161.86	247.62	–
`sctap1-2r`	8	102		**0.009**	0.010	0.010	0.013	0.019	0.023	0.034	0.071
`stat96v3`	4	2 817		**0.53**	0.67	0.57	0.61	0.59	0.61	0.70	–
`bayer01`	8	257		**0.004**	**0.004**	0.006	0.007	0.009	0.011	0.014	0.051
`memchip`	256	68 122		**3.45**	3.67	4.86	8.29	16.10	–	–	–

III.2.3.c Node aware placement of master and worker processes

Here we use the node aware placement of master and worker processes introduced in Section III.2.2.a, to apply the BC and ABCD methods on the square matrices `bayer01`, `cage14`, and `memchip`. These experiments are based on the same settings as those performed in [45], and we recall now the useful details for the analysis of the results. We apply the ABCD-Solver in parallel using 128 MPI processes spread over 16 distributed nodes. Each process runs with 2 OpenMP threads. Matrix `bayer01` is partitioned in blocks of rows, and matrices Hamrle3 and `memchip` are partitioned in blocks of columns, using the partitioning giving the smallest number of iterations for convergence.

In this setting, each matrix is partitioned into 32 blocks which are given to separate master processes. The processes left are then assigned as workers to help with the factorisation and the solution of the projection systems. We recall that only masters are involved in the solution of the Schur complement system S, thus no impact on its solution time is expected from placing masters and workers. Note that here we use a fixed number of partitions such that the number of masters (32), is low compared to the number of workers ($128 - 32 = 96$). While 32 partitions stays of the same order as the number of partitions presented in Table III.7, we are again exaggerating the situation to force the importance of the master-worker parallelisation. We then use the three matrices `bayer01`, `cage14`, and `memchip` again because they show the main behaviours in terms of computation and communications.

Here we apply two methods to distribute the masters and workers: the Scatter method, which spreads the masters over the allocated nodes, and the Compact method, which places the masters close to each other. Table III.10 displays for the BC method the execution times for the MUMPS factorisation, the block-CG execution, the sum of projections and also gives the number of iterations. As for the ABCD method, Table III.10 gives the execution time for factorisation also, as well as the timing to compute the solution of the S system. Using the Scatter approach generally reduces the factorisation time (e.g. up to 75% for `memchip`). Factorisation only increases slightly for the matrix `memchip` in ABCD. While the masters are scattered, the master-workers groups are meant to be grouped in the same node. This way, the intra-communication is reduced together with the concurrent access in memory compared to the Compact approach. A benefit from the Scatter approach is also observed for the sum of projections in the BC method. This phase includes the solve step of MUMPS, behaving similarly to the factorisation w.r.t. the Scatter method, and a communication between masters to exchange the local projections. In the case of `bayer01`, the execution time for the sum of projections increases. In fact, `bayer01` has highly interconnected partitions of size only around 60 variables. As the granularity in the direct solver crumbles, the communication from the sum of projections predominates and there is no benefit from the Scatter method on this point. To sum up, when using the Scatter approach, the execution time for the BCG still decreases in this case, because less concurrent access to memory are performed by the masters in the dense kernels used when applying the Scatter technique. Overall, the execution time for the BC method, including factorisation and BCG, is reduced by 22% in average, and the execution time for the ABCD method, including factorisation and solution of the Schur complement system, is reduced in average by 17%.

Table III.10 Impact of the placement of masters and workers on the execution times of the ABCD Solver. All runs were performed with 32 partitions and 128 MPI processes with 2 threads on 16 nodes.

Matrix	Algo.	Block Cimmino				ABCD		
		Facto(s)	BCG(s)	it.	Proj. sum(s)	Facto(s)	Sol.(s)	Aug.
`bayer01`	Compact	0.89	0.73	81	0.40	1.41	0.07	1 083
	Scatter	0.07	**0.82**	81	0.41	**0.82**	0.09	
`cage14`	Compact	37.80	9.67	13	8.63	112.56	–	2 380 194
	Scatter	37.79	**8.82**	13	8.07	**104.87**	–	
`memchip`	Compact	1.20	90.70	242	62.60	1.14	40.70	34 048
	Scatter	0.44	**83.30**	242	61.30	**1.44**	40.41	

III.2.3.d Comparison with direct solvers

We now compare the parallel efficiency of the BC iterative method and pseudo-direct ABCD method (with the best settings for each of them) together with state-of-the-art direct solvers. We focus on the well-known parallel direct solvers MUMPS-5.3.1[10] [6] and QR-MUMPS-2.0[11] [29]. MUMPS is designed for the solution of unsymmetric square systems via an LU factorisation, and uses a hybrid MPI-OpenMP parallelism. QR-MUMPS is a multithreaded software based on the StarPU runtime engine[12], and is intended for rectangular matrices with the minimum-norm solution of underdetermined systems and the solution of least-squares problems. Both solvers are known to be very robust and support an efficient parallelism based on variants of the multifrontal method [6].

For our comparison, we extracted all the matrices from the SuiteSparse Matrix Collection with more than 1 000 rows. In the case of the square matrices, we then compare the iterative and pseudo-direct methods from the ABCD-Solver with the solver MUMPS. These runs are performed on the cluster Meggie with 64 MPI processes spread over 16 nodes. Each MPI process is attached to 4 OpenMP threads.

In the case of rectangular matrices, we compare the ABCD-Solver with QR-MUMPS in shared memory on the cluster Kraken[13] at CERFACS, France. Kraken is a cluster with Skylake Intel Xeon Gold 6140 processors at 2.3GHz. Each of its 185 compute nodes is a 2-socket system with 96 GB memory, where the 18 cores of each processor constitute a separate NUMA (non-uniform memory access) domain. Kraken uses the Intel OmniPath interconnect. This cluster was chosen for its larger number of cores and memory per node, which makes it better suited to shared-memory parallelism than Meggie. In our experiments, we use 18 threads for QR-MUMPS, i.e. a full NUMA domain on a node, and for the ABCD-Solver we use 9 MPI processes with 2 threads attached to each process, i.e. 18 active cores spread over the 2 sockets on a single node. We use this configuration to run QR-MUMPS in the best conditions available for a fair comparison with respect to the resulting efficiency. In that respect, the ABCD-Solver is run in a shared memory setting (which is not ideal for such a hybrid solver) but still takes advantage of the hybrid MPI-OpenMP parallelism.
In Figure III.11, we display the relative difference between the direct solver and our solver in terms of time and memory consumption. Positive values in the figure show an

[10] `http://mumps-solver.org/`
[11] `http://buttari.perso.enseeiht.fr/qr_mumps/`
[12] `https://starpu.gitlabpages.inria.fr/`
[13] https://cerfacs.fr/en/cerfacs-computer-resources/

advantage for the ABCD-Solver. After merging matrices from the same families and only keeping results whenever both compared solvers actually achieve the run, we are left with 187 classes of problems in the comparison with MUMPS, and 128 for the comparison with QR-MUMPS. In terms of execution times, the first thing we can observe is that the direct solvers usually performs better than ABCD or BC as most of the points are on the left side, associated to the negative part of the x-axis. Nevertheless, there are about 17% of cases where either ABCD or BC shows some advantage compared to either MUMPS or QR-MUMPS. The second thing to observe is that the ABCD-Solver has a lower memory consumption than MUMPS as 75% of points are on the upper and positive part of the y-axis. With respect to QR-MUMPS, the memory footprint gives advantage to the ABCD-Solver only in around half of the cases. The results from Figure III.11 are summarised in the table III.11 where we display the number of cases giving advantage to either solver based on execution time or memory.

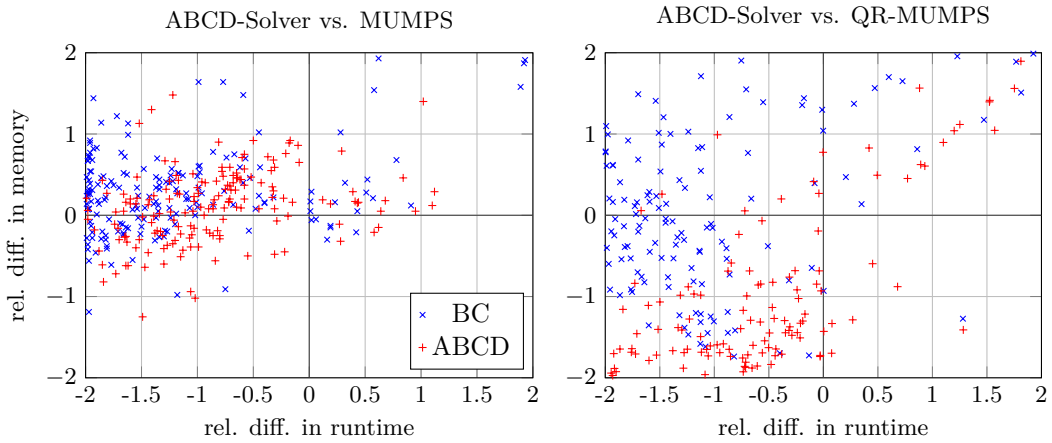


Figure III.11 Relative difference between the use of the ABCD-Solver and *(Left)* MUMPS or *(Right)* QR-MUMPS in terms of *(x-axis)* execution time and *(y-axis)* memory consumption for the block Cimmino iterative method applied to 187 and 128 classes of problems from the SuiteSparse Matrix Collection. Positive values show a smaller execution time (resp. smaller memory consumption) when using the ABCD-Solver.

In order to get a better understanding of major differences with respect to the behaviours of these solvers, we now focus on some specific examples. In [47], the authors present a detailed comparison between the ABCD-Solver and MUMPS for unsymmetric square matrices. In this thesis, the main contribution is for the solution of rectangular systems, and we now focus on the matrices `LargeRegFile`, `neos` and `shar_te2-b2` from Table III.7 to discuss the performance of the ABCD-Solver compared to QR-MUMPS. These matrices were chosen to show the typical behaviour where each method, QR-

Table III.11 Best solver in terms of execution time or memory for different classes of problems from the SuiteSparse Matrix Collection. We distinguish the comparison in distributed memory MUMPS vs ABCD-Solver (187 classes), and the comparison in shared-memory QR-MUMPS vs ABCD-Solver (128 classes).

Time		**Memory**	
MUMPS	**ABCD-Solver**	**MUMPS**	**ABCD-Solver**
155	32 (BC: 23, $ABCD$: 18)	51	136 (BC)
QR-MUMPS	ABCD-Solver	QR-MUMPS	ABCD-Solver
101	27 (BC: 7, $ABCD$: 20)	68	60(BC: 58, $ABCD$: 2)

MUMPS, ABCD, or BC, has the advantage depending on the particular structure and properties of the matrix. We start with the solution of the underdetermined matrix `neos`, which runs are given in Table III.12. The execution times are summarised for runs of the BC and ABCD methods as well as the QR-MUMPS solver. In the Table III.12, we distinguish

- Analysis: including the partitioning and augmentation of the matrix as well as the analysis of the projection systems by MUMPS inside the ABCD-Solver. For QR-MUMPS, this represents the analysis of the whole system.
- Facto.: the parallel factorisation of the projection systems by MUMPS inside the ABCD-Solver. The factorisation of the whole matrix for QR-MUMPS. The average memory per core is displayed for these steps.
- Solve: for BC, this corresponds to the execution time of the block-CG for BC, i.e. Algo. 5 in the case of the column partitioning. For QR-MUMPS, this is simply the solve phase.
- We also show for ABCD: the size and density of S, as well as the time to build it; the execution time for factorisation and solving using MUMPS and the corresponding requirement in terms of average memory per active core. In the case of the column partitioning approach, these steps correspond to the Algo. 8.

As expected, the preprocessing time increases between the BC and ABCD methods, which corresponds to the computation of the augmentation. This is reflected by a slightly longer analysis of the projection systems. Even with the augmentation, the analysis phase stays faster for the ABCD-Solver than for QR-MUMPs.

The same tendency is observed for the factorisation phase: only 1.3s and 2.0s for BC and ABCD, compared to 108.1s for QR-MUMPS. Additionally, we observe that the memory required for the augmented approach is increased by a factor 2.3 compared to memory needed by the iterative scheme. The direct solver requires more than 26 times

the amount of memory. This is expected as we split the problem into 64 smaller problems to factorise inside the ABCD-Solver, reducing the requirements in both memory and computation time.

The iterative method takes a high number of iterations to converge, thus requiring a long execution time. In the case of the augmented approach, the size of the augmentation is small, around 8% of the column dimension of the original matrix, but S is quite dense with around 3 100 entries per row. Due to this density, the required memory rises up to almost 2.0 GB, which is 23.5 times the memory required to factorise the projection systems. As for QR-MUMPS, once the price for the factorisation is paid, getting the solution is cheap with only 2.5s. Overall, the augmented approach is the fastest for this problem, and we can also notice that it requires 13% less memory than the direct solver. In terms of accuracy, QR-MUMPS and ABCD both reach backward errors of the order of machine precision, while BC only computes an approximation of the solution corresponding to the sopping criteria. In most applications, this accuracy would be enough, since errors are usually present in the data and/or models anyway.

Table III.12 Execution times for the different steps involved in BC, ABCD and QR-MUMPS on the problem `neos` partitioned into 64 partitions. The backward error ω on the normal equations for the computed solution is given.

	BC		**ABCD**		**QR-MUMPS**
Analysis	Preprocess	Anal. proj.	Preprocess	Anal. proj.	88.1
	5.3	1.1	5.7	1.2	
factorisation	1.3		2.0		108.1
Memory usage/core (MB)	37		84		2 247
solve	754.0 (920 it.)				2.5
Size of S			$4.27 \cdot 10^4$		
Entries (density) of S			$1.34 \cdot 10^8$		
Building S			63.4		
Fact. + solve with S			20.0		
Memory usage/core (MB)			1 970		
Total	761.8		92.3		198.7
ω	$9.88 \cdot 10^{-10}$		$1.12 \cdot 10^{-16}$		$3.00 \cdot 10^{-21}$

We now move to the case of the overdetermined matrix `LargeRegFile`, which is a difficult one for the augmented approach. Despite an augmentation size of around 8% the row dimension of the original system, the density of S is so high that the memory required to apply the ABCD method on this matrix does not fit our machine. In this case, we focus on the results of BC and QR-MUMPS, as shown in Table III.13.

Table III.13 Execution time for the different steps involved in block Cimmino, and `QR-MUMPS` on the problem `LargeRegFile` partitioned into 128 partitions. The backward error ω^{LS} on the normal equations for the computed solution is also given.

	BC	**QR-MUMPS**
Analysis	6.6	3.8
factorisation	23.2	1.2
Memory usage/core (MB)	152.7	80.9
solve	152.0 (28 it.)	0.6
Total	181.8	5.7
ω^{LS}	$3.20{\cdot}10^{-10}$	$5.50{\cdot}10^{-23}$

The iterative BC scheme exhibits a very fast convergence with only 28 iterations. However, each iteration costs 5.4s in average, 90% of the BCG iterations is spent on MPI communication during the distributed sum of projections. This is due to 2 very dense columns on the left of the matrix `LargeRegFile`, which interconnect the corresponding partition to all the other partitions. On the opposite, it is a particularly good matrix for QR-MUMPS which can actually benefit from pivoting strategies to overcome nicely the issues induced by these dense columns. After reordering using a simple SymAMD (Symmetric approximate minimum degree permutation), the normal equations end up in a perfect arrowhead form, see left Figure III.12. Due to this particular structure, the complexity of the direct solver is very small, both in terms of computations and memory. This explains why BC requires even more memory than QR-MUMPS. Back to the matrix `neos`, the structure of the matrix was not so easy to handle for QR-MUMPS, which explains its low performance on that matrix. Using the same SymAMD algorithm to reorder the normal equations of this matrix shows a structure very likely to fill-in inside QR-MUMPS (see right Figure III.12). This is due to particularly dense columns in this underdetermined matrix (and QR-MUMPS actually factorises its transpose).

We finally focus on the matrix shar_te2-b2, and show the results in Table III.14. For this matrix, the size of the augmentation with ABCD becomes large with around 46% of the row dimension of the original matrix. Computing the solution with the matrix S requires a long execution time, due to its high density of around 5 700 entries per row. Due to this high computational cost, the augmented approach is still 40 times slower than the direct solver. Furthermore, the memory required for ABCD is larger with 51 GB per core compared to almost 1 GB per core for QR-MUMPS. With this type of matrices, the iterative BC shows up as a good alternative. BC converges in 11 cheap iterations, and the memory cost for the factorisation of the projection system is very low

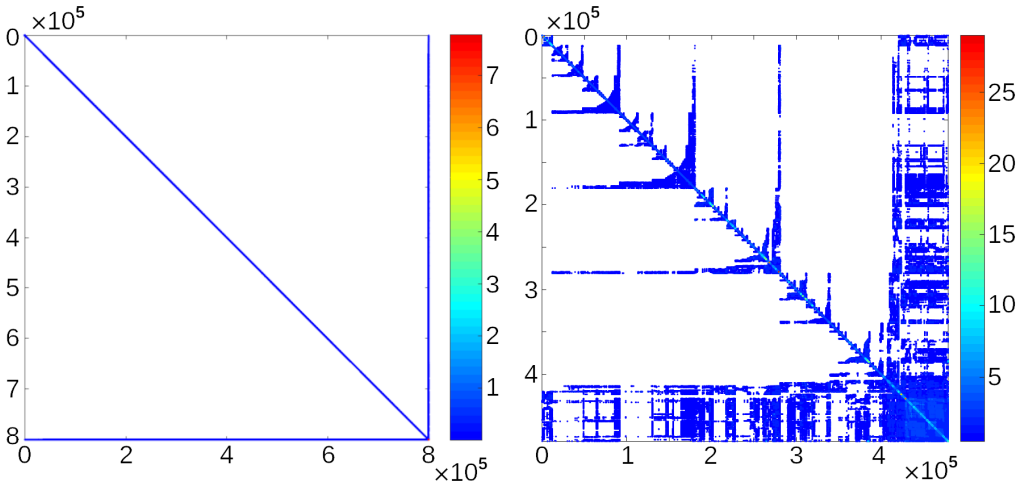


Figure III.12 Structure of the normal equations after reordering with a SymAMD (Symmetric Approximate Minimum Degree) algorithm for the matrices *(Left)* `LargeRegFile` and *(Right)* `neos`.

with only 27.8 MB per core. In terms of accuracy, we have the same conclusion as for the previous underdetermined problem `neos`. In particular, BC computes an approximation with a precision of the order 10^{-11} again.

Table III.14 Execution time for the different steps involved in BC, ABCD and QR-MUMPS on the problem shar_te2-b2 partitioned into 64 partitions. The backward error ω^{LS} for the computed solution is also given.

	BC		**ABCD**		**QR-MUMPS**
Analysis	Preprocess	Anal. proj.	Preprocess	Anal. proj.	88.3
	0.5	0.3	1.8	7.0	
factorisation	0.4		4.2		104.9
Memory usage/core (MB)	27.8		480.5		980
solve	2.4 (11 it.)				2.3
Size of S			$9.26 \cdot 10^4$		
Entries (density) of S			$5.27 \cdot 10^8$		
Building S			758		
Fact. + solve with S			1 552.7		
Memory usage/core (MB)			51 000		
Total	3.7		2 331.4		59.1
ω^{LS}	$4.36 \cdot 10^{-11}$		$1.60 \cdot 10^{-20}$		$3.67 \cdot 10^{-20}$

Concluding remarks

We demonstrated that our approach implemented in parallel shows generally a good memory consumption, even compared to a direct solver based on QR decomposition. However, in cases where the matrix S is too big or dense, the memory and computation required to solve it may become prohibitive. For these cases, we provide a block Cimmino accelerated with a stabilised block-CG algorithm for the solution of least squares problems. The latter method, whose convergence is problem dependent, keeps a very low memory requirement.

The ABCD approach makes a good option to solve least squares problems or underdetermined systems. The efficiency of this approach is of course dependent on the structure of the matrix, and e.g. dense rows or columns can destroy the behaviour of the solver due to an excessively big augmentation. ABCD compares well with the direct solver QR-MUMPS in a shared memory setting, giving comparable accuracy for rectangular systems. The iterative BC approach is then a good alternative with low memory consumption in cases where the structure of the matrix induces a too large or too dense S matrix, particularly when the convergence is fast.

In cases where the system is constructed from the discretization of a PDE problem, we can exploit the geometry of the system to obtain an efficient partitioning. In Chapter IV, we introduce a method based on a relaxed augmentation procedure. The resulting method, inspired from multigrid, is no longer a pseudo-direct method, but an iterative one with hopefully fast linear convergence.

Chapter IV

A multigrid-based approach for the Augmented Block Cimmino Distributed solver

"Do not forget the famous law of conservation of nastiness"

Ulrich Ruede

For large PDE problems, we investigate extensions of the *augmented block Cimmino method* (ABCD) from Section II.3, in which we relax the strict orthogonality between blocks in order to reduce the size of the augmentation blocks. The purpose is a better control of the requirements in memory and computations, mostly involved with the solution of the Schur complement. We exploit ideas from the multigrid framework, assuming that we have several levels of grids for the discretisation of the geometry. In Section IV.2, we augment the original matrix by enforcing the strict orthogonality between partitions on a coarse level. While in the exact ABCD approach, the size of the augmentation can be large for highly connected subdomains, this new approach gives a way to control explicitly the augmentation through the choice of the coarse grid level. This system can be solved with the block-CG acceleration of the *block Cimmino* method (BC) from Section II.2, with fast linear convergence for a range of systems coming from PDE problems. We demonstrate the efficiency of this method on heterogeneous Elliptic, Helmholtz and Convection-Diffusion 2D problems.

The main issue of this approach is the construction and solution of the reduced Schur complement S, which cannot be obtained directly through a single sum of projections

anymore. In the second part of this chapter, we propose 2 methods to handle this matrix. In Section IV.3, the Schur complement is built iteratively with the help of BC iterations. In Section IV.4, we use the algebraic expression of S and its inverse to apply the latter without construction.

IV.1 Controlling the Schur complement

The BC method, introduced in Section II.2, is efficient for a variety of problems, but its convergence stays hard to predict and is problem dependent. To address this issue, the ABCD approach was introduced in Section II.3, yielding a pseudo-direct alternative that relies on the solution of a smaller condensed matrix corresponding to a Schur complement. The factorisation and solution with this Schur matrix S, brings two issues

1. **S can be ill-conditioned:** the matrix S is a condensed representation of the ill-conditioning of the original system and the relations between partitions. By improving the scaling of the original matrix, following the method introduced in Section III.1.1, it is possible to improve partly the numerical properties of S.
2. **S can be large and/or dense:** as seen in the results of Section III.1.3, when the number of interactions between subdomains is high, the size of the augmentation grows accordingly. A way to explicitly control this size is required to be able to address with this method very large 3D PDE problems in particular, as it can make the solution with S impossible with a direct solver because of the large memory and computational requirements.

The latter point is not specific to a restricted class of problems as it can systematically arise when looking at highly interconnected matrices.

IV.1.1 Using coarser levels

As seen in Section II.3, ABCD is equivalent to a DDM where we decouple domains by duplicating variables at the interface. The size of the augmentation is then of the same order as the number of interface nodes between subdomains. Consider for instance a 3D Poisson problem discretized on a cube, with standard 7-point finite difference stencil. Let the cube be meshed with a number of $N = 2^{12} = 4.096$ points in each direction. The total number of nodes is then $n = N^3 \sim 68.7 \cdot 10^9$. Let the 3D cube be partitioned in $l_p = 16$ blocks in each direction, which gives $p = l_p^3 = 4096(= N)$ partitions in total, see Figure IV.1. The interconnection between 1 partition and the rest is approximately equal to the number of variables in its 6 faces: $M_i = 6 \times f$ with $f = (\frac{N}{l_p})^2$ the number of

variables in 1 face. The size of the augmentation block C will then be of the same order

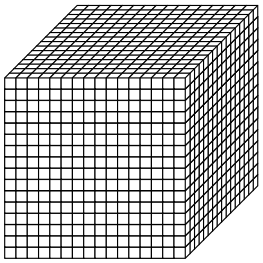


Figure IV.1 Partitioning of a cube with 16 division in every directions.

as the size of the faces

$$size(C) \sim \sum_{i=1}^{p} M_i = 6p(\frac{N}{l_p})^2 \approx \mathcal{O}(N^3). \tag{IV.1}$$

When facing 3D problems, one cannot afford to augment the system with such a huge size for the Schur matrix S, for which a direct solver would require too much memory, and we need to reduce and control better the augmentation size. With a reduction of the augmentation, information is no longer completely condensed on the interface, as there remains lingering interconnections. Algebraically, the augmented partitions are no longer mutually orthogonal in the resulting augmented system. We are then forced back to an iterative scheme, as the convergence in 1 iteration for the block Cimmino method will be lost. The goal is to find an augmentation procedure opening sufficiently the angles between the augmented partitions, to enable a fast convergence of BC, while controlling the size of the resulting augmentation. In the context of discretized PDE problems, a natural approach is to use a coarser discretization of the problem, in particular on the interface. This idea is directly inspired from multigrid methods [27]. As a matter of fact, using a hierarchy of grids naturally decreases the size of the considered problem when going to coarser levels. In principle, two successive levels of grids are linked by the prolongation operator P of size $n \times n_c$. The prolongation will be our main tool in this chapter.

Back to the 3D Poisson discretized on a cube, let's consider we have several levels of grid refinement with n_l variables on grid l, and $l = 0$ is the coarsest level. As seen in Section I.2.2, when using standard coarsening methods on regular grids the number of variables in 2 consecutive levels is linked by the approximate relation $n_l \approx \frac{1}{2^d} n_{l+1}$ where d is the dimension. Considering 2 consecutive grid levels, the number of variables on the 2D interfaces is divided by 4 on the coarser level. Simply put, starting from the finest level of

our 3D cube where the interface has roughly $1.6.10^9$ variables, only by considering the 3rd coarser grid, the size of the interface is reduced to $2.5.10^7$ variables, i.e. a 98.5% reduction.

Using such coarse levels would allow to choose an acceptable augmentation size. In the case of problems not coming from the discretization of PDE problems, algebraic multigrid tools could eventually be considered to allow the use of several grid levels thanks to aggregation methods applied directly on the entries of the sparse matrix [132].

IV.1.2 Challenging PDE problems

In this chapter, we are focusing on the solution of discretized PDE problems on which a hierarchy of refined meshes is naturally constructed. In order to study the efficiency of the methods we propose, we are focusing on 3 linear PDE problems, inspired from those introduced in [53].

IV.1.2.a Domains and discretization

We consider 2 dimensional PDE problems discretized on

- the square domain $\Omega_\square = (-1, 1) \times (-1, 1)$,
- the L-shaped domain $\Omega_\mathbf{L}$, defined as the union of the 3 smaller square domains $(-1, 0) \times (-1, 0)$, $(-1, 0) \times (0, 1)$, and $(0, 1) \times (0, 1)$.

The dimensions of the space are noted as usual x and y. An initial grid is defined on these domains, see Figure IV.2. Concerning $\Omega_\mathbf{L}$, this initial grid splits the domain in several subdomains and in particular 2 rectangular patches (see the red lines in Figure IV.2) on which the PDE problem will have special properties.

A hierarchy of L levels of nested triangular grids is then built using *initmesh* for the initial grid and the refinement method *refinemesh* from the Matlab PDE toolbox©. For illustration, the grids generated with 2 levels of refinement are displayed on Figure IV.2 for the 2 types of domains.

IV.1.2.b Test problems

The problems considered here are derived from the equations introduced in Section I.1.1. We are interested in the solution of

- the Helmholtz problem

$$\begin{aligned} -\nabla^2 u - k^2 u &= f \text{ in } \Omega_\square, \\ u &= 0 \text{ on } \partial\Omega_\square, \end{aligned} \tag{IV.2}$$

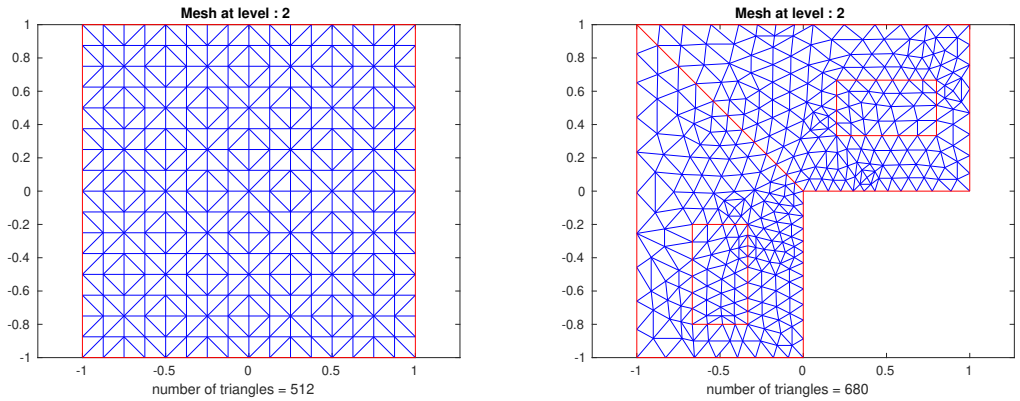


Figure IV.2 Structured grid after 2 levels of refinement for the *(Left)* square domain $\Omega_{\square}$ and *(Right)* L-shaped domain $\Omega_{\mathbf{L}}$.

with the wave number $k = 40$, and a sinusoidal forcing term $f = (5\pi^2 - 40^2)sin(2\pi x)sin(\pi y)$.

- the convection-diffusion equation

$$\begin{aligned} -\nu\nabla^2 u + \overrightarrow{w} \cdot \nabla u &= 0 \text{ in } \Omega_{\square}, \\ u = \begin{cases} 1 & \text{if } x = 1, \\ 0 & \text{else,} \end{cases} &\text{ on } \partial\Omega_{\square}, \end{aligned} \tag{IV.3}$$

  with the viscosity $\nu = 1/200$, i.e. dominating convection, and $\vec{w}$ a recirculating wind.

- the heterogeneous diffusion equation

$$div(c\nabla u) = f \text{ in } \Omega_{\mathbf{L}}, \tag{IV.4}$$

  with the forcing term $f = 200$. The heterogeneous diffusivity c and the boundary conditions are defined as shown in Figure IV.4. One of the difficulty of this problem is the presence of diffusivity jumps.

The shape of the solution for the 3 test problems is displayed in Figures IV.3 and IV.4 (Right). These problems are discretized on the previously generated grids using P1 finite elements, i.e. piece-wise linear continuous finite elements where the degrees of freedom are the values on the vertices. The solution to these linear systems is known to be challenging for multigrid methods in particular, due to either heterogeneous coefficients in the domain in (IV.4), or to high frequencies difficult to capture on a coarse grid in the case of Helmholtz problems [55], or due to strong non-ellipticity with a dominant convection effect in (IV.3).

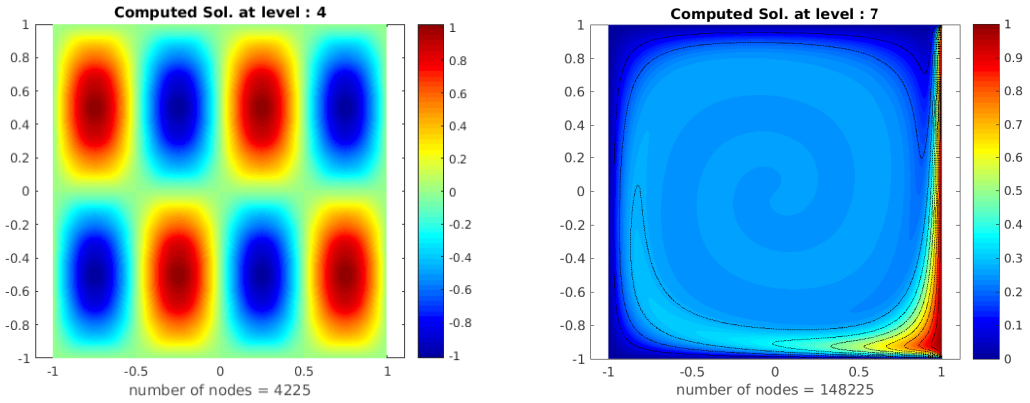


Figure IV.3 Shape of the solution for *(Left)* the Helmholtz problem (IV.2) *(Right)* the convection-diffusion problem (IV.3).

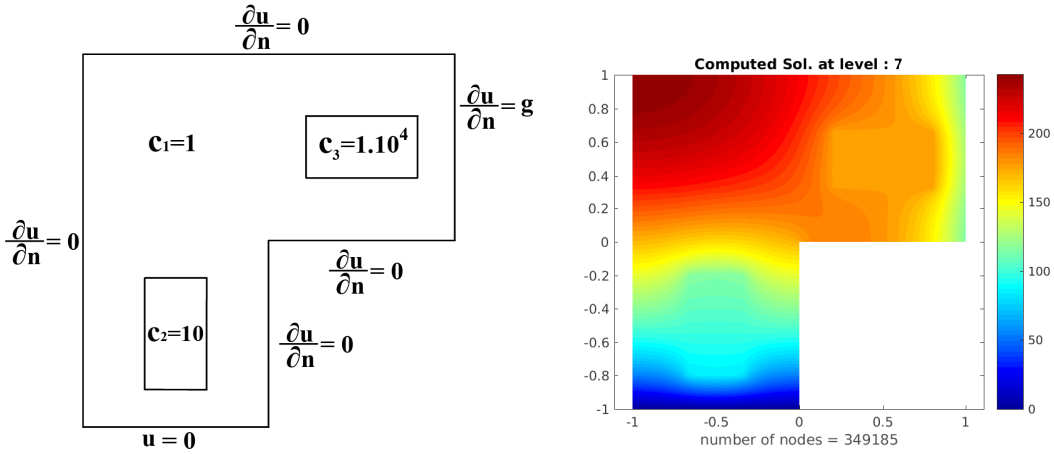


Figure IV.4 *(Left)* Diffusivity and boundary conditions for the Diffusion problem (IV.4) on the L-shaped domain $\Omega_{\mathbf{L}}$. The bottom is a Dirichlet, and the right a Neumann boundary condition, while we have an adiabatic condition on the sides. *(Right)* Shape of the solution.

IV.1.2.c Multigrid elements and partitioning

One way to deal with Dirichlet boundary conditions is to remove them from the linear system. In the following, we consider the solution of the linear system $Ax = b$ where A and b have been reduced to remove the Dirichlet boundary conditions on all grid levels. The complete discrete solution of the discretized PDE problem is then given by $u = P_{BC}x + u_D$, where u_D corresponds to the boundary conditions, and P_{BC} expands the computed solution x with zeros in order to integrate u_D. We also define between 2 consecutive levels of grids, l and $l+1$ (l being the coarsest), the prolongation operator

P_l^{l+1} as bilinear interpolation, and the restriction operator R_{l+1}^l. The restriction is usually taken as $R_{l+1}^l = P_{l+1}^{l^T}$. The linear operator on each level l is denoted $A_l \in \mathbb{R}^{m_l \times m_l}$, where $A_L = A \in \mathbb{R}^{m \times m}$, $L > 0$ is the finest grid level considered, and we usually define recursively A_l, $l \geq 0$ with the Galerkin operator, i.e. $A_l = R_{l+1}^l A_{l+1} P_l^{l+1}$.

Finally, as the methods introduced in this chapter are variants of the block Cimmino methods seen in Chapter II, we need to define a partitioning of the generated matrices. We use here the geometry of the problem. The partitioning is then defines by splitting the 2D domain into small squares with a given number of subdivision $ndiv$ in each direction, for the whole domain $\Omega_\square$, or with a similar parameter to subdivide each of the 3 smaller squares forming $\Omega_\mathbf{L}$. Figure IV.5 displays the partitioning of $\Omega_\square$ into 9 subdomains ($ndiv = 3$) and $\Omega_\mathbf{L}$ into 12 subdomains ($ndiv = 2$).

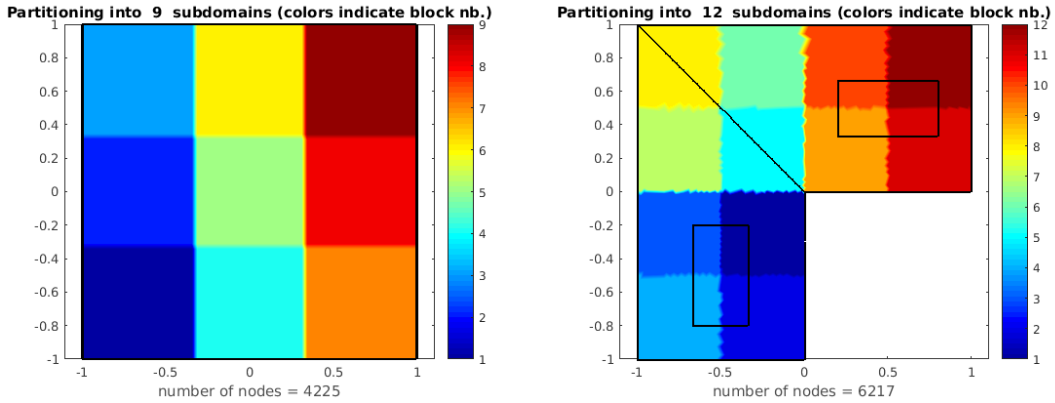


Figure IV.5 Partitioning of the domains based on the geometry: *(Left)* $\Omega_\square$ into 9 subdomains and *(Right)* $\Omega_\mathbf{L}$ into 12 subdomains.

During our experiments in the next sections, we use 2 sets of test cases

- $\mathcal{P}_{small}$: small test cases for the PDE problems above, for illustration purposes. The PDE problems are discretized on 3 levels of grids, respectively refined 2 times from the initial grid. The obtained matrix on the finest grid level is partitioned into 12 blocks of rows for the diffusion, 9 blocks for the others, following the geometry.
- $\mathcal{P}_{large}$: large test cases for the PDE problems. We generate a hierarchy of 7 grid levels corresponding to 6 refinements applied on the initial grid. The obtained matrix is partitioned into 16 blocks of rows for the convection-diffusion and Helmholtz problems, and 12 blocks of rows for the diffusion problem.

In all these test cases, we have at hand the prolongation operators from aggressive coarsening, i.e. between the coarse grid and the finest grid levels. For each problem, the size of the operators is given in Table IV.1, for every available levels of grid.

Set	Problem	Grid level					
		0	1	2	3	4	5
$\mathcal{P}_\mathcal{S}$	diffusion (IV.4)	97	364	1 408	–	–	–
	convection-diffusion (IV.3) Helmholtz (IV.2)	49	225	961	–	–	–
$\mathcal{P}_\mathcal{L}$	diffusion (IV.4)	97	364	1 408	5 536	21 952	87 424
	convection-diffusion (IV.3) Helmholtz (IV.2)	49	225	961	3 969	16 129	65 025

Table IV.1 Size of the matrix depending on the grid level for the different test sets and problems.

As we have seen above, one of the limiting factor of the ABCD approach concerns the Schur complement matrix S. Even using an efficient direct solver, S often remains either too dense, too large or both. As the sparse linear problems become more and more complicated, this will prove to be a limit in the end. In the following, we use the previously introduced hierarchy of grids in a relaxed variant of the ABCD method to control the size of the augmentation. We demonstrate this control on the above PDE problems.

IV.2 Enforcing a relaxed orthogonality

In this section, we introduce a new augmentation procedure to solve the linear system on the finest grid $Ax = b$ with $A \in \mathbb{R}^{m\times m}$, using ideas inspired from multilevel methods. We focus on 2-level methods, i.e. based on the use of a single coarser level chosen among the hierarchy of grids. The coarse level is integrated in the augmentation through the intergrid transfer operators defined from an aggressive coarsening scheme. Considering the finest level is the level $L > 0$, where $A_L = A$, and we use only the coarsest grid at level 0, where $A_0 : m_0 \times m_0$, then the prolongation and restriction operators are defined as the product of the intermediary operators

$$\begin{aligned} P &= P_{L-1}^{L} \dots P_0^1 : \ m \times m_c \\ R &= R_1^0 \dots R_L^{L-1} : \ m_c \times m \end{aligned} \tag{IV.5}$$

IV.2.1 General discussion on ways to construct a relaxed augmentation

A naive approach to relax the augmentation is to look for a general form of the augmented matrix $\begin{pmatrix} \overline{A} \\ W \end{pmatrix} = \begin{pmatrix} A & C \\ B & S \end{pmatrix}$. The goal is to construct an augmented system on which the convergence of the BC method is accelerated. What would be the ideal properties of such system ?

First, as we have seen, the convergence of BC is closely related to the angles between subspaces spanned by the partitions, and in a broader sense our target is to *improve the block diagonal dominance of* $\overline{A}\,\overline{A}^T$.
Then, as we need additional closure equations to maintain the solution of the original system, the second targeted property is: *the normal equations of the additional constraints block* WW^T *has a good spectrum*, to ensure an easy solution technique when computing the projections linked to this additional block.
Finally, the angles between the super partitions $\overline{A}$ and W should not slow the convergence so: $\overline{A}$ *and* W *should be close to orthogonal.*

Here we focus on a simple case with the matrix $A \in \mathbb{R}^{m\times m}$, and where we have the prolongation operator $P \in \mathbb{R}^{m\times m_0}$. The restriction operator is defined as $R = P^T \in \mathbb{R}^{m_0\times m}$, and we use the Galerkin operator on the coarse grid $A_0 = P^T A P \in \mathbb{R}^{m_0\times m_0}$, i.e. we satisfy the variational properties. To build the augmentation blocks, we focus on *linear transformations* involving P, A and A_c. The consistency of the system must be satisfied, which can be fulfilled by enforcing the additional variables to 0 and forcing the additional right hand side to be dependent on the original b. Based on dimensional arguments, the general form of the augmented system could be set as

$$\begin{bmatrix} A & M_1 P M_2 \\ N_1 P^T N_2 A & -G \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} = \begin{bmatrix} b \\ N_1 P^T N_2 b \end{bmatrix} \tag{IV.6}$$

where $M_1, N_2 \in \mathbb{R}^{m\times m}$, $M_2, N_1 \in \mathbb{R}^{m_c\times m_c}$, and $G \in \mathbb{R}^{m_c\times m_c}$. There is a number of possibilities on the choice of the matrices M_i and N_i satisfying all or part of the previously introduced properties.

We would like to mention at this point that it is also possible to avoid the use of the closure equations (viz $W = \begin{bmatrix} N_1 P^T N_2 A & -G \end{bmatrix}$ in (IV.6)). To do so, we have the possibility to choose $M_1 = A\widetilde{M}_1$, where $\widetilde{M}_1 : m \times m$, and the solution is then obtained as $x^* = x + \widetilde{M}_1 P M_2 y$ where y introduces extra degrees of freedom in the system. In this case, we only have to choose $\widetilde{M}_1$ and M_2 such that the normal equations of $\overline{A}$ are better

suited for fast linear convergence in BC.

In the same line of ideas, it is also possible to avoid considering the extra variables and to build only an overdetermined system $\begin{bmatrix} A \\ B \end{bmatrix} x = \begin{bmatrix} b \\ Bx \end{bmatrix}$ that has the same solution x. As long as we keep the system consistent as before, and the partitions used stay full rank, the block Cimmino algorithm will converge to the right solution. Obviously, the key point is to know Bx by default, and this is possible only with some particular choices for B. This kind of augmentation is closely related to the notion of overlapping partitioning we have introduced in Section III.1.2.e. Instead of replicating full rows into separate partitions, the idea here would be to take linear combinations of the rows in A to build a new block, which can then be used as an separate partition.
As above, the 2 new possible underdetermined and overdetermined alternatives are

$$\begin{bmatrix} A & A\widetilde{M}_1 P M_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = b, \text{ and } \begin{bmatrix} A \\ N_1 P^T N_2 A \end{bmatrix} x = \begin{bmatrix} b \\ N_1 P^T N_2 b \end{bmatrix}.$$

We now introduce three specific augmentations that we explored, based on the previous ideas. From (IV.6), one possibility is to augment the partitions with the prolongation operator, and set the additional constraints such that their normal equations are reduced to the identity. We use the augmented system

$$\begin{bmatrix} A & P \\ GN^{-1}P^T A & -G \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} = \begin{bmatrix} b \\ GN^{-1}P^T b \end{bmatrix},$$

where $M = (P^T A A^T P)(P^T P)^{-1}$, $S = (P^T P)^{-1} N$, and $G = chol((S + I_q)S^{-1})$ with *chol* giving the lower triangular factor from a Cholesky decomposition of the corresponding matrix. Using this augmentation, $\overline{A}$ and W are not orthogonal but the normal equations $\overline{A}\,\overline{A}^T$ are directly improved for small systems as we will see in the experiments below. Then we introduce the underdetermined and overdetermined augmented systems

$$\begin{bmatrix} A & A P A_c^{-1} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = b, \text{ and } \begin{bmatrix} A \\ P^T A \end{bmatrix} x = \begin{bmatrix} b \\ P^T b \end{bmatrix}. \tag{IV.7}$$

Concerning the overdetermined augmentation above, we show in Appendix C.1 that it corresponds to a 2-level multigrid cycle where the coarse grid correction is applied in an additive way instead of the classical product introduced in Section I.2.2. As such, it also corresponds to a 2-level additive Schwartz method. We now consider the convection-diffusion problem (IV.3) with 3 and 5 refinement levels. We use the corresponding

finest level of grid and the coarser grid is 2 levels below, i.e. we have for the problem refined 3 times $A \in \mathbb{R}^{961\times961}$ and $A_c \in \mathbb{R}^{49\times49}$, and for the problem refined 5 times $A \in \mathbb{R}^{16\,129\times16\,129}$ and $A_c \in \mathbb{R}^{961\times961}$.

In the first sub-figure of Figure IV.6, we show in blue the convergence of the BC method applied on the original system and in red the convergence of the BC method applied on the underdetermined augmentation in (IV.7). The convergence profile for the augmented system is characterised by plateaus, and converges with 2.5 times more iterations than in the non-augmented case. An improvement of the prolongator P itself was then considered, using Chebyshev filtering, in order to target explicitly the small eigenvalues on the coarse level. After filtering the prolongation operator using this method, we apply the BC method on the original system (blue curves) and using the 3 previously introduced augmentations (pink curves), see Figure IV.6. In the case of the smaller system, the Chebyshev filtering applied on the prolongation operator enabled a linear convergence of the BC method for the 3 augmentation techniques. The method with fastest convergence is then the overdetermined variant which converges in only 100 iterations compared to almost 1 000 for the original system. However, none of these approaches were found to be robust with respect to the size of the system.

All-in-all, as these formulations are not robust even with the algebraic improvement of the prolongation operator, we were then conducted to consider other augmentation procedures. In particular, we study a variant of the ABCD method taking into account the information coming from a coarse grid level.

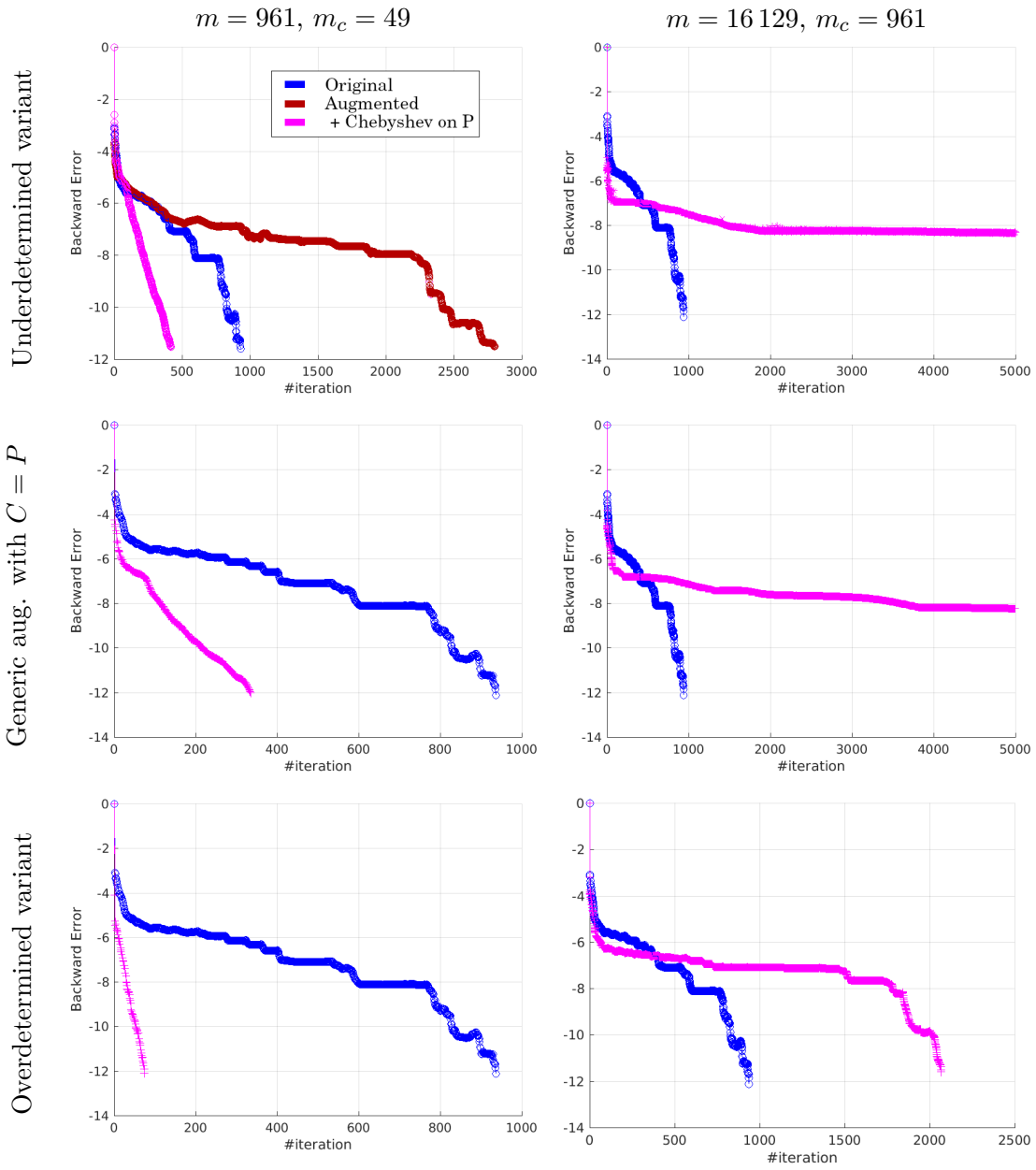


Figure IV.6 Convergence of the BC method applied on the original system *(blue)*, and on the augmented systems for the convection-diffusion problem with grids from 3 and 5 refinements. The augmentations considered are the underdetermined, the generic augmentation with $C = P$, and the overdetermined variants using P the prolongation between the finest grid (size m) and the next coarser grid (size m_c). The prolongator used in the augmentation is *(red)* the original, or *(pink)* the version improved with a Chebyshev filtering.

IV.2.2 A relaxed ABCD

We are now looking for the solution of the square linear system

$$Ax = b, \tag{IV.8}$$

where $A \in \mathbb{R}^{m \times m}$ is partitioned in blocks of rows $A_i \in \mathbb{R}^{m_i \times m}$ as in (II.5). Considering an augmentation scheme $\mathcal{F}$ from Section III.1.3, the ABCD method enforces the strict mutual orthogonality between augmented partition $\overline{A}_i$ through the block $\mathcal{F}(A)$. Using a rectangular thinner matrix $V \in \mathbb{R}^{m \times m_V}$, $m_V \ll m$, we propose a new approach where the augmentation block is built as $C = \mathcal{F}(AV) \in \mathbb{R}^{m \times q}$. AV and $\mathcal{F}(AV)$ are partitioned in the same way as A, we note their partitions respectively $(AV)_i = A_i V$ and $\mathcal{F}(AV)_i$, $i = 1, \ldots, p$. The result of this process is the augmented matrix

$$\overline{A} = \begin{bmatrix} A & \mathcal{F}(AV) \end{bmatrix}. \tag{IV.9}$$

For the choice of the matrix V

1. the size of the augmentation should be small i.e. $q \ll n$,
2. the normal equations of the matrix in (IV.9) are approximately block diagonally dominant, i.e. from a domain decomposition point of view, the subdomains defined by the partitions are mostly decoupled.

In fact, the strict orthogonality between partitions is enforced only for the matrix

$$\begin{bmatrix} AV & \mathcal{F}(AV) \end{bmatrix},$$

As a consequence, in the new system (IV.9), we have purely decoupled subdomains within the subrange $\mathcal{R}(AV)$, viz

$$\begin{aligned} \forall\, i, j \in \{1, \ldots, p\}, \qquad & (AV)_i (AV)_j^T + \mathcal{F}(AV)_i \mathcal{F}(AV)_j^T = 0 \\ \iff & \mathcal{F}(AV)_i \mathcal{F}(AV)_j^T = -A_i V V^T A_j^T \end{aligned} \tag{IV.10}$$

Typically, in the case of a classical multigrid cycle, smooth error components are problematic for the smoother applied of the finer levels of grids. Through the restriction and prolongation operators, multigrid methods resolve these smooth components onto the coarse grid. In the approach we propose, a natural choice for the matrix V would be, as in the previous general approach, to take the prolongation operator P between the finest grid and a chosen level of coarse grid. The expectation is that with such a choice for $V = P$, the information contained in AP (which actually corresponds to

the representation of the PDE problem on the coarse grid developed in the fine grid finite element basis) will be enough to capture the low frequency interactions between partitions and $\mathcal{F}(AV)$ will incorporate appropriately decoupled partitions.

IV.2.2.a Size of the augmentation

The size of AP on which the augmentation is built can be controlled through the choice of a more or less coarse level of grid. At the extreme, using the augmentation procedure directly on the finest level of grid, i.e. with $P = I_m$, simply corresponds to the ABCD method with convergence in 1 iteration.

To illustrate the new augmentation procedure, we introduce again the block tridiagonal matrix partitioned in 3 blocks of rows

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & & & \\ & A_{1,2} & A_{2,2} & A_{2,3} & \\ & & & A_{2,3} & A_{3,3} \end{pmatrix}.$$

We have a similar, possibly denser, structure for the matrix AP

$$AP = \begin{pmatrix} (AP)_{1,1} & (AP)_{1,2} & & & \\ & (AP)_{1,2} & (AP)_{2,2} & (AP)_{2,3} & \\ & & & (AP)_{2,3} & (AP)_{3,3} \end{pmatrix},$$

where $(AP)_{i,j}$ is the interconnection block between A_i and A_j. An augmentation technique, e.g. $\mathcal{F}^{FR}$ from Section III.1.3, is applied on AP to obtain

$$\overline{A} = \begin{bmatrix} A & \mathcal{F}^{FR}(AP) \end{bmatrix} = \left(\begin{array}{ccccc|cc} A_{1,1} & A_{1,2} & & & & (AP)_{1,2} & \\ & A_{1,2} & A_{2,2} & A_{2,3} & & -(AP)_{1,2} & (AP)_{2,3} \\ & & & A_{2,3} & A_{3,3} & & -(AP)_{2,3} \end{array} \right).$$

Note that, when generated from the geometry of the system, as in Section IV.1.2, the prolongation P stays relatively sparse, which gives good chances for AP to stay sparse as well. For an illustration of this sparsity, let's consider the small heterogeneous diffusion problem with 12 partitions in $\mathcal{P}_{small}$. Figure IV.7 displays the matrix A obtained on the finest level, the prolongation operator P between the 2 levels of grids as well as the product AP and the obtained augmentation $C = \mathcal{F}^{FR}(AP)$.

Remark: As a reminder from Section III.1.1, we build the augmentation for ABCD based on a well-scaled scaled matrix $\widetilde{A} = D_r A D_c$, with D_r and D_c diagonal scaling

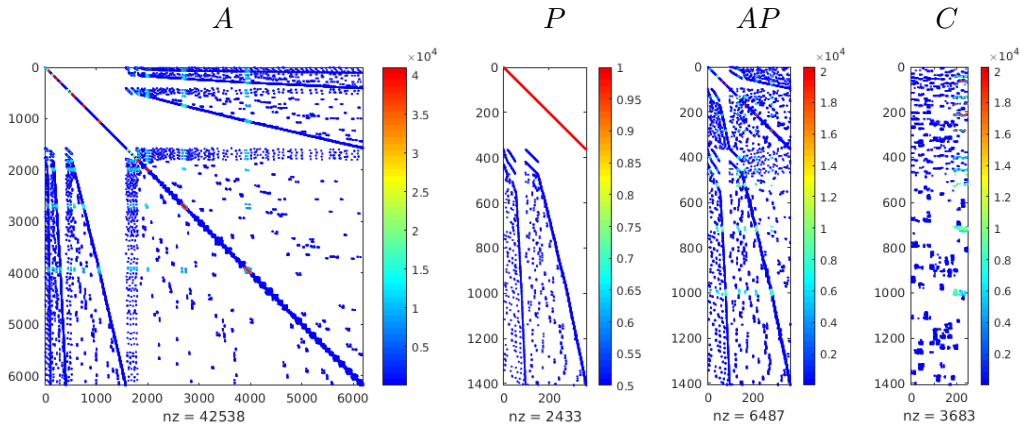


Figure IV.7 Matrix $A \in \mathbb{R}^{1408\times1408}$, prolongation $P \in \mathbb{R}^{1408\times364}$, the product AP and the computed augmentation $C = \mathcal{F}^{A_{ij}}(AP) \in \mathbb{R}^{1408\times248}$ obtained for the heterogeneous diffusion problem (IV.4) with 2 grid levels.

matrices. We have to be careful that the scaling is coherent when using the relaxed augmentation on AP. Using $\widetilde{A}$ as instead of A, we compensate the column scaling D_c on the rows of P for the augmentation. Also, we can add a diagonal scaling matrix D_c^c on the columns of P, i.e. a scaling on the coarse space, to obtain a scaled prolongator $\widetilde{P} = D_c^{-1} P D_c^c$. In this way, we obtain a scaled product $\widetilde{A}\,\widetilde{P} = D_r A P D_c^c$ which does not mix different magnitudes of scaling factors. We add the notion of scaled restriction $\widetilde{R} = D_r^c P^T D_r^{-1}$ which compensates the row scaling of A and also adds a scaling on the coarse space. In practice, it is enough to set $D_r^c = D_c^c$ and to compute these as scaling factors for the symmetric coarse grid operator $P^T A A^T P$ which is the Galerkin operator for the normal equations. We can show that these scaled operators $\widetilde{A}$, $\widetilde{P}$ and $\widetilde{R}$ can be used in a coherent 2-grids cycle applied on the normal equations AA^T, as introduced in Section I.2.2.

We now consider the set of large problems $\mathcal{P}_{large}$, i.e. the 3 PDE problems (IV.4), (IV.3), and (IV.2) from Section IV.1.2 with refined grids, and we have the prolongation operators defined from aggressive coarsening (IV.5) to transfer information from any grid level to the finest. In Table IV.2, we display the size of the prolongation operator, corresponding to the size of the linear operator on this level, as well as the size of the augmentation obtained with $\mathcal{F}^{FR}(AP)$ for the 3 problems. These sizes are shown for all choices of coarse grid level $l = 0, \ldots, 5$. Let's remind that the finest level amounts to applying the classical ABCD. We observe here that when going down to a coarser level, the size of P is divided by approximately 4 while the size of the augmentation is

divided by a factor close to 2. This reduction is expected as the problem is in 2D, thus the interfaces are in 1D, and we have seen that the number of unknown is divided by 2 in each direction using standard coarsening, see (I.9). This also confirms the previous interpretation that the augmentation is applied on the interfaces between subdomains.

Table IV.2 Size of the prolongation and the augmentation depending on the grid level chosen for the 3 PDE problems in test set $\mathcal{P}_{Large}$.

Coarse grid levels	**Diffusion**		**Helmholtz**		**Convection-Diffusion**	
	P	$\mathcal{F}^{FR}(AP)$	P	$\mathcal{F}^{FR}(AP)$	P	$\mathcal{F}^{FR}(AP)$
0 (ABCD)	87 424	9 329	65 025	4 112	65 025	4 096
1	21 952	4 766	16 129	2 072	16 129	2 064
2	5 536	2 477	3 969	1 048	3 969	1 044
3	1 408	1 339	961	536	961	536
4	364	764	225	280	225	280
5	97	471	49	152	49	152

IV.2.2.b Widening the angles between partitions

The expectation is that, with such a choice for $P = V$, the information contained in AP (which actually corresponds to the representation of the PDE problem on the coarse grid developed in the fine grid finite element basis) will be enough to capture the low frequency interactions between partitions, and $\mathcal{F}(AV)$ will appropriately decouple the partitions.

As we have seen in (IV.10), through the relaxed augmentation, partitions are made orthogonal on a subrange of the linear operator which corresponds to the low frequencies. We are decoupling the subdomains on a coarser space, and we expect a large improvement of the non-zero spectrum of H^{row}, the sum of projections on the partitions in $\overline{A}$. Considering as before the small diffusion problem in $\mathcal{P}_{small}$. Figure IV.8 shows the non-zero spectrum of the BC iteration matrix H^{row}, before and after augmentation using grid level 1 to define P. As expected, there is a very large improvement of the conditioning of H^{row} from $3 \cdot 10^{-6}$ to $2 \cdot 10^{-1}$. The resulting spectrum is well clustered around 1 which should imply a fast linear convergence in the block-CG acceleration. As a reminder, when ABCD is used with strict orthogonality between partitions, H^{row} becomes the identity.

IV.2.3 The relaxed augmented block Cimmino method

Once the relaxed augmentation has been applied, we need to add additional constraints to close the system in order to keep the same solution as the original problem. The

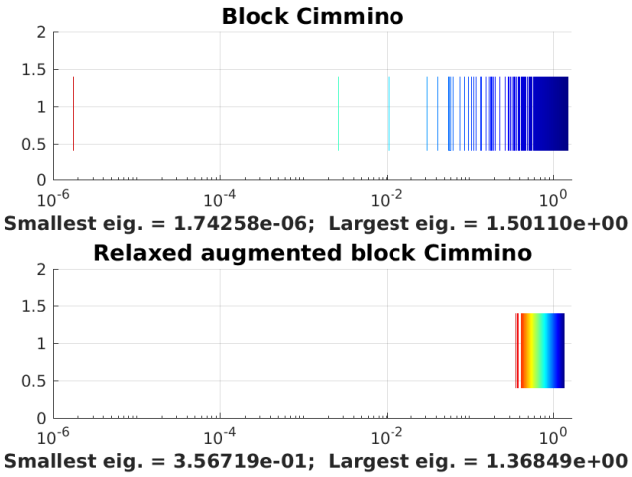


Figure IV.8 Non-zero spectrum of the iteration matrix H^{row} for the BC iterations applied on *(Above)* A and *(Below)* $\overline{A}$, obtained for the heterogeneous diffusion problem (IV.4) in $\mathcal{P}_{small}$ with 2 grid levels.

additional variables are still set to 0 by adding the block $Y = \begin{bmatrix} 0_{q\times m} & I_q \end{bmatrix}$ as in ABCD, and we need to make the additional constraints orthogonal to $\overline{A}$ so as to be able to get the final solution in 2 steps. As before, we project Y^T in the nullspace of $\overline{A}$, see (II.24), to obtain the new block

$$W = \begin{bmatrix} B & S \end{bmatrix} = Y(I - \overline{P}), \tag{IV.11}$$

with $\overline{P} = \mathcal{P}_{\mathcal{R}(\overline{A}^T)}$. The corresponding right hand side f, see (II.46), is

$$f = W \begin{bmatrix} x^* \\ 0 \end{bmatrix} = -Y\overline{A}^{+}b.$$

In this way, we construct the following enlarged system of equations

$$\begin{bmatrix} \overline{A} \\ W \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} A & \mathcal{F}(AP) \\ B & S \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ f \end{bmatrix}. \tag{IV.12}$$

The proof from Section II.3.2.a concerning the equivalence between the solution of (IV.8) and the solution of the augmented system using the ABCD method only relies on A being full rank, not on the orthogonality. Thus, this proof is still valid for the equivalence of the solution of the original system (IV.8) and (IV.12). Furthermore, from

(II.49) in Section II.3.2.a, the solution of (IV.12) is expressed as

$$\begin{bmatrix} x^* \\ 0 \end{bmatrix} = \overline{A}^+ b + W^+ f = \overline{A}^+ b - (I_{\overline{n}} - \overline{P}) Y^T S^{-1} Y \overline{A}^+ b. \tag{IV.13}$$

However, as the mutual strict orthogonality between partitions is no longer true, the projection onto $\mathcal{R}(\overline{A}^T)$ cannot be expressed as a sum of orthogonal projections anymore, i.e.

$$\overline{P} = \mathcal{P}_{\mathcal{R}(\overline{A}^T)} \neq \sum_{i=1}^{p} \mathcal{P}_{\mathcal{R}(\overline{A}_i^T)}.$$

Then, in the expression of the solution in (IV.13), one problem is that the pseudo inverse $\overline{A}^+$ is no longer decomposed as a sum. With the equality (IV.11), the Schur complement S is still given by

$$S = I_q - C^T (\overline{A}\,\overline{A}^T)^{-1} C,$$

where $\overline{A}\,\overline{A}^T$ is not block diagonal and thus $S \neq I_q - \sum_{i=1}^{p} C_i^T (\overline{A}_i \overline{A}_i^T)^{-1} C_i$.

Therefore, we can no longer compute the block W, nor the Schur S through a single sum of projections as in ABCD, due to the lost orthogonality between partitions. As the projection $\overline{P} = \mathcal{P}_{\mathcal{R}(\overline{A}^T)}$ involves the inverse of the normal equations $\overline{A}\,\overline{A}^T$ which is not block diagonal, we will have to approximate W and S using an iterative scheme. In the following sections, we propose two approaches for this iterative construction

1. in Section IV.3, we approximate the block W, and thus S, by an application of the BC iterations on $\overline{A}$ applied on Y^T. We then integrate the approximated block W as an additional partition to solve (IV.12) with a global BC scheme on the total enlarged system in (IV.12).
2. in Section IV.4, we seek a good preconditioner for S to apply directly its inverse using a classical iterative scheme, e.g. *preconditioned conjugate gradient* (PCG) iterations.

In the rest of this section, we concentrate on the convergence of BC on $\overline{A}$. Considering that W is computed exactly, the solution of (IV.12) is split as in (IV.13), and the block Cimmino iterations are only applied on $\overline{A}$, independently from W. We expect a fast linear convergence since the relaxed orthogonality is true on a subrange of A corresponding to the chosen coarser level, and hopefully this is enough to open the principal angles between subspaces spanned by the partitions.

We consider again the set of large problems $\mathcal{P}_{\mathcal{L}}$ from Section IV.1.2. Figure IV.9 displays the convergence profile in terms of backward error for the BC method on the

original system (IV.8) and on $\overline{A}$ with the same partitions, and considering a relaxed augmentation made with the grid level 2 (level 5 is the finest) so as to minimise the extra variables. The stopping criteria of the iterations is a backward error of 10^{-10}. Without augmentation, we observe a convergence characterised by long plateaus. On the opposite, when the matrix is augmented using the grid level 2, the convergence appears to be linear and fast. This means that the angles between partitions are quickly opened even with information coming from a very coarse grid discretization. This also implies that we can construct $\mathcal{P}_{\mathcal{R}(\overline{A}^T)}$ applied to any vector by means of a fast converging iterative method based on the sum of projectors onto the range of the partitions in $\overline{A}$. In the case of the diffusion problem, the number of iterations needed is 126 for an augmentation of size 1339, i.e. 2% of the original system size. We also observe that without these extra columns, the BC methods needs more than 2 000 iterations to converge which means small angles between partitions.

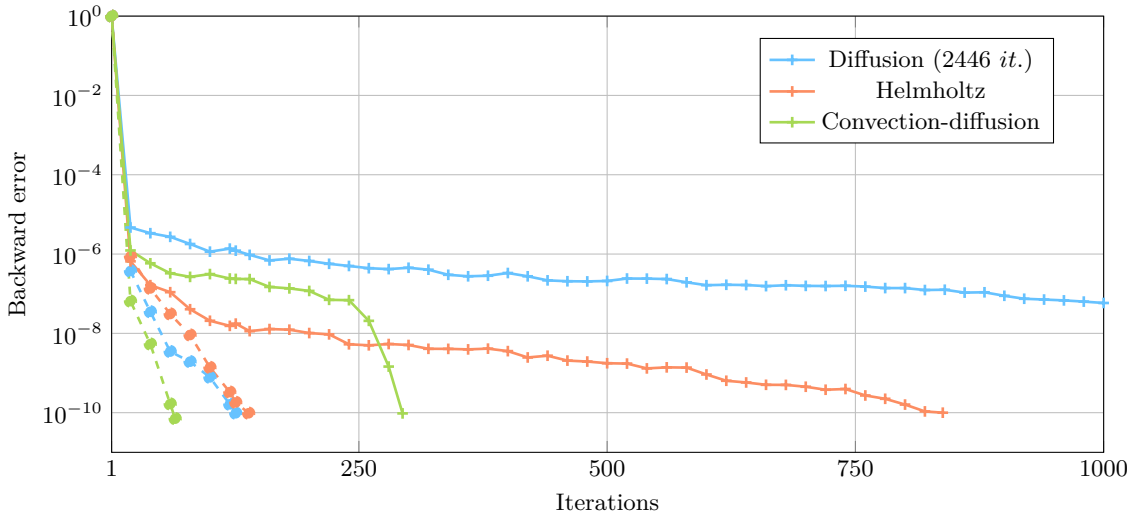


Figure IV.9 Convergence of the BC method applied *(plain line, plus marks)* on A (IV.8) and *(dashed line, round marks)* on $\overline{A}$ augmented using the grid level 2 for the 3 PDE problems in $\mathcal{P}_{Large}$

In Figure IV.10, we display for the 3 small problems in the test set $\mathcal{P}_{small}$ a representation of the solution obtained with the relaxed augmentation method on the domain split in 2 parts, respectively $\overline{A}^{+}b$ corresponding to the augmented partitions, and $W^{+}f$ for the closure equations. Each case is very different. For the diffusion problem, most of the solution is contained in the part coming from the closure equations, and the part of the solution coming from the augmented partitions does not act on the interfaces. As for

the Helmholtz problem, it is the contrary. Most of the global solution is contained in $\overline{A}^{+}b$, while $W^{+}f$ seems to take care of underlying waves together with the crossroads between partitions. Finally, $\overline{A}^{+}b$ for the convection-diffusion only acts on the boundary where the wind enters the domain, and $W^{+}f$ then resolves the spiral in the rest of the domain. The error of the global solution is then represented in Figure IV.11, and we notice that for the convection-diffusion and Helmholtz problems, the largest part of the error (of the order 10^{-8}) is centered around the interfaces. We are still missing the general interpretation from these representations.

Of course, we expect that by going down into coarser levels for large scale problems, the amount of available information is lower and there may be some trade-off between the size of the augmentation and the convergence speed. The idea is to fix a desirable grid size and a necessary amount of information for convergence. In the first experiments (Figure IV.9), we have used the grid level 2 to check the robustness of the approach, and we may expect even faster convergence with a finer grid level. Figure IV.12 displays the resulting augmentation size as well as the number of iterations for convergence of BC applied on $\overline{A}$ when varying the choice of the coarse grid level from the finest to coarsest available. The 3 test problems are coming from the set $\mathcal{P}_{large}$ and the iterations are monitored with the same stopping criterion. While the size of the augmentation is divided by 2 from one level to the next coarsened one, the number of iterations is also increased by a factor of approximately 2. Through the choice of a more or less coarse grid for the augmentation, we introduce a whole class of methods with linear convergence, standing in the middle between at one extreme the ABCD method, with a pure orthogonality and convergence in 1 iteration, and at the other extreme the BC method, with no augmentation and a convergence often in plateaus. We call this new approach *Coarse-ABCD* (C-ABCD).

As our approach is very similar to a 2-level domain decomposition method, one question is the influence of the number of subdomains on its convergence. In the domain decomposition literature, we call an iterative scheme optimal if it is independent of the number of subdomains. For the 3 problems in $\mathcal{P}_{Large}$, we now vary the number of divisions in each direction $ndiv$ for the partitioning of the domains. The resulting number of partitions is of $ndiv^2$ for the square domain, and $3ndiv^2$ for the L-shape domain. Figure IV.13 shows the evolution of the number of iterations for the convergence of the block Cimmino iterations on $\overline{A}$, and the augmentation size for $ndiv = 2, \ldots, 10$. It is interesting to see that when we augment the number of subdomains, the number of iterations stays relatively stable at the price of a larger augmentation size in the case of the diffusion and convection-diffusion problems. As for the Helmholtz problem, the

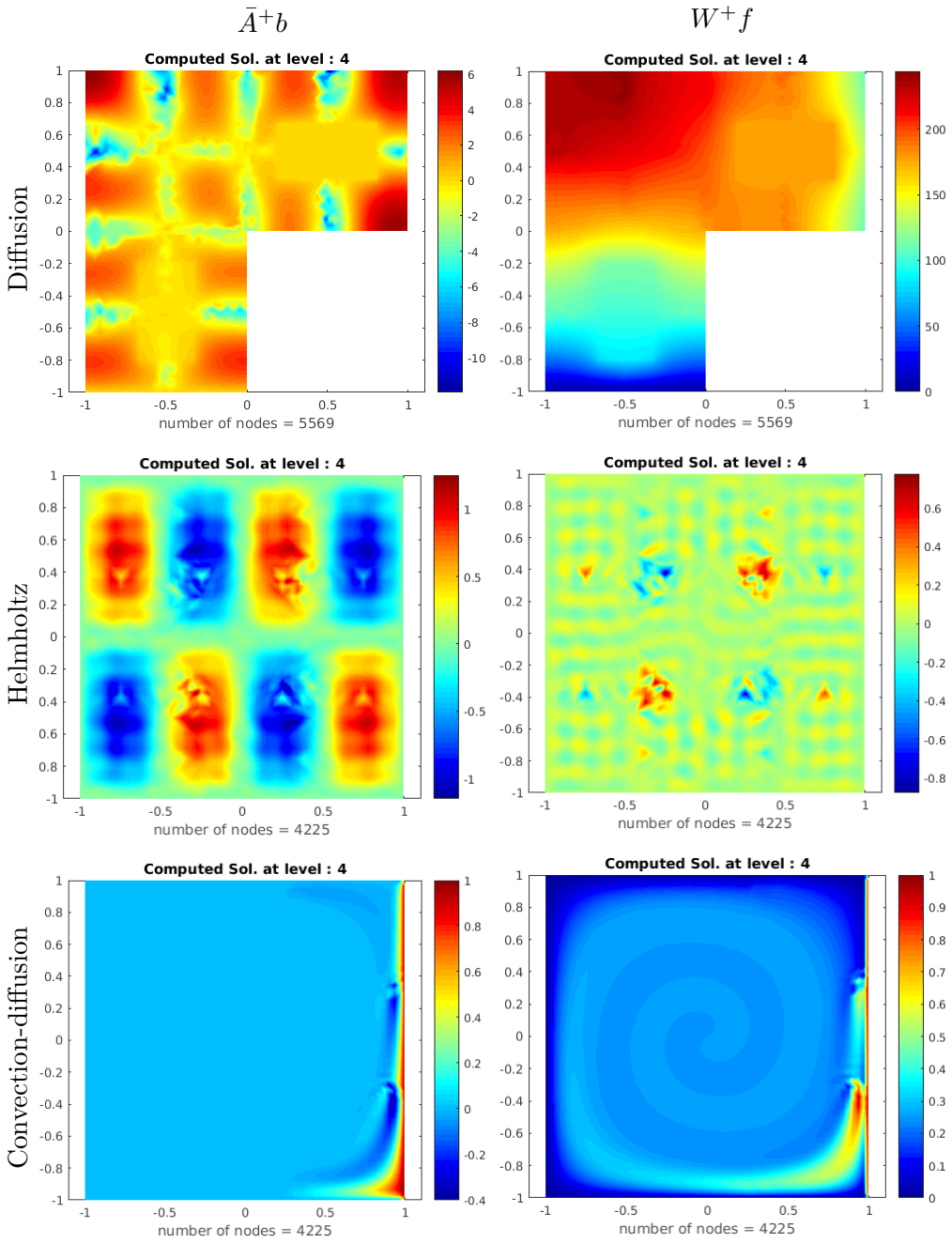


Figure IV.10 Application of the augmentation from the relaxed augmentation method for the problems in $\mathcal{P}_{small}$. Representation of the 2 solution parts, one from the augmented partitions and the other from the closure equations..

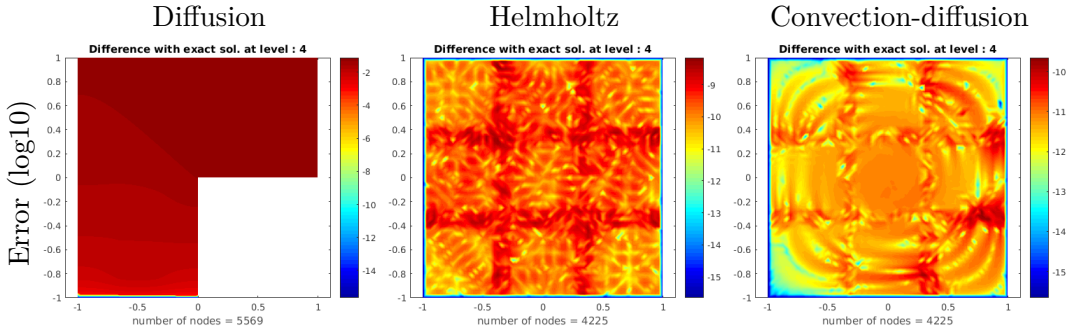


Figure IV.11 Error (in log10) from the application of the augmentation from the relaxed augmentation method for the problems in $\mathcal{P}_{small}$.

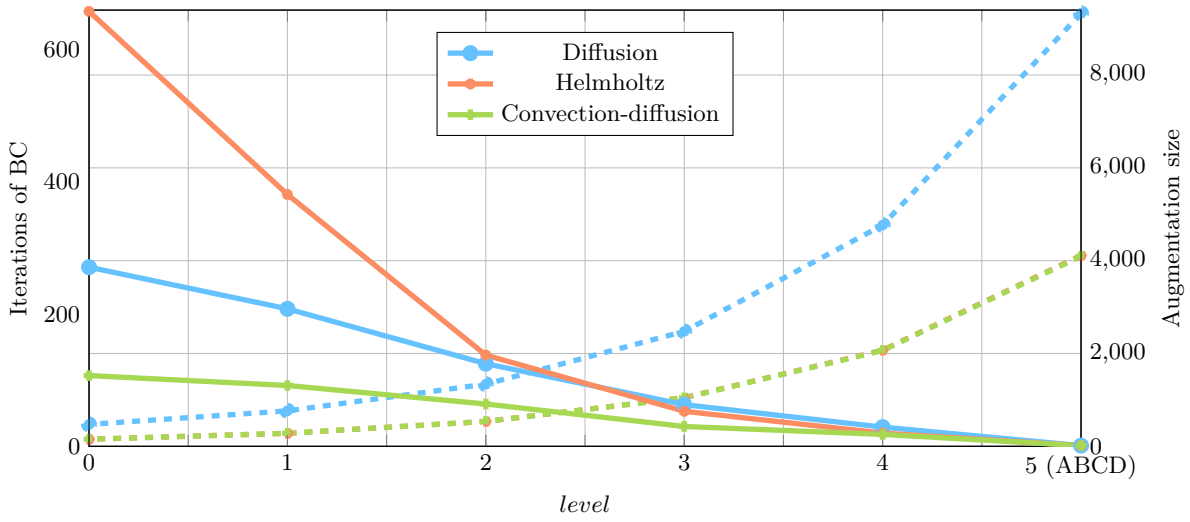


Figure IV.12 Number of iterations *(plain lines, left y-axis)* and size of the augmentation *(dashed lines, right y-axis)* for the BC method applied on $\overline{A}$, obtained using the different coarse grid levels for the test set $\mathcal{P}_{large}$. The augmentation sizes for Helmholtz and convection-diffusion completely overlap

number of iterations follows the same general trend but with peaks regularly appearing.

After handling the part of the solution which corresponds to $\overline{A}$, remains the problem of dealing with the additional constraints W. Considering the small diffusion problem in $\mathcal{P}_{small}$, Figure IV.14 displays the spectrum of the Schur complement S as computed with the pure augmentation from ABCD, and with the augmentation from C-ABCD applied on the coarsest level. We observe that the spectrum of S for the 2 methods is very similar. Relaxing the augmentation does not seem to make the conditioning of

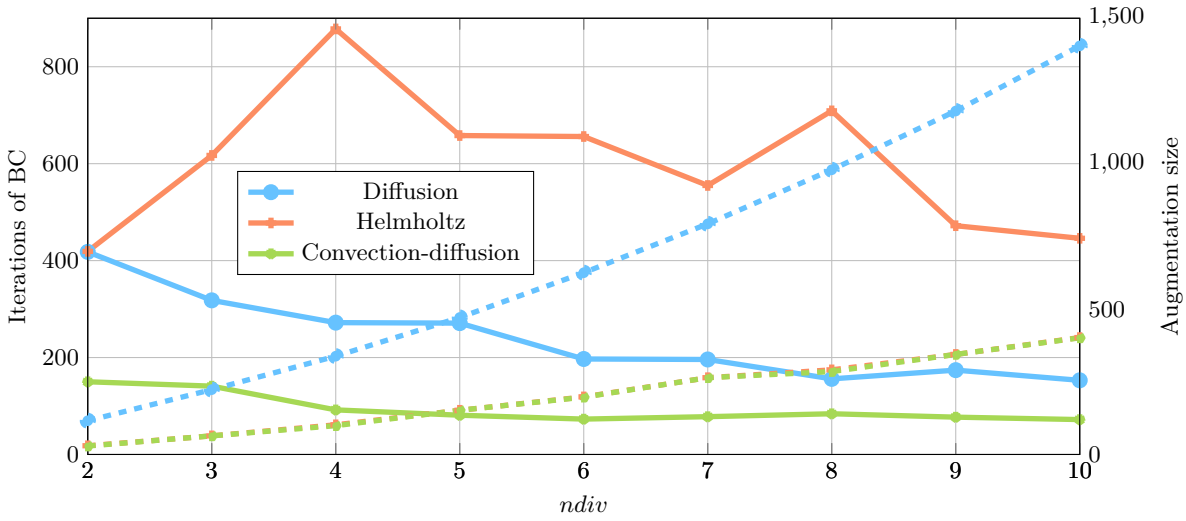


Figure IV.13 Number of iterations *(plain lines, left y-axis)*, and augmentation size *(dashed lines, right y-axis)* for the BC method applied on $\overline{A}$, using different number of partitions: $3ndiv^2$ for the Diffusion (IV.4), and $ndiv^2$ for Helmholtz (IV.2) and convection-diffusion problems (IV.3). The augmentation sizes for Helmholtz and Convection-Diffusion completely overlap.

the resulting Schur complement worse (nor better unfortunately, but this is the *"law of conservation of nastiness"*).

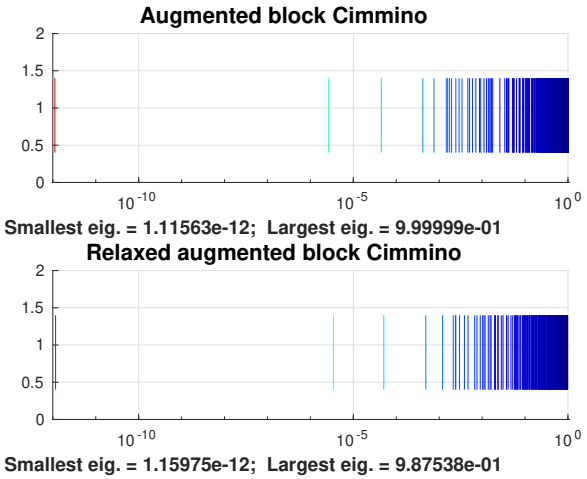


Figure IV.14 Spectrum of the Schur complement S obtained with *(Above)* the ABCD method, and *(Below)* the C-ABCD method for the heterogeneous diffusion problem in $\mathcal{P}_{small}$.

IV.3 A block Cimmino approximation of the closure equations

Previously, we considered that the block of additional constraints W was computed exactly, e.g. using a direct solver on the normal equations, and thus adds no numerical effect on the convergence of the block Cimmino method. As previously stated, the computation of W involves the normal equations $(\overline{A}\,\overline{A}^T)^{-1} = (AA^T + CC^T)^{-1}$ which solution with a direct solver is not realistic.

1. Indeed, due to the loss of orthogonality between partitions, $(\overline{A}\,\overline{A}^T)^{-1}$ is not block diagonal anymore, and as we have said before, the sum of projections for the partitions $\overline{A}_i$ does not correspond to the projection on the range of $\overline{A}^T$.
2. So, to build W, and thus the Schur matrix S, we need to consider an iterative approach. Doing so, we also need to investigate the numerical quality of the approximation of W, and its impact on the global method.
3. Finally, we still have to figure out how to perform the solution with the Schur matrix S, which can be ill-conditioned, though much smaller. This can be done at the expense of building and factorising S, but it might also be computationally more effective to consider iterative solutions of S.

IV.3.1 Construction of W

We can express W^T as the following sum

$$W^T = (I_q - \overline{P})Y^T = Y^T - \overline{A}^+\overline{A}Y^T.$$

Denoting $Z = \overline{A}^+\overline{A}Y^T$, the projection of Y^T on $\mathcal{R}(A^T)$, we may compute this through the minimum norm solution of the underdetermined system with multiple right hand sides

$$\overline{A}Z = \overline{A}Y^T,$$

which can be obtained considering again the block Cimmino iterations.
In ABCD, the construction of S requires the computation of the projection $\overline{A}^+\overline{A}e_k = \sum_{i=1}^{p} \overline{A}_i^+\overline{A}_i e_k$ for each canonical vector e_k of Y^T which is obtained with one sum of projectors only. Due to the structure of the augmentation, in which $\overline{A}_i e_k \neq 0$ only for a small subset of the partitions, we have the opportunity to obtain the entries of S in an embarrassingly parallel as detailed in [133].

The problem is that in C-ABCD, due to the loss of orthogonality between partitions,

$\overline{A}^+\overline{A} \neq \sum_{i=1}^{p} \overline{A_i}^+\overline{A_i}$, and thus all partitions are involved iteratively in the construction of the whole W^T. In order to make the best use of the BLAS3 kernels, we can decompose $Y^T \in \mathbb{R}^{m\times q}$ into blocks of columns $Y_k^T \in \mathbb{R}^{m\times q_k}$, with q_k a fixed block size. We then apply the block Cimmino iterations on

$$\overline{A}Z_k = \overline{A}Y_k^T, \tag{IV.14}$$

to obtain each part $Z_k \in \mathbb{R}^{m\times q_k}$ of $Z \in \mathbb{R}^{m\times q}$. As we have seen, the convergence for these iterations applied on $\overline{A}$ should be fast and linear, the issue is that we may have to run the BC iterations on a lot of separate blocks, depending on the size of the augmentation and how many blocks of q_k there will be.

Depending on the stopping criteria used to stop the iterations, W will approximately be orthogonal to $\overline{A}$. In that case, we can no longer split the solution in 2 parts as in (IV.13). Instead, we add the block W as a full-fledged partition for a global BC scheme which computes the final solution of the augmented system (IV.12) as an outer iteration (which hopefully requires very few iterations, as we expect near orthogonality). There is a trade-off to find between setup cost and solution cost, when setting the stopping criteria. At one extreme, the computation of W is very accurate. In this case, while its computation is expensive, W is very close to be orthogonal to $\overline{A}$ and does not affect the fast linear convergence (same behaviour as ABCD). At the other extreme, W is a crude approximation, far from orthogonal to $\overline{A}$. In that case, the computation is much cheaper, but the convergence of the block Cimmino iterations can display plateaus again (same behaviour as BC).

Following the theory from Section II.3, and assuming A is full rank, the solution of the enlarged system (IV.12) is equivalent to the solution x^* of the original system (IV.8) as long as W is full rank. $Z = \overline{A}^+\overline{A}Y^T$ is approximated through an iterative scheme with an arbitrary accuracy, and we need to keep the consistency of the system with a right hand side f updated that fits with the computed value of Z. As f enforces the

additional variables to 0, it is expressed as

$$\begin{aligned} f = W \begin{bmatrix} x^* \\ 0 \end{bmatrix} &= (Y - Z^T) \begin{bmatrix} x^* \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & I_q \end{bmatrix} \begin{bmatrix} x^* \\ 0 \end{bmatrix} - Z^T \begin{bmatrix} x^* \\ 0 \end{bmatrix} \\ &= -Z^T \begin{bmatrix} x^* \\ 0 \end{bmatrix}, \end{aligned} \tag{IV.15}$$

and f can then be constructed together with Z in a single block-CG algorithm, as detailed in Algorithm 11.

Algorithm 11 Stabilised Block Conjugate Gradient acceleration of the row-BC applied to the simultaneous construction of the additional constraints $W = \overline{A}^{+}\overline{A}Y^T$ and right hand side f for C-ABCD

1: $Z^{(0)} = 0$, $R^{(0)} = \overline{H}Y^T - \overline{H}Z^{(0)} = \overline{A}^T\,\overline{D}^{-1}\overline{A}Y^T$
2: $f^{(0)} = 0$, $r_f^{(0)} = K - \overline{H}f^{(0)} = \overline{A}^T\,\overline{D}^{-1}b$
3: $\overline{R}^{(0)} = R^{(0)}\gamma_0^{-1}$ such that $\overline{R}^{(0)^T}\overline{R}^{(0)} = I$
4: $\overline{r}_f^{(0)} = \gamma_0^{-T} Y r_f^{(0)}$
5: $\overline{P}^{(0)} = \overline{R}^{(0)}\beta_0^{-1}$ such that $\overline{P}^{(0)^T}\overline{HP}^{(0)} = I$
6: $\overline{p}_f^{(0)} = \beta_0^{-T}\overline{r}_f^{(0)}$
7: $\lambda_0 = \beta_0^{-T}$
8: **for** $j = 0, 1, 2..$ until convergence **do**
9: $\quad \alpha_j = \lambda_j(\prod_{i=j}^{0}\gamma_i)$
10: $\quad Z^{(j+1)} = Z^{(j)} + \overline{P}^{(j)}\alpha_j$
11: $\quad f^{(j+1)} = f^{(j)} - \alpha_j^T\,\overline{p}_f^{(j)}$
12: $\quad \overline{R}^{(j+1)} = (\overline{R}^{(j)} - \overline{HP}^{(j)}\lambda_j)\gamma_{j+1}^{-1}$ such that $(\overline{R}^{(j+1)})^T\,\overline{R}^{(j+1)} = I$
13: $\quad \overline{r}_f^{(j+1)} = \gamma_{j+1}^{-T}(\overline{r}_f^{(j)} - \lambda_j^T\overline{P}^{(j)^T} r_f^{(0)})$
14: $\quad \delta_j = \beta_j\gamma_{j+1}^T$
15: $\quad \overline{P}^{(j+1)} = (\overline{R}^{(j+1)} + \overline{P}^{(j)}\delta_j)\lambda_j^T$ such that $\overline{P}^{(j+1)^T}\overline{HP}^{(j+1)} = I$
16: $\quad \overline{p}_f^{(j+1)} = \lambda_j(\overline{r}_f^{(j+1)} + \delta_j^T\,\overline{p}_f^{(j)})$
17: $\quad \lambda_{j+1} = \beta_j^{-T}$
18: **end for**

This algorithm is the one developed for the minimum norm solution of underdetermined systems, in which we have just added the updates for the desired right hand side

f. In the algorithm, $\overline{H} = \overline{A}^T \overline{D}^{-1} \overline{A}$, with $\overline{D} = blkdiag(\overline{A}_1 \overline{A}_1^T, \ldots, \overline{A}_p \overline{A}_p^T)$, is the block Cimmino iteration matrix H^{row} expressed on the augmented matrix $\overline{A}$. At each iteration j, we update $Z^{(j)}$ and $f^{(j)}$ using $\overline{R}^{(j)}$ and $\overline{r}_f^{(j)}$, as well as the conjugate directions $\overline{P}^{(j)}$ and $\overline{p}_f^{(j)}$. In order to respect the property (IV.15), we prove that for all iterations j, there exists $\widetilde{Z}^{(j)}$ such that

$$Z^{(j)} = \overline{A}^T \widetilde{Z}^{(j)^T}, \text{ and } f^{(j)} = -\widetilde{Z^{(j)}} b \tag{IV.16}$$

Therefore, we have $f^{(j)} = Z^{(j)^T} \begin{bmatrix} x^* \\ 0 \end{bmatrix} = \widetilde{Z}^{(j)} \overline{A} \begin{bmatrix} x^* \\ 0 \end{bmatrix} = \widetilde{Z}^{(j)} b$, which gives us the appropriate expression of f to maintain $\begin{bmatrix} x^* \\ 0 \end{bmatrix}$ as the solution of the enlarged system,

Proof For the sake of this proof, we additionally prove that for all iterations j, there exists $\widetilde{R}^{(j)}$ and $\widetilde{P}^{(j)}$ such that

$$\overline{R}^{(j)} = \overline{A}^T \widetilde{R}^{(j)^T}, \text{ and } \overline{r}_f^{(j)} = \widetilde{R}^{(j)} b, \tag{IV.17}$$

$$\overline{P}^{(j)} = \overline{A}^T \widetilde{P}^{(j)^T}, \text{ and } \overline{p}_f^{(j)} = \widetilde{P}^{(j)} b. \tag{IV.18}$$

We now prove the properties (IV.16), (IV.17) and (IV.18) recursively. At iteration 0, we take $f^{(0)} = Z^{(0)} = 0$, thus property (IV.16) is given by $\widetilde{Z}^{(0)} = 0$. Also we have

$$\begin{aligned} \overline{R}^{(0)} &= R^{(0)} \gamma_0^{-1} = \overline{A}^T \overline{D}^{-1} \overline{A} (Y^T - Z^{(0)}) \gamma_0^{-1}, \\ \overline{r}_f^{(0)} &= \gamma_0^{-T} Y r_f^{(0)} = \gamma_0^{-T} Y \overline{A}^T \overline{D}^{-1} (b - \overline{A} f^{(0)}). \end{aligned}$$

As $f^{(0)} = Z^{(0)} = 0$, if we take $\widetilde{R}^{(0)} = \gamma_0^{-T} Y \overline{A}^T \overline{D}^{-1}$ then property (IV.17) is satisfied. Using this result, we can write

$$\begin{aligned} \overline{P}^{(0)} &= \overline{R}^{(0)} \beta_0^{-1} = \overline{A}^T \widetilde{R}^{(0)^T} \beta_0^{-1}, \\ \overline{p}_f^{(0)} &= \beta_0^{-T} \overline{r}_f^{(0)} = \beta_0^{-T} \widetilde{R}^{(0)} b, \end{aligned}$$

and property (IV.18) is satisfied for $\widetilde{P}^{(0)} = \beta_0^{-T} \widetilde{R}^{(0)}$.

We now consider the 3 properties are satisfied at iteration j and prove them at iteration $j+1$. Considering first the residuals, from (IV.17) and since $\overline{H} = \overline{A}^T \overline{D}^{-1} \overline{A}$, we have

$$\begin{aligned} \overline{R}^{(j+1)} &= (\overline{R}^{(j)} - \overline{H P}^{(j)} \lambda_j) \gamma_{j+1}^{-1} = \overline{A}^T (\widetilde{R}^{(j)^T} - \overline{D}^{-1} \overline{A P}^{(j)} \lambda_j) \gamma_{j+1}^{-1}, \\ \overline{r}_f^{(j+1)} &= \gamma_{j+1}^{-1} (\overline{r}_f^{(j)} - \lambda_j^T \overline{P}^{(j)^T} r_f^{(0)}) = \gamma_{j+1}^{-1} (\widetilde{R}^{(j)} - \lambda_j^T \overline{P}^{(j)^T} \overline{A}^T \overline{D}^{-1}) b, \end{aligned} \tag{IV.19}$$

and we take $\widetilde{R}^{(j+1)} = \gamma_j^{-1}(\widetilde{R}^{(j)} - \lambda_j^T \overline{P}^{(j)^T} \widetilde{R}^{(j)})$ to satisfy (IV.17). Similarly, using (IV.19) and from (IV.18), we have

$$\begin{aligned}
\overline{P}^{(j+1)} &= (\overline{R}^{(j+1)} + \overline{P}^{(j)}\delta_j)\lambda_j^T = \overline{A}^T(\widetilde{R}^{(j+1)^T} + \widetilde{P}^{(j)^T}\delta_j)\lambda_j^T, \\
\overline{p}_f^{(j+1)} &= \lambda_j(\overline{r}_f^{(j+1)} + \delta_j^T\, \overline{p}_f^{(j)}) = \lambda_j(\widetilde{R}^{(j+1)} + \delta_j^T\, \widetilde{P}^{(j)})b,
\end{aligned}$$

and property (IV.18) is satisfied for $\widetilde{P}^{(j+1)} = \lambda_j(\widetilde{R}^{(j+1)} + \delta_j^T\, \widetilde{P}^{(j)})$. Finally, we can conclude for property (IV.16) with

$$\begin{aligned}
\overline{Z}^{(j+1)} &= Z^{(j)} + \overline{P}^{(j)}\alpha_j = \overline{A}^T(\widetilde{Z}^{(j)^T} + \widetilde{P}^{(j)^T}\alpha_j), \\
\overline{p}_f^{(j+1)} &= f^{(j)} - \alpha_j^T\, \overline{p}_f^{(j)} = -(\widetilde{Z}^{(j)} + \alpha_j^T\, \widetilde{P}^{(j)})b, \\
\widetilde{Z}^{(j+1)} &= \widetilde{Z}^{(j)} + \alpha_j^T\, \widetilde{P}^{(j)},
\end{aligned} \tag{IV.20}$$

which completes the proof.

In the following, we call BC-W the approximation of W and f using Algo. 11, to distinguish it from the BC method which is used to compute the final solution. The choice for the block-CG size, i.e. the number of columns in the blocks Y_k, is open and should be chosen depending on the characteristics of the target computer.

IV.3.2 Approximation of W and convergence of a global BC

In this section, we approximate W and f using the previously introduced algorithm with a default threshold of $\epsilon = 10^{-10}$ on the backward error. Using the obtained approximation of W as an additional partition, we apply the BC method on the resulting enlarged system obtained for the set of large tests $\mathcal{P}_{large}$, and we observe its convergence behaviour depending on several parameters. In the augmentation, we enforce orthogonality between partitions in AP with the grid level 2 as coarse grid.

IV.3.2.a Block-CG size for the approximation of W

First, we vary the block-CG size s (i.e. the value of q_k explained in (IV.14)) for the approximation of W between 1 and 16 by powers of 2. Figure IV.15 presents for the 3 problems in $\mathcal{P}_{large}$ the evolution of it_s, the average number of iterations for convergence of a block Z_k with a block size s, relative to it_1, the average number of iterations with a block size of 1. This evolution is represented by the ratio it_1/it_s. With C-ABCD, that implements the relaxed augmentation on the coarse grid, the convergence of the block-CG

applied on $\overline{A}$ is close to linear, thus no plateau in the convergence is present and needs to be reduced. In the case of the Helmholtz problem, this is confirmed by the ratio in Figure IV.15, which is close to the block size itself meaning that using a block size s for the block-CG iterations has the same behaviour as s times the convergence of a simple CG. As a result, the benefit from an increased block-CG size is only expected from the BLAS3 effect, i.e. a reduced execution time from more efficient matrix operations at each iteration. In the case of the diffusion and convection-diffusion problems, the reduction of iterations is respectively 1.25 and 2 times more efficient than the equivalent number of CG, this shows that the convergence of the global BC in this case is not exactly linear and has a little room for improvement.

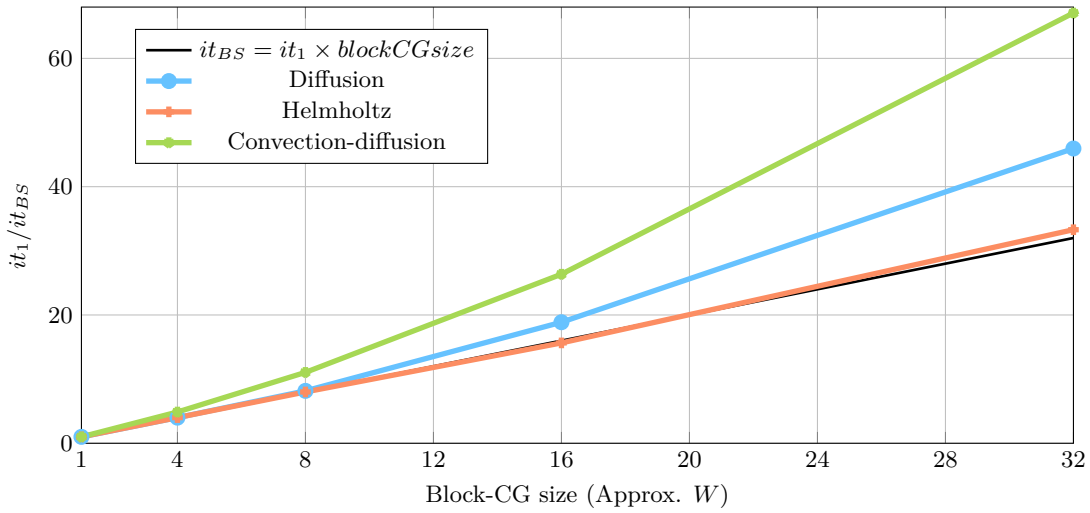


Figure IV.15 Evolution of the average number of iterations to approximate a part of W of size s relative to the average number of iterations with a block size of 1. The results for Diffusion and Convection-Diffusion completely overlap.

As stated above, once the block W has been approximated, it is used as an additional partition for the application of a global BC method on the augmented system. Table IV.3 displays the number of iterations for the convergence of this global BC, depending on the block size used in BC-W. As expected, the number of iterations are not varying much depending on the chosen block size. The variations are explained by little differences in which approximations of W was obtained. Because here we do not compute W exactly, the obtained number of iterations is not exactly equal to the results in Figure IV.12.

Block size	Diffusion	Convection-Diffusion	Helmholtz
1	206	112	49
4	204	112	58
8	195	112	55
16	180	112	59
32	183	112	59

Table IV.3 Evolution of the number of iterations for the convergence of the global BC method applied on the system augmented on the coarse grid level 2, depending on the block-CG size for BC-W.

IV.3.2.b Stopping criterion for the Block-CG approximation of W

The approximated block W depends on the stopping criterion we choose for BC-W. Now, we approximate W with a varying stopping criterion on the backward error $\epsilon \in \{1 \cdot 10^{-12}, 1 \cdot 10^{-10}, 1 \cdot 10^{-8}, 1 \cdot 10^{-6}, 1 \cdot 10^{-4}\}$. Figure IV.16 presents the evolution of the average number of iterations for the convergence of BC-W. Again, the Helmholtz problem stands as a special case since the number of iterations for BC-W decreases linearly with the value of *epsilon*. As for the two other problems, the iterations of both BC-W and the global BC do not change for ϵ between $1 \cdot 10^{-12}$ and $1 \cdot 10^{-10}$. Then when ϵ increases, W gets further away from being orthogonal to $\overline{A}$, as its numerical quality degrades, thus slowing down the iterations of the global BC method applied on the enlarged system. Figure IV.17 shows the convergence profile of the global BC depending on ϵ for the Helmholtz problem. With $\epsilon > 1 \cdot 10^{-8}$, plateaus start to appear. A trade-off must be found between a cheaper construction of W and a fast convergence of the global BC. For the 3 test cases, it seems that an intermediate $\epsilon = 1 \cdot 10^{-8}$ would be a reasonnable choice, but this may also be relaxed when the problems are less ill-conditioned.

Table IV.4 displays the conditioning of the Schur complement matrix S in W. Another effect of decreasing the accuracy of the approximation of W is a better conditioning of S. In a very informal way, we call this the "law of conservation of nastiness", as accelerating the global BC method is often done at the cost of a more expensive approximation of W, combined with a worse conditioning of S, and vice-versa.

In summary, the block-CG approximation of the closure equations W and the right hand side f does not depend much on the choice of a good block size, except for a little improvement in cases where the convergence is not exactly linear like the convection-diffusion and diffusion problems. Also, when setting the desired accuracy for the approximation of W, a trade-off must be found between decreasing the cost of the BC-W iterations, and having plateaus reappearing in the convergence profile of the global BC iterations.

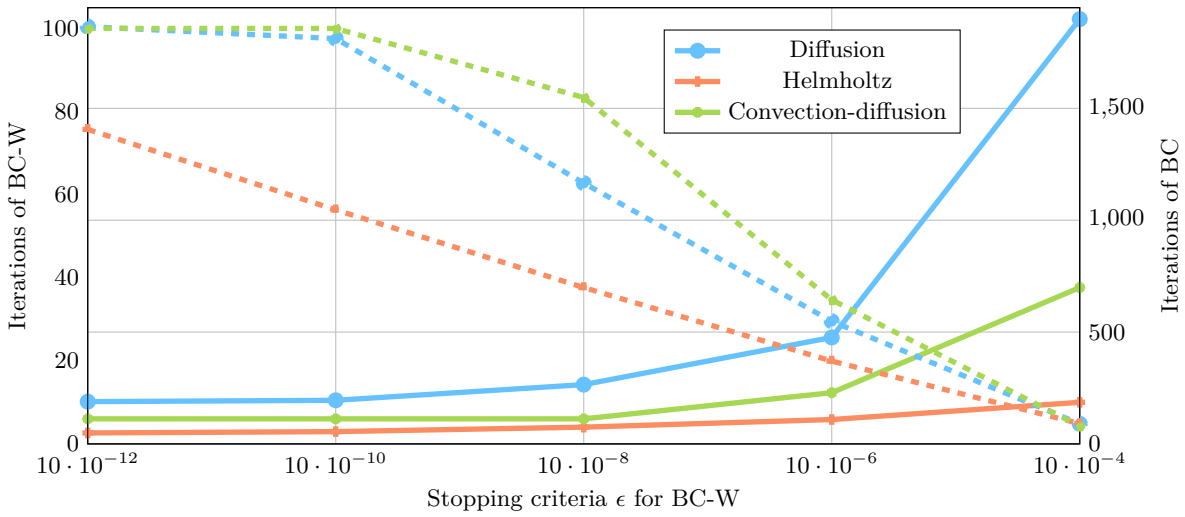


Figure IV.16 For the problems in $\mathcal{P}_{large}$, we show the evolution of *(dashed line with left y-axis)* the average number of iterations for BC-W and *(plain line with right y-axis)* the number of iterations for the global BC depending on the stopping criterion ϵ chosen for the approximation of W.

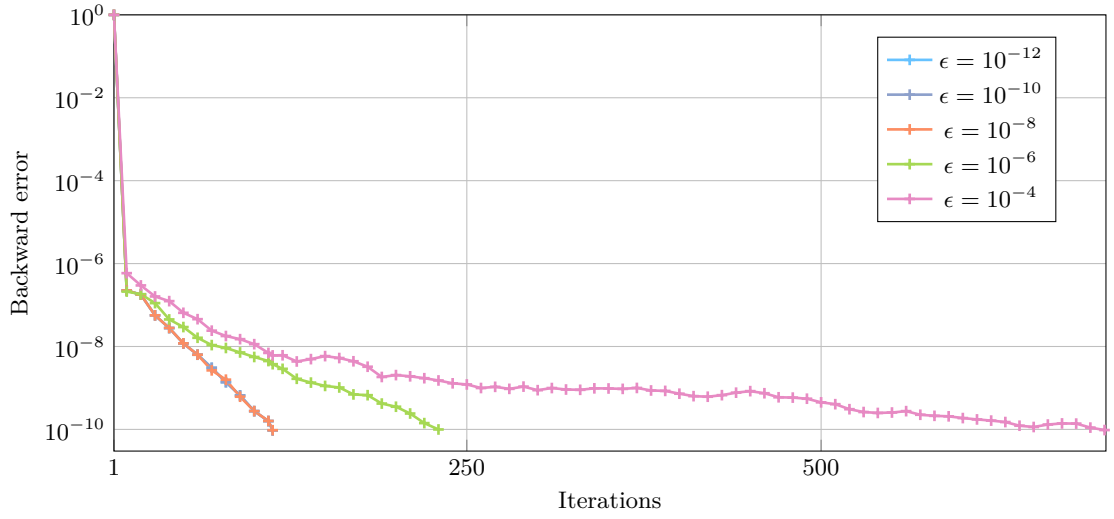


Figure IV.17 Convergence of BC applied on the large Helmholtz problem with a threshold ϵ for BC-W varying between 10^{-12} and 10^{-4}. The convergence for $\epsilon = 10^{-12}$, $\epsilon = 10^{-10}$, and $\epsilon = 10^{-8}$ completely overlaps.

One aspect of the matrix W we haven't mentioned before is that it is almost completely dense once approximated with the BC-W algorithm. The construction of the block W

ϵ	**Diffusion**	**Convection-Diffusion**	**Helmholtz**
$1.0 \cdot 10^{-12}$	$2.0 \cdot 10^{4}$	$1.1 \cdot 10^{3}$	$5.1 \cdot 10^{3}$
$1.0 \cdot 10^{-10}$	$1.1 \cdot 10^{4}$	$1.1 \cdot 10^{3}$	$4.7 \cdot 10^{3}$
$1.0 \cdot 10^{-8}$	$6.9 \cdot 10^{3}$	$1.1 \cdot 10^{3}$	$1.4 \cdot 10^{3}$
$1.0 \cdot 10^{-6}$	$2.2 \cdot 10^{3}$	$1.2 \cdot 10^{3}$	$5.5 \cdot 10^{2}$
$1.0 \cdot 10^{-4}$	$9.6 \cdot 10^{1}$	$1.3 \cdot 10^{2}$	$7.3 \cdot 10^{1}$

Table IV.4 Conditioning of S obtained with C-ABCD applied on the large problems in the set $\mathcal{P}_{large}$, with varying ϵ threshold for BC-W between 10^{-12} and 10^{-4}.

is computationally heavy because it amounts to applying sequentially a CG to each canonical vector in Y. Due to its density, the factorisation of the projection system corresponding to W using a direct solver is also an issue in terms of computational and memory cost due to its high density.

We may compensate for the computational cost when solving a linear system with multiple right hand sides, indeed W only needs to be computed and solved once. As for the approximation of f, we need to update $\widetilde{Z}^{(j)}$ from (IV.20) instead of $f^{(j)}$ and keep it for later right hand sides. In that case, it may be worthy to approximate W with a high accuracy so that it is orthogonal to $\overline{A}$ and only the application of BC on $\overline{A}$ is necessary for each right hand side as in (IV.13). We propose in Appendix C.2 a sketch for the parallelisation scheme of the ABCD-Solver corresponding to C-ABCD with the construction of W.

However, the density of W can make it ineffective in practice, depending on the number of rows in W (e.g. the size of the augmentation). We have therefore tried to investigate other approaches, in which we may not need to build W explicitly, but instead we can use S or its inverse in an iterative fashion through matrix-vector products.

IV.4 Implicit approximation of the Schur complement

The role of this last section is to present the latest tracks we have explored as an opening for future research. In the first part of chapter IV, we have introduced a relaxed augmentation method that enables a fast linear convergence of the BC method applied on partitions augmented using a coarse grid representation of the problem. While an explicit construction of the closure equations is costly in practice, as seen above, the other possibility is to implicitly apply the inverse of the Schur complement through an iterative procedure, with matrix-vector products based on S (without the need to build S explicitly).

IV.4.1 Iterative solution with S

Let's start with some reminders. Once the original matrix has been augmented with the relaxed augmentation technique applied on the coarse grid, we obtain the augmented matrix $\overline{A}$ where the partitions $\overline{A}_i$ are not purely orthogonal. The exact additional constraints are then obtained as

$$W = \begin{bmatrix} B & S \end{bmatrix} = Y(I - \overline{P}),$$

where $Y = \begin{bmatrix} 0 & I_q \end{bmatrix}$, and $\overline{P} = \mathcal{P}_{\mathcal{R}(\overline{A}^T)} = \overline{A}^+\overline{A}$. The solution of the augmented system is then in two parts, corresponding respectively to $\overline{A}$ and W, expressed as

$$\begin{bmatrix} x^* \\ 0 \end{bmatrix} = \overline{A}^+ b - (I_q - \overline{P})Y^T S^{-1} Y \overline{A}^+ b,$$

where the vector x^* is the solution of the original system. This expression is composed of multiple steps

- ✓ $\overline{A}^+ b$ is obtained as the minimum-norm solution (m.n.s.) of $\overline{A}x = b$ using the BC method with linear convergence (because of the relaxed augmentation),
- ✓ $I_q - \overline{P}$ is a projection, and $\overline{P} = \overline{A}^+\overline{A}$ can be applied to a vector v through the m.n.s. of $\overline{A}x = \overline{A}v$ using BC again.
- × S^{-1} is the problematic part first of all. The use of S^{-1} relies on the assumption that S is still a good approximation of the normal equations of W, as $WW^T = Y(I - \overline{\mathcal{P}})^2 Y^T$, and if $\overline{P}$ is computed with a relatively good accuracy then

  $$S = Y(I_q - \overline{P})Y^T.$$

  Additionally, since we do not want to construct W, we need to work iteratively using matrix products with $Y(I - \mathcal{P})Y^T$, which in itself implies an inner BC iteration which realising the product with $\overline{P}$.

Since S has been proven SPD, as long as A is full row rank, we propose to use a CG algorithm to solve with S. However, the conditioning of S has been observed as possibly ill-conditioned in IV.3.2.b. We should look for a good preconditioner $M^{-1} \approx S^{-1}$ so as to avoid too many inner-outer steps, and consider a preconditioned CG on the equation

$$M^{-1}Y(I_q - \overline{P})Y^T x = v.$$

IV.4.2 Preconditioning S

Since we do not construct the matrix S explicitly, most classical preconditioning methods cannot be applied directly, such as block-Jacobi, Gauss-Seidel or incomplete Cholesky factorisation. We must look for a preconditioner based on the algebraic forms of S or S^{-1}. We present here our current research tracks. In Table IV.5, we give the conditioning of the matrix S obtained with the augmentation based on the second finest grid level (the conditioning of S worsens) for the 3 small problems in $\mathcal{P}_{small}$, as well as the conditioning of the preconditioned matrix $M^{-1}\,S$, as computed in MATLAB for some choices of M that we shall describe.

M^{-1}	Prolongation	$\kappa(M^{-1}S)$		
		Diffusion	Convection-diffusion	Helmholtz
I_q	$\times$	$5.2 \cdot 10^{10}$	$1.4 \cdot 10^{8}$	$5.2 \cdot 10^{7}$
$(I_q - \sum_{i=1}^{p} \overline{A}_i^{+}\overline{A}_i)^{-1}$	$\times$	$3.4 \cdot 10^{13}$	$1.3 \cdot 10^{6}$	$1.8 \cdot 10^{5}$
$I_q + C^T P(P^T A A^T P)^{-1} P^T C$	P	$1.9 \cdot 10^{9}$	$9.8 \cdot 10^{5}$	$3.8 \cdot 10^{3}$
	$\mathcal{F}^{FR}(P)$	$4.4 \cdot 10^{10}$	$5.8 \cdot 10^{5}$	$1.7 \cdot 10^{5}$

Table IV.5 Conditioning of the Schur complement before and after preconditioning with various methods.

Sum of projections

A naive idea is to consider that the normal equations $\overline{A}\,\overline{A}^T$ are not too far from being block diagonal. This would mean that the augmentation technique applied on the coarse grid gives augmented partitions close to orthogonal. We may then consider only the block diagonal elements $\overline{A}_i\overline{A}_i^T$ of the normal equations and, in the same way as S is constructed in ABCD in an embarrassingly parallel way, we construct M as the restricted sum of projections

$$M = I_q - Y\left(\sum_{i=1}^{p} \mathcal{P}_{\mathcal{R}(\overline{A}_i)}\right) Y^T = I_q - \sum_{i=1}^{p} C_i^T(\overline{A}_i\overline{A}_i^T)^{-1}C_i,$$

where $C_i,\ i = 1, \ldots, p$, are the blocks of C corresponding to the partitions. As observed in Table IV.5, this preconditioner actually helps a little to improve the conditioning of S for the convection-diffusion and Helmholtz problems, but the conditioning worsens for the diffusion. In the previous section, we have observed that the construction of W requires several iterations of the BC method to obtain a satisfactory accuracy. It is then

natural that the preconditioner cannot be constructed from a single sum of projections.

Coarse grid approximation of S^{-1}

We have looked at the previous approach because it gives an approximation for S cheap to construct and inverse. Another original angle of attack to the problem is to use the algebraic form for the inverse of S given by

$$S^{-1} = I_q + C^T(AA^T)^{-1}C.$$

While S depends on the normal equations of the augmented partitions, S^{-1} depends on the normal equations of the original system. In the context of this chapter, a natural idea is to use the hierarchy of grids we have at hand, to build the augmentation block. Basically, we can choose a level of grid, e.g. the one used to build the relaxed augmentation, and exploit the prolongation operator between the finest grid and this chosen level. Then, we can approximate the inverse of the normal equations with

$$(AA^T)^{-1} \approx P(P^TAA^TP)^{-1}P^,$$

within the expression for S^{-1} above. In Table IV.5, we display the conditioning of the preconditioned matrix, with this setting, using coarse grid level 1 i.e. the second finest level of grid. We observe an improvement of the conditioning in all cases. In practice, we would apply implicitly $(P^TAA^TP)^{-1}$ on a vector v through the solution of the equation $(P^TAA^TP)^{-1}x = v$, e.g. using a direct solver. Choosing a coarser grid level means a smaller column dimension for P and a less expensive application of the approximated normal equations, but also a potentially less accurate approximation.

Splitting the prolongation operator

While the previous 2-level preconditioner improves the conditioning of S a little, it is clear that the approximation of the inverse of the normal equations applied on C is missing some information. The columns of C express the decoupling between subdomains through the augmentation technique, and the previous approximation may lose this information because the coarse grid does not incorporate the corresponding splitting. Indeed, on the coarse grid, the interface itself is swallowed in the middle of larger finite elements.

Here, we attempt to improve the prolongation operator by splitting the columns of P following the augmentation technique applied on the matrix A to get P_{split}. Using this split prolongation operator, we compute the same preconditioner as previously. Again, the

conditioning of S is almost untouched, except a small fraction for the convection-diffusion problem. All-in-all, we did not yet succeed in finding an appropriate preconditioner to fast convergence for the PCG method in all our test cases.

Perspective and concluding remarks

We have introduced a method based on a variant of the ABCD method where the augmentation is done on a coarse level of grid. In this approach, it is then possible to control the number of additional variables through the choice of a more or less coarse level. Due to these additional variables and the introduction of the corresponding closure equations, it possible to obtain a linear convergence of the BC method applied on the augmented matrix $\overline{A}$.

The closure equations $W = \begin{bmatrix} B & S \end{bmatrix}$ are based on a projection on the nullspace of $\overline{A}$ which, contrary to the ABCD method, cannot be obtained with a single projection on the augmented partitions $\overline{A}_i$. It is instead obtained with a convergence of the BC method on $\overline{A}$ applied on a set of canonical vectors, with a simultaneous update of the right hand side f to keep the system consistency. Obtained using this method, the construction of each vector in W, taken separately, is easy since it amounts to applying a fast linear convergence of the BC method using the augmented partitions. However, a few hundreds of such convergence are required and the obtained W is very dense.

Future works:

- A possible improvement for BC-W is to consider that, since the convergence of BC on $\overline{A}$ is linear for all test problems, the distribution of the eigenvalues of $\overline{A}$ is not important, then the polynomial we apply to a random vector inside the block-CG can be reused for any other vectors. This means that in Algo. 11, we can save the small matrices γ and β at every iterations for the block-CG applied on a random vector, to reconstruct the full polynomial applied on all canonical vectors for the construction of W. At each iteration, a sum of projection still needs to be computed to apply $\overline{H}$ but the cost from the stabilisation process vanishes. This approach is only an idea at the moment and would need to be demonstrated, then proven in the general case for a matrix showing linear convergence.
- Also, still based on the fact that the BC method applied on $\overline{A}$ is linear, we can consider an efficient construction of W using Chebyshev iterations on the system

$$\overline{H}x = K, \tag{IV.21}$$

where $\overline{H}$ is the iteration matrix of BC based on $\overline{A}$, and $K = \overline{A}^T \overline{D}^{-1} Y^T$. The

Chebyshev iterations require an estimation of the smallest and largest eigenvalues [75]. We know from [69] that the convergence rate on the error for the CG algorithm is bounded with the relation

$$\left\|e^{(k)}\right\|_H \leq 2\left(\frac{\sqrt{\kappa(H)}-1}{\sqrt{\kappa(H)}+1}\right)^k \left\|e^{(0)}\right\|_H, \tag{IV.22}$$

with $\kappa(\overline{H}) = \lambda_{max}(\overline{H})/\lambda_{min}(\overline{H})$, and we also have that $\rho(\overline{H}) \leq p$ as a sum of p projections. Through the application of a few iterations of the BC method on $\overline{A}$, a stabilised block-CG applied on (IV.21), we obtain the fixed convergence rate $\rho = \left\|e^{(k)}\right\|_H / \left\|e^{(0)}\right\|_H$ of the linear convergence. Then, we can get an upper bound for λ_{min} from (IV.22).

- Remains the problem of the density of W, and an idea is to sparsify W with a carefully constructed filtering of its values based on the values on the diagonal of the Schur complement S. Indeed, the values on the diagonal of S can be small, which explains its relatively ill-conditioning. Also, S is SPD and thus diagonal dominant so it gives an upper bound of the sum of the values on the corresponding rows and columns from which we could base a symmetric relative filtering of the values in S. Additionally, in Section II.3.1.b, it was observed that $BB^T = S(S + I_q)$ and thus $S_{ii}(S_{ii} - 1) = B_{i:}B_{i:}^T$, then we consider filtering the values in the rows of B relatively to the corresponding diagonal element in S. Using an arbitrary filtering on the values of W could otherwise drop crucial information in the system, since we may then lose the information of the small eigenvalues in S. With this filtering, while we keep the information in S, we may lose the consistency of the system. Another filtering approach, which would then guaranty the consistency of the system, would be to drop values in the matrix $\widetilde{Z}$ introduced in (IV.16).

Using the computed approximation of W as an additional partition, we obtain the solution of the complete augmented system with the convergence of a global BC method. The projection on W at each iteration may however be expensive due to its high density.

Future work:

- While we are using the block Cimmino approach, i.e. a method based on a sum of projections, we could use the partition W based on the block Kaczmarz approach [108]. From a current iterate $x^{(k)}$, we would apply the update

$$x_A^{(k+1)} = \overline{A}^+ x^{(k)}$$
$$x^{(k+1)} = W^+ x_A^{(k+1)},$$

where $\overline{A}^+$ is only approximated with several iterations of the BC method applied on $\overline{A}$ with linear convergence. W was constructed to have a very near orthogonality with $\overline{A}$, and the block Kaczmarz method is known to converge faster than block Cimmino in this case [108]. By choosing the number of iterations for the approximation of $\overline{A}^+$, we reduce the number of times we need to use W and thus the associated computational difficulty.

- While we can compute the projection of W using a projection system similarly to the projections on $\mathcal{R}(\overline{A}_i)$, it is possible to decrease the complexity of this projection. The idea is to consider that W was approximated with an accuracy sufficient such that the contained sub-block S is a good approximation of the normal equations of W. The projection can then be computed as

$$W^+ = W^T S^{-1},$$

which implies "only" the inversion of the smaller matrix S. In the end, we could completely remove the block W from this expression, viz.

$$W^+ = (I - \overline{P}) Y^T S^{-1}.$$

where $\overline{P}$ is computed with another application of the BC method on $\overline{A}$. Of course we then suppose that the projection is computed with a high accuracy and that the inverse of S corresponds to the normal equations of the implictely constructed partition W, which remains to be verified in practice.

From the previous statements and using (IV.13), we can then consider approximating S and the projection $\overline{P}$ iteratively with enough accuracy such that we directly get the solution in 3 steps. First the application of a BC convergence on $\overline{A}$ applied on b, then the iterative application of the inverse of S on the obtained result, and finally the projection with yet another application of the BC method on $\overline{A}$ to apply $\overline{P}$. We propose to apply the inverse of S using a Krylov method which requires the implicit application of S using again the projections $\overline{P}$. A good preconditioner is necessary to get a fast convergence from this Krylov solver. The construction of this preconditioner is still an ongoing work.

The precise interpretation of the C-ABCD approach, with respect to the solved PDE, is not fully understood yet. In particular, the way the information is distributed between the augmented partitions and the closure equations, depends highly on the problem, see e.g. Figure IV.10 and Figure IV.11. This distribution of information, even though

linked to a fine and coarse grids through the augmentation, does not seem to display a behaviour similar to multigrid methods where the fine grid serve to smooth the error components while the coarse grid correct this smooth error. Here, both $\overline{A}$ and W seem to work on smooth and oscillatory components. In order to use this different behaviour from the methods we introduced (BC, C-ABCD, and ABCD), we may consider building a classical multigrid framework where the BC method is used as a type of smoother on the finer grid levels and the ABCD method is applied for the coarsest grid level, considering the problem on this level is small enough.

Conclusion and perspectives

> A mathematician is a device for
> turning coffee into theorems.
>
> Alfréd Rényi, often also attributed to
> Paul Erdõs

In this manuscript, we considered the solution of large sparse linear systems with the use of hybrid direct-iterative methods with two main research axis.

Parallel multigrid at large scale

In a first axis, we study the parallel efficiency of multigrid methods at very large scale. In Chapter I, we started with the introduction of the different aspects underlying the world of linear solvers for scientific computing as well as the various PDE problems considered in this thesis. We detailed the two main classes of hybrid solvers, namely the domain decomposition methods (DDM) and the multigrid methods (MG). On large supercomputers, one significant limit for the parallel efficiency of MG schemes is the computation of the solution on the coarse grid, where a standard Krylov solver is applied. We address in the Chapter I the 2 reasons for this limitation

- First, when the linear problem is complex, e.g. with jumping coefficients, iterative solvers often converge slowly. The use of a direct solver on the coarse grid is then ideal since it has a robust behaviour, and the underlying factorisation can be computed once and reused cheaply during each MG cycle, for problems where the matrix stays unchanged. We chose to use the sparse direct solver MUMPS, known for its good parallel behaviour, which offers the possibility to approximate the factorisation through the use of block-low rank approximation and single precision arithmetic.

- Secondly, the limited granularity of the coarse grid problem makes communication dominant compared to computation when solved with a high number of processes. We proposed 2 agglomeration techniques in order to reduce the execution by solving the coarse grid problem with fewer processes. The Superman agglomeration in particular adds a degree of parallelism as the factorisation of the direct solver is performed at the same time as the factorisation of the direct solver.

We studied the solution of a saddle-point problem, with up to 10^{11} unknowns using the multigrid *Hierarchical Hybrid Grids* framework (HHG) with up to 43 000 cores. With the combination of the approximate direct solver and the Superman agglomeration applied on the coarse grid, we demonstrated an overall improvement of the parallel efficiency by 9% point. This is an essential improvement for extreme scale simulation using multigrid.

The ABCD-Solver

We then explored a second axis with the study of hybrid solvers for unsymmetric square and rectangular systems, based on the row projection methods. We first introduced the principle of these methods which, based a partitioning of the matrix into blocks, construct an approximation of the solution iteratively through projections on these blocks. We then focused on the block Cimmino scheme, which has a good potential for parallelism thanks to the independence between the projections at each iteration. In the Chapter II, we introduced two methods based on the block Cimmino iterations with their extension to the minimum-norm solution of underdetermined systems and least-square problems, and a detailed analysis on their behaviour.

The block Cimmino iterations can be proven to be a type of domain decomposition method, namely an additive Schwartz method with minimal overlap, as they correspond to applying the block Jacobi iterations on the normal equations. The convergence of the block Cimmino iterations is however known to be slow, often with plateaus in the convergence due to clusters of small eigenvalues in the iteration matrix spectrum. In order to overcome these convergence issues, a *stabilised block-CG acceleration of the block Cimmino iterations* (BC) was studied with a partitioning of the matrix into blocks of rows. Through a good choice of the block size, the block-CG algorithm allows to reduce partly the plateaus in the convergence of the block Cimmino iterations, and exploit efficient matrix operations with BLAS3 kernels. The projections required at each iteration are performed using a direct solver applied on an augmented system built from the partition. This combination of a direct solver applied for the independent projections on subdomains inside a global iterative scheme is what makes this method hybrid.

Despite the block-CG acceleration, the convergence of this solver stays dependent on the angles between the subspaces spanned by the partitions and thus depends on the problem. An alternative was proposed in which the system is enlarged with additional variables and constraints to enforce the mutual orthogonality between partitions. Applied on this enlarged system, the block Cimmino iterations are guaranteed to converge in one iteration. We obtain a pseudo-direct approach called the Augmented Block Cimmino method (ABCD). This method requires the embarrassingly parallel construction and the solution of a relatively smaller, but denser, system S. We also showed a link to DDM, as the ABCD method decouples completely the subdomains with the additional variables, and the information from the interconnection between these subdomains is condensed in the matrix S, a Schur complement. The issue of this method is the solution of S which may require prohibitive amounts of memory when solved with a direct solver due to its size and density.

In the Chapter II, the main originality of the approach is the extension of the existing BC and ABCD methods, to the minimum-norm solution of underdetermined systems and to the solution of least-squares problems. The overdetermined case is based on a partitioning of the matrix into blocks of columns. The main points for these extensions are a block-CG acceleration in the case of a column partitioning, and the proof for rectangular problems that the solution of the augmented system in the ABCD method corresponds to the solution of the original system. In the underdetermined case, we showed in particular that the solution obtained with the ABCD method is the minimum-norm solution in addition to respecting $Ax = b$. The number of methods capable of solving rectangular systems is limited, and thus this is a positive contribution for scientific computing. The possibility to partition the matrix either in blocks of rows or in blocks of columns gives an additional choice to solve unsymmetric square systems, and this can greatly affect the behaviour of the BC and ABCD methods.

In the first part of Chapter III, we studied the impact of preprocessing techniques applied on the original matrix. Improving the scaling factors of the original matrix can have a great impact on the subsequent Schur complement in the case of square matrices. Also, we detail several choices for the partitioning technique, based on graph-partitioning method. After applying the BC and ABCD methods on a very large number of matrices, we show that the GRIP partitioner introduced in [126] is more likely to accelerate BC, while the partitioner PaToH generally gives the smallest augmentation size in the ABCD method. Then, we introduce a new augmentation technique in order to further reduce the size of the Schur complement in the case of highly interconnected partitions. On a

very large number of matrices (including rectangular ones), we show that the size of the Schur complement in the ABCD method stays in acceptable ranges most of the time, and thus ABCD is widely applicable.

In the second part of Chapter III, we introduce the parallel implementation of the iterative and the augmented method inside the ABCD-Solver. Executed with MPI processes and OpenMP threads, the solver uses a hybrid parallelisation scheme. Several levels of parallelism are managed: the independence between projections, the parallelisation of the direct solver to compute the projections, and the multithreaded underlying dense kernels. After partitioning the matrix, one or more partitions are given to a process, called master. At each iteration, the master computes the projections corresponding to its local partitions, and communication is performed between masters to obtain the global sum of these projections. The distribution of partitions in groups must balance the average workload, induced by the computation of projections, over all processes. For the BC method, we introduce a distribution of partitions which additionally decrease the amount of communications between masters. This distribution decreases the overall execution time for highly interconnected systems where communication is dominating. After the partitions have been distributed, processes with no partitions can be assigned to the masters to parallelise the computation of projections. We introduce a technique which spreads the masters explicitly on the computing architecture, while putting the workers from the same master close to each other. Through decreased, concurrent memory accesses from masters, the overall computation time is then decreased for both the BC and the ABCD method. In the case of the ABCD method, we also show that solving the Schur complement with too many processes can slow the computation. Indeed, the cases where the ABCD method is efficient are cases where the Schur complement is small, then the direct solver used to get its solution cannot scale. From these results, we decide to use only the masters to solve the Schur complement.

Using the introduced scaling technique, partitioning and augmentation technique, as well as the introduced methods to improve the execution time of our implementation the BC and ABCD methods, we applied our approaches to a very wide range of matrices from the SuiteSparse Matrix Collection. The BC and ABCD methods are then compared to the direct solvers MUMPS, for square matrices, and QR-MUMPS, for rectangular matrices. Both direct solvers are generally faster than our approach, with only 17% of the matrices where BC or ABCD is faster. However, the ABCD-Solver generally has the advantage in terms of memory consumption. From a detailed comparison with QR-MUMPS, we see that the difference in behaviour highly depends on the matrices.

Cases where the ABCD method will be best are cases where the Schur complement stays relatively small and sparse, e.g. dense rows or columns can give very large augmentations. As for the BC method, it stays a good choice whenever memory is decisive.

For the solution of discretized PDE problems, we finally propose in Chapter IV to apply the augmentation technique on a coarser level of grid. The goal was to open the principal angles between the subspaces spanned by the partitions augmented with this approach, compared to the angles between the original partitions. With such opened angles, the BC method applied on the augmented partition is shown to have a fast linear convergence even with a very coarse grid, on three types of 2D problems, namely an heterogeneous diffusion, an Helmholtz, and a convection-diffusion problems. Through a choice of a coarser level, the size of the augmentation decreases at the same rate as the number of iterations increases. And the contrary is observed with a choice of a finer grid. For the simultaneous construction of the closure equations, and corresponding right hand side, we then propose a block-CG algorithm based on the application of the BC method, with the partitions augmented using a coarse level, applied on a set of canonical vectors. Even though the convergence of the BC method for each separate vector is linear and fast, a few hundred must be computed and the obtained closure equations are dense. Using these approximated closure equations in a global BC method, we then obtain the final solution with a linear convergence depending on the angles between $\overline{A}$ and W, thus depending on the accuracy of W. The issue issue is that, due to the high density of W, computing the corresponding projection can be expensive. We then proposed possible research tracks in order to solve the latter problem.

Overall, this approach, called the C-ABCD method, allows a fast linear convergence of BC applied on the augmented partitions. Even though finding an efficient solution for the projection on the closure equations is still an open question, this approach is a positive evolution for the block Cimmino method. In [68], the authors proposed to combine the CG acceleration with Chebyshev filtering to target directly the small clusters of eigenvalues in the iteration matrix of block Cimmino, whose eigenvalues are clustered around 1, and small clusters of eigenvalues remain at the extreme of the spectrum. While the authors showed that this approach can be beneficial for the BC method, the number of small eigenvalues that are improved, and thus the acceleration of convergence, could not be explicitly controlled. Then, the ABCD method was introduced in [47], and while the convergence is obtained in 1 single iteration, the size of the augmentation is dependent on the problem and the partitioning. With the C-ABCD approach, we have a way to encourage fast linear convergence for the BC method with a number of added variables

that is explicitly controlled, and this is big step forward for the block Cimmino methods.

Perspectives and future work

The methods proposed in this thesis opened the way for several future research tracks. In particular the relaxed augmentation method for which a lot of possible evolutions were given in Chapter IV. The search of a good preconditioner for the Schur complement in C-ABCD is a long-term perspective. In the context of PDE problems, using the properties of a specific problem and discretisation method would allow to get a precise understanding of the relaxed augmentation in physical terms. Then, e.g. using algebraic methods [132], other specific properties could be introduced such as the approximation property [73]. Throughout this thesis, we have established the interpretation of the BC method as iterative domain decomposition methods, and the ABCD method as a Schur complement method. It is then fairly logical to assume that an interpretation of the C-ABCD approach is possible in terms of a 2-level domain decomposition method, or at least some links could be made with known methods such as BDDC or FETI-DP.

Bibliography

[1] Patrick Amestoy, Cleve Ashcraft, Olivier Boiteau, Alfredo Buttari, Jean-Yves L'Excellent, and Clément Weisbecker. Improving multifrontal methods by means of block low-rank representations. *SIAM J. Sci. Comput.*, 37(3):A1451–A1474, 2015.

[2] Patrick Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Theo Mary. On the complexity of the block low-rank multifrontal factorization. *SIAM Journal on Scientific Computing*, 39(4):A1710–A1740, 2017.

[3] Patrick R. Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Theo Mary. Bridging the gap between flat and hierarchical low-rank matrix formats: The multilevel block low-rank format. *SIAM J. Sci. Comput.*, 41(3):A1414–A1442, 2019.

[4] Patrick R. Amestoy, Alfredo Buttari, Jean-Yves L'Excellent, and Theo Mary. Performance and scalability of the block low-rank multifrontal factorization on multicore architectures. *ACM Trans. Math. Software*, 45(1):2:1–2:26, February 2019.

[5] Patrick R Amestoy, Timothy A Davis, and Iain S Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.

[6] Patrick R Amestoy, Iain S Duff, Jean-Yves L'Excellent, and Jacko Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[7] PR Amestoy. Factorization of large sparse matrices based on a multifrontal approach in a multiprocessor environment. *INPT PhD Thesis TH PA*, 91(2), 1991.

[8] Mario Arioli, Iain Duff, Joseph Noailles, and Daniel Ruiz. A block projection method for sparse matrices. *SIAM Journal on Scientific and Statistical Computing*, 13(1):47–70, 1992.

[9] Mario Arioli, Iain Duff, and Daniel Ruiz. Stopping criteria for iterative solvers. *SIAM Journal on Matrix Analysis and Applications*, 13(1):138–144, 1992.

[10] Mario Arioli, Iain S Duff, and Peter PM de Rijk. On the augmented system approach to sparse least-squares problems. *Numerische Mathematik*, 55(6):667–684, 1989.

[11] Mario Arioli, Iain S Duff, Daniel Ruiz, and Miloud Sadkane. Block lanczos techniques for accelerating the block cimmino method. *SIAM Journal on Scientific Computing*, 16(6):1478–1511, 1995.

[12] A. Baker, R.D. Falgout, H. Gahvari, T. Gamblin, W. Gropp, T. V. Kolev, K. E. Jordan, M. Schulz, and U. M. Yang. Preparing algebraic multigrid for exascale. Technical Report LLNL-TR-533076, Lawrence Livermore National Laboratory, 2012.

[13] Friedrich L Bauer. Optimally scaled matrices. *Numerische Mathematik*, 5(1):73–87, 1963.

[14] S. Bauer, M. Huber, S. Ghelichkhan, M. Mohr, U. Rüde, and B. Wohlmuth. Large-scale simulation of mantle convection based on a new matrix-free approach. *J. Comput. Sci.*, 31:60–76, 2019.

[15] S. Bauer, M. Mohr, U. Rüde, J. Weismüller, M. Wittmann, and B. Wohlmuth. A two-scale approach for efficient on-the-fly operator assembly in massively parallel high performance multigrid codes. *Appl. Numer. Math.*, 122:14–38, 2017.

[16] Simon Bauer, Hans-Peter Bunge, Daniel Drzisga, Siavash Ghelichkhan, Markus Huber, Nils Kohl, Marcus Mohr, Ulrich Rüde, Dominik Thönnes, and Barbara Wohlmuth. Terraneo—mantle convection beyond a trillion degrees of freedom. *Software for Exascale Computing SPPEXA*, page 569, 2020.

[17] Mario Bebendorf and Wolfgang Hackbusch. Existence of $\mathcal{H}$-matrix approximants to the inverse fe-matrix of elliptic operators with l_∞-coefficients. *Numerische Mathematik*, 95(1):1–28, 2003.

[18] B. Bergen and F. Hülsemann. Hierarchical hybrid grids: Data structures and core algorithms for multigrid. *Numer. Linear Algebra Appl.*, 11:279–291, 2004.

[19] Åke Björck. Iterative refinement of linear least squares solutions i. *BIT Numerical Mathematics*, 7(4):257–278, 1967.

[20] Åke Björck. Solving linear least squares problems by gram-schmidt orthogonalization. *BIT Numerical Mathematics*, 7(1):1–21, 1967.

[21] Åke Björck and Gene H Golub. Numerical methods for computing angles between linear subspaces. *Mathematics of computation*, 27(123):579–594, 1973.

[22] Randall Bramley and Ahmed Sameh. Row projection methods for large nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13(1):168–193, 1992.

[23] Achi Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation*, 31(138):333–390, 1977.

[24] Achi Brandt. Rigorous quantitative analysis of multigrid, i. constant coefficients two-level cycle with l_2-norm. *SIAM Journal on Numerical Analysis*, 31(6):1695–1730, 1994.

[25] Achi Brandt and Oren E Livne. *Multigrid techniques: 1984 guide with applications to fluid dynamics*, volume 67. SIAM, 2011.

[26] Franco Brezzi and Jim Douglas. Stabilized mixed methods for the Stokes problem. *Numer. Math.*, 53(1):225–235, 1988.

[27] William L Briggs, Van Emden Henson, and Steve F McCormick. *A multigrid tutorial*. SIAM, 2000.

[28] C. Burstedde, G. Stadler, L. Alisic, L. C. Wilcox, E. Tan, M. Gurnis, and O. Ghattas. Large-scale adaptive mantle convection simulation. *Geophys. J. Internat.*, 192(3):889–906, 2013.

[29] Alfredo Buttari. Fine-grained multithreading for the multifrontal qr factorization of sparse matrices. *SIAM Journal on Scientific Computing*, 35(4):C323–C345, 2013.

[30] Alfredo Buttari, Markus Huber, Philippe Leleux, Théo Mary, Ulrich Ruede, and Barbara Wohlmuth. Block Low Rank Single Precision Coarse Grid Solvers for Extreme Scale Multigrid Methods. working paper or preprint, April 2020.

[31] Alfredo Buttari and Bora Uçar. Parallel analysis and scaling. http://mumps.enseeiht.fr/doc/ud_2010/parana_talk.pdf, 2010.

[32] Umit V Catalyürek and Cevdet Aykanat. Patoh: a multilevel hypergraph partitioning tool, version 3.0. *Bilkent University, Department of Computer Engineering, Ankara*, 6533, 1999.

[33] Jaeyoung Choi, Jack Dongarra, Susan Ostrouchov, Antoine Petitet, David Walker, and R Clinton Whaley. A proposal for a set of parallel basic linear algebra subprograms. In *International Workshop on Applied Parallel Computing*, pages 107–114. Springer, 1995.

[34] Gianfranco Cimmino. *Estensione dell'identita di Picone alla piu generale equazione differenziale lineare ordinaria autoaggiunta: nota*. Bardi, 1929.

[35] Bernardo Cockburn, Jayadeep Gopalakrishnan, and Raytcho Lazarov. Unified hybridization of discontinuous galerkin, mixed, and continuous galerkin methods for second order elliptic problems. *SIAM Journal on Numerical Analysis*, 47(2):1319–1365, 2009.

[36] Elizabeth Cuthill and James McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, pages 157–172, 1969.

[37] Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55, 1998.

[38] D. Rhodri Davies, S. Goes, J.H. Davies, B.S.A. Schuberth, H.-P. Bunge, and J. Ritsema. Reconciling dynamic and seismic models of earth's lower mantle: The dominant role of thermal heterogeneity. *Earth and Planetary Science Letters*, 353-354:253 – 269, 2012.

[39] Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1–25, 2011.

[40] Victorita Dolean, Pierre Jolivet, and Frédéric Nataf. *An introduction to domain decomposition methods: algorithms, theory, and parallel implementation*, volume 144. SIAM, 2015.

[41] Jack Dongarra, Steven Gottlieb, and William TC Kramer. Race to exascale. *Computing in Science & Engineering*, 21(1):4–5, 2019.

[42] LA Drummond, Iain S Duff, Ronan Guivarch, Daniel Ruiz, and Mohamed Zenadi. Partitioning strategies for the block cimmino algorithm. *Journal of Engineering Mathematics*, 93(1):21–39, 2015.

[43] Maksymilian Dryja and Olof B Widlund. Some domain decomposition algorithms for elliptic problems. In *Iterative methods for large linear systems*, pages 273–291. Elsevier, 1990.

[44] D. Drzisga, L. John, U. Rüde, B. Wohlmuth, and W. Zulehner. On the analysis of block smoothers for saddle point problems. *SIAM J. Matrix Anal. Appl.*, 39(2):932–960, 2018.

[45] Iain Duff, Philippe Leleux, Daniel Ruiz, and F Sukru Torun. Improving the scalability of the abcd solver with a combination of new load balancing and communication minimization techniques. *Parallel Computing: Technology Trends*, 36:277–286, 2020.

[46] Iain S Duff, Albert Maurice Erisman, and John Ker Reid. *Direct methods for sparse matrices*. Oxford University Press, 2017.

[47] Iain S Duff, Ronan Guivarch, Daniel Ruiz, and Mohamed Zenadi. The augmented block cimmino distributed method. *SIAM Journal on Scientific Computing*, 37(3):A1248–A1269, 2015.

[48] Iain S Duff and Stéphane Pralet. Strategies for scaling and pivoting for sparse symmetric indefinite problems. *SIAM Journal on Matrix Analysis and Applications*, 27(2):313–340, 2005.

[49] Iain S Duff and John K Reid. The multifrontal solution of indefinite sparse symmetric linear. *ACM Transactions on Mathematical Software (TOMS)*, 9(3):302–325, 1983.

[50] A Dumitrasc, Ph Leleux, C Popa, D Ruiz, and U Ruede. On numerical solution of full rank linear systems. *arXiv preprint arXiv:1908.11746*, 2019.

[51] A Dumitrasc, Ph Leleux, Constantin Popa, D Ruiz, and Ulrich Ruede. On numerical solution of full rank linear systems. *SIAM Journal on Scientific Computing (SISC)*, Copper Mountain Special Section, 2020.

[52] Tommy Elfving. Block-iterative methods for consistent and inconsistent linear equations. *Numerische Mathematik*, 35(1):1–12, 1980.

[53] Howard C Elman, David J Silvester, and Andrew J Wathen. *Finite elements and fast iterative solvers: With applications in incompressible fluid dynamics*. Oxford University Press., Oxford, 2014.

[54] Bjorn Engquist and Hong-Kai Zhao. Absorbing boundary conditions for domain decomposition. *Applied numerical mathematics*, 27(4):341–365, 1998.

[55] Oliver G Ernst and Martin J Gander. Why it is difficult to solve helmholtz problems with classical iterative methods. In *Numerical analysis of multiscale problems*, pages 325–363. Springer, 2012.

[56] Robert D. Falgout and Jim E. Jones. Multigrid on massively parallel architectures. In Erik Dick, Kris Riemslagh, and Jan Vierendeels, editors, *Multigrid Methods VI*, pages 101–107. Springer Berlin Heidelberg, 2000.

[57] Radii Petrovich Fedorenko. A relaxation method for solving elliptic difference equations. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 1(5):922–927, 1961.

[58] Charles M Fiduccia and Robert M Mattheyses. A linear-time heuristic for improving network partitions. In *19th design automation conference*, pages 175–181. IEEE, 1982.

[59] George Elmer Forsythe and Cleve B Moler. *Computer solution of linear algebraic systems*. Prentice-Hall, 1967.

[60] Martin Jakob Gander. Schwarz methods over the course of time. *Electronic transactions on numerical analysis*, 31:228–255, 2008.

[61] Carl Friedrich Gauss. *Disquisitio de elementis ellipticis Palladis ex oppositionibus annorum 1803, 1804, 1805, 1807, 1808, 1809*. Königliche Gesellschaft der Wissenschaften, 1811.

[62] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973.

[63] Alan George. An automatic one-way dissection algorithm for irregular finite element problems. *SIAM Journal on Numerical Analysis*, 17(6):740–751, 1980.

[64] J Alan George. Computer implementation of the finite element method. Technical report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1971.

[65] B. Gmeiner, U. Rüde, H. Stengel, C. Waluga, and B. Wohlmuth. Towards textbook efficiency for parallel multigrid. *Numer. Math. Theory Methods Appl.*, 8(1):22–46, 2015.

[66] Björn Gmeiner, Markus Huber, Lorenz John, Ulrich Rüde, and Barbara Wohlmuth. A quantitative performance study for Stokes solvers at the extreme scale. *J. Comput. Sci.*, 17:509 – 521, 2016.

[67] Gene Golub and William Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224, 1965.

[68] Gene H Golub, Daniel Ruiz, and Ahmed Touhami. A hybrid approach combining chebyshev filter and conjugate gradient for solving linear systems with multiple right-hand sides. *SIAM journal on matrix analysis and applications*, 29(3):774–795, 2007.

[69] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU press, 2012.

[70] Joseph F Grcar. Matrix stretching for linear equations. *arXiv preprint arXiv:1203.2377*, 2012.

[71] William Gropp, William D Gropp, Ewing Lusk, Anthony Skjellum, and Argonne Distinguished Fellow Emeritus Ewing Lusk. *Using MPI: portable parallel programming with the message-passing interface*, volume 1. MIT press, 1999.

[72] Wolfgang Hackbusch. *Iterative solution of large sparse systems of equations*, volume 95. Springer, 1994.

[73] Wolfgang Hackbusch. *Multi-grid methods and applications*, volume 4. Springer Science & Business Media, 2013.

[74] Wolfgang Hackbusch. *Hierarchical matrices : algorithms and analysis*, volume 49 of *Springer series in computational mathematics*. Springer, Berlin, 2015.

[75] Louis A Hageman and David M Young. *Applied iterative methods*. Courier Corporation, 2012.

[76] Timo Heister, Juliane Dannberg, Rene Gassmöller, and Wolfgang Bangerth. High accuracy mantle convection simulation through modern numerical methods–ii: realistic models and problems. *Geophysical Journal International*, 210(2):833–851, 2017.

[77] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[78] Pascal Hénon, Pierre Ramet, and Jean Roman. Pastix: a high-performance parallel direct solver for sparse symmetric positive definite systems. *Parallel Computing*, 28(2):301–321, 2002.

[79] Magnus R Hestenes, Eduard Stiefel, et al. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952.

[80] Nicholas J Higham. Exploiting fast matrix multiplication within the level 3 blas. *ACM Transactions on Mathematical Software (TOMS)*, 16(4):352–368, 1990.

[81] Nicholas J Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002.

[82] Nicholas J. Higham and Theo Mary. Solving block low-rank linear systems by LU factorization is numerically stable. MIMS EPrint 2019.15, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, September 2019.

[83] T J R Hughes, L P Franca, and M Balestra. A new finite element formulation for computational fluid dynamics: V. circumventing the Babuška-Brezzi condition: A stable Petrov-Galerkin formulation of. *Comput. Methods Appl. Mech. Eng.*, 59(1):85–99, 1986.

[84] F. Hülsemann, M. Kowarschik, M. Mohr, and U. Rüde. Parallel Geometric Multigrid. In A. M. Bruaset and A. Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, number 51 in Lecture Notes in Computational Science and Engineering, pages 165–208. Springer, 2005.

[85] Frank Hülsemann, Benjamin Bergen, and Ulrich Rüde. Hierarchical hybrid grids as basis for parallel numerical solution of PDE. In *European Conference on Parallel Processing*, pages 840–843. Springer, 2003.

[86] Caroline Japhet. Optimized krylov-ventcell method. application to convection-diffusion problems. In *Proceedings of the 9th international conference on domain decomposition methods*, pages 382–389, 1998.

[87] Emmanuel Jeannot, Guillaume Mercier, and François Tessier. Process placement in multicore clusters: Algorithmic issues and practical techniques. *IEEE Transactions on Parallel and Distributed Systems*, 25(4):993–1002, 2013.

[88] S Karczmarz. Angenaherte auflosung von systemen linearer glei-chungen. *Bull. Int. Acad. Pol. Sic. Let., Cl. Sci. Math. Nat.*, pages 355–357, 1937.

[89] George Karypis. Metis, a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices version 4.0. *http://glaros. dtc. umn. edu/gkhome/metis/metis/download*, 1998.

[90] Axel Klawonn, Martin Kühn, and Oliver Rheinbach. Adaptive feti-dp and bddc methods with a generalized transformation of basis for heterogeneous problems. *Electronic Transactions on Numerical Analysis (ETNA)*, 49:1–27, 2018.

[91] Israel Kleiner et al. *A history of abstract algebra*. Springer Science & Business Media, 2007.

[92] Philip A Knight, Daniel Ruiz, and Bora Uçar. A symmetry preserving algorithm for matrix scaling. *SIAM journal on Matrix Analysis and Applications*, 35(3):931–955, 2014.

[93] Nils Kohl and Ulrich Rüde. Textbook efficiency: massively parallel matrix-free multigrid for the stokes system, 2020.

[94] Charles E Leiserson, Neil C Thompson, Joel S Emer, Bradley C Kuszmaul, Butler W Lampson, Daniel Sanchez, and Tao B Schardl. There's plenty of room at the top: What will drive computer performance after moore's law? *Science*, 368(6495), 2020.

[95] Thomas Lengauer. *Combinatorial algorithms for integrated circuit layout*. Springer Science & Business Media, 2012.

[96] Leroy Anthony Drummond Lewis. *Solution of general linear systems of equations using block Krylov based iterative methods on distributed computing environments.* PhD thesis, CERFACS, 1995.

[97] Xiaoye S Li. An overview of superlu: Algorithms, implementation, and user interface. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):302–325, 2005.

[98] Pierre-Louis Lions. On the schwarz alternating method. i. In *First international symposium on domain decomposition methods for partial differential equations*, volume 1, page 42. Paris, France, 1988.

[99] Pierre-Louis Lions. On the schwarz alternating method. iii: a variant for nonoverlapping subdomains. In *Third international symposium on domain decomposition methods for partial differential equations*, volume 6, pages 202–223. SIAM Philadelphia, PA, 1990.

[100] T. Mary. *Block Low-Rank multifrontal solvers: complexity, performance, and scalability.* PhD thesis, Université de Toulouse, 2017.

[101] Dave A. May, Patrick Sanan, Karl Rupp, Matthew G. Knepley, and Barry F. Smith. Extreme-scale multigrid components within PETSc. In *Proceedings of the Platform for Advanced Scientific Computing Conference, PASC 2016, Lausanne, Switzerland, June 8-10, 2016*, page 5, 2016.

[102] Gordon E Moore et al. Cramming more components onto integrated circuits, 1965.

[103] Gordon E Moore et al. Progress in digital integrated electronics. In *Electron devices meeting*, volume 21, pages 11–13, 1975.

[104] R. D. Müller, M. Sdrolias, C. Gaina, and W. R. Roest. Age, spreading rates, and spreading asymmetry of the world's ocean crust. *Geochem. Geophys. Geosyst.*, 9(4):1525–2027, 2008.

[105] Werner Oettli and William Prager. Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides. *Numerische Mathematik*, 6(1):405–409, 1964.

[106] Dianne P O'Leary. The block conjugate gradient algorithm and related methods. *Linear Algebra and its Applications*, 1980.

[107] François Pellegrini. Scotch and libscotch 5.1 user's guide. *HAL*, 2008.

[108] Constantin Popa. *Projection Algorithms-Classical Results and Developments: Applications to Image Reconstruction.* LAP, Lambert Academic Pub., 2012.

[109] A. Reisner, L. Olson, and J. Moulton. Scaling structured multigrid to 500k+ cores through coarse-grid redistribution. *SIAM Journal on Scientific Computing*, 40(4):C581–C604, 2018.

[110] S. Reiter, A. Vogel, I. Heppner, M. Rupp, and G. Wittum. A massively parallel geometric multigrid solver on hierarchically distributed grids. *Comput. Visual. Sci.*, 16(4):151–164, 2014.

[111] Ulrich Ruede, Karen Willcox, Lois Curfman McInnes, and Hans De Sterck. Research and education in computational science and engineering. *Siam Review*, 60(3):707–754, 2018.

[112] Daniel Ruiz. Solution of large sparse unsymmetric linear systems with a block iterative method in a multiprocessor environment. *CERFA CS TH/PA/9*, 6, 1992.

[113] Daniel Ruiz. A scaling algorithm to equilibrate both rows and columns norms in matrices. Technical report, CM-P00040415, 2001.

[114] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[115] Yousef Saad and Henk A Van Der Vorst. Iterative solution of linear systems in the 20th century. *Journal of Computational and Applied Mathematics*, 123(1-2):1–33, 2000.

[116] Olaf Schenk, Klaus Gärtner, Wolfgang Fichtner, and Andreas Stricker. Pardiso: a high-performance serial and parallel sparse linear solver in semiconductor device simulation. *Future Generation Computer Systems*, 18(1):69–78, 2001.

[117] Hermann Amandus Schwarz. *Ueber einen Grenzübergang durch alternirendes Verfahren*. Zürcher u. Furrer, 1870.

[118] Kangshen Shen, John N Crossley, Anthony Wah-Cheung Lun, and Hui Liu. *The nine chapters on the mathematical art: Companion and commentary*. Oxford University Press on Demand, 1999.

[119] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[120] N. A. Simmons, S. C. Myers, G. Johannesson, E. Matzel, and S. P. Grand. Evidence for long-lived subduction of an ancient tectonic plate beneath the southern Indian Ocean. *Geophys. Res. Lett.*, 42(21):9270–9278, 2015.

[121] David Skinner and William Kramer. Understanding the causes of performance variability in hpc workloads. In *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.*, pages 137–149. IEEE, 2005.

[122] Gilbert Strang and LB Freund. Introduction to applied mathematics, 1986.

[123] H. Sundar, G. Biros, C. Burstedde, J. Rudi, O. Ghattas, and G. Stadler. Parallel geometric-algebraic multigrid on unstructured forests of octrees. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–11, November 2012.

[124] Paul J Tackley. Effects of strongly variable viscosity on three-dimensional compressible convection in planetary mantles. *Journal of Geophysical Research: Solid Earth*, 101(B2):3311–3332, 1996.

[125] HS Tang, RD Haynes, and G Houzeaux. A review of domain decomposition methods for simulation of fluid flows: Concepts, algorithms, and applications. *Archives of Computational Methods in Engineering*, pages 1–33, 2020.

[126] F Sukru Torun, Murat Manguoglu, and Cevdet Aykanat. A novel partitioning method for accelerating the block cimmino algorithm. *SIAM Journal on Scientific Computing*, 40(6):C827–C850, 2018.

[127] Andrea Toselli and Olof Widlund. *Domain decomposition methods-algorithms and theory*, volume 34. Springer Science & Business Media, 2006.

[128] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

[129] Ulrich Trottenberg, Cornelius W Oosterlee, and Anton Schuller. *Multigrid*. Elsevier, 2000.

[130] James H Wilkinson. Error analysis of floating-point computation. *Numerische Mathematik*, 2(1):319–340, 1960.

[131] JH Wilkinson. Rounding errors in algebraic processes. information processing, 1960.

[132] Ulrike Meier Yang. Parallel algebraic multigrid methods—high performance preconditioners. In *Numerical solution of partial differential equations on parallel computers*, pages 209–236. Springer, 2006.

[133] Mohamed Zenadi. *The solution of large sparse linear systems on parallel computers using a hybrid implementation of the block Cimmino method*. PhD thesis, EDMITT, 2013.

[134] Walter Zulehner. Analysis of iterative methods for saddle point problems: A unified approach. *Math. Comput.*, 71(238):479–505, April 2002.

Contributions

Journal papers in review

- Alfredo Buttari, Markus Huber, Philippe Leleux, Théo Mary, Ulrich Ruede, and Barbara Wohlmuth. Block low rank single precision coarse grid solvers for extreme scale multigrid methods. *Wiley's Numerical Linear Algebra with Applications*, 2020

Journal papers accepted

- A Dumitrasc, Ph Leleux, Constantin Popa, D Ruiz, and Ulrich Ruede. On numerical solution of full rank linear systems. *SIAM Journal on Scientific Computing (SISC)*, Copper Mountain Special Section, 2020

Journal publications

- Johan Quilbé, Léo Lamy, Laurent Brottier, Philippe Leleux, Joel Fardoux, Ronan Rivallan, Thomas Benichou, Rémi Guyonnet, Manuel Becana, Irene Villar, et al. Genetics of nodulation in aeschynomene evenia uncovers mechanisms of the rhizobium–legume symbiosis. *Nature Communications*, 12(1):1–14, 2021
- Clémence Chaintreuil, Ronan Rivallan, David J Bertioli, Christophe Klopp, Jerome Gouzy, Brigitte Courtois, Philippe Leleux, Guillaume Martin, Jean-Francois Rami, Djamel Gully, et al. A gene-based map of the nod factor-independent aeschynomene evenia sheds new light on the evolution of nodulation and legume genomes. *DNA Research*, 23(4):365–376, 2016
- Saliha Hammoumi, Tatiana Vallaeys, Ayi Santika, Philippe Leleux, Ewa Borzym, Christophe Klopp, and Jean-Christophe Avarre. Targeted genomic enrichment and sequencing of cyhv-3 from carp tissues confirms low nucleotide diversity and mixed genotype infections. *PeerJ*, 4:e2516, 2016

Conference papers

- Philippe Leleux, Ulrich Rüde, and Daniel Ruiz. A multigrid-inspired approach for the augmented block cimmino distributed solver. In *20th Copper Mountain Conference On Multigrid Methods*, SIAM CM 2021, March 2021
- Markus Huber, Nils Kohl, Philippe Leleux, Ulrich Rüde, Dominik Thönnes, and Barbara Wohlmuth. Massively Parallel Multigrid with Direct Coarse Grid Solvers. In *NIC Symposium 2020*, volume 50 of *Publication Series of the John von Neumann Institute for Computing (NIC) NIC Series*, pages 335 – 344, Jülich, Feb 2020
- Iain S. Duff, Philippe Leleux, Daniel Ruiz, and F. Sukru Torun. Improving the scalability of the ABCD solver with a combination of new load balancing and communication minimization techniques. In *Parallel Computing: Technology Trends, Proceedings of the International Conference on Parallel Computing, PARCO 2019, Prague, Czech Republic, September 10-13, 2019*, volume 36 of *Advances in Parallel Computing*, pages 277–286. IOS Press, 2019
- Andrei Dumitrasc, Philippe Leleux, and Ulrich Rüde. Block partitioning of sparse rectangular matrices. *PAMM*, 19(1):e201900287, 2019

- Nicolas Briot, Annie Chateau, Remi Coletta, Simon De Givry, Philippe Leleux, and Thomas Schiex. An integer linear programming approach for genome scaffolding. In *WCB: Workshop on Constraint-Based Methods for Bioinformatics*, 10th Workshop on Constraint-Based Methods for Bioinformatics (WCB), 2014, page 16 p., Lyon, France, September 2014

International conferences

- Alfredo Buttari, Markus Huber, Philippe Leleux, Théo Mary, Ulrich Ruede, and Barbara Wohlmuth. Massively parallel multigrid with direct coarse grid solvers. In *The Platform for Advanced Scientific Computing 2021*, PASC 2021, July 2021
- U Ruede, D Thoennes, N Kohl, and P Leleux. Scalability and effficiency of parallel multigrid for the stokes system. In *SIAM Conference on Computational Science and Engineering, Online*, SIAM CSE 2021, March 2021
- P Leleux, U Ruede, and D Ruiz. Multigrid-based augmented block-cimmino method. In *Sparse Days*, CERFACS, July 2019
- P Leleux, U Ruede, and D Ruiz. A multigrid-based approach for the augmented block cimmino distributed solver. In *International Congress on Industrial and Applied Mathematics, Valencia*, ICIAM 2019, July 2019
- P Leleux, U Ruede, and D Ruiz. Multigrid-based augmented block-cimmino method. In *28th Biennial Conference on Numerical Analysis*, Strathclyde University, Glasgow, June 2019
- Andrei Dumitrasc, Philippe Leleux, Constantin Popa, Ulrich Ruede, Daniel Ruiz, and Fahreddin Sükrü Torun. Recent advances on the Augmented Block Cimmino method. In *12th Workshop on Mathematical Modelling of Environmental and Life Sciences Problems - MMELSP 2018*, Constanta, Romania, October 2018. Universitatea Ovidius din Constanta, Romania and Romanian Academy, Bucharest, Romania. (Conférencier invité)

Research reports

- I. Khalfaoui-Hassani, Ph. Leleux, and D. Ruiz. Internship report: The column block cimmino method. Technical report, CERFACS, April 2020
- A Dumitrasc, Ph Leleux, Constantin Popa, D Ruiz, and Ulrich Ruede. On numerical solution of full rank linear systems. *arXiv preprint arXiv:1908.11746*, 2019
- A Dumitraşc, Ph Leleux, C Popa, D Ruiz, and S Torun. The augmented block cimmino algorithm revisited. *arXiv preprint arXiv:1805.11487*, 2018

Seminars

- P Leleux, U Ruede, and D Ruiz. Hybrid solvers based on the block cimmino method for the solution of full rank rectangular systems. In *Numerical analysis e-seminar*, Numerical Analysis group, University of Strathclyde, February 2021
- P Leleux, I Duff, and D Ruiz. Variants of block-cimmino and multigrid. In *Research day seminars*, LSS, FAU, November 2018
- P Leleux, I Duff, and D Ruiz. Hybrid direct and iterative solvers for sparse indefinite and overdetermined systems on future exascale architectures. In *Parallel algorithm team seminars*, CERFACS, October 2018

- P Leleux, I Duff, and D Ruiz. Variants of block-cimmino and multigrid. In *Research day seminars*, LSS, FAU, April 2018
- P Leleux, I Duff, and D Ruiz. Efficient solvers in the energy-oriented-center for excellence (eocoe). In *Research day seminars*, LSS, FAU, May 2017

Appendix A

On numerical solution of full rank linear systems

A.1 Block Cimmino as a Domain Decomposition Method

Let's first remind the definition of the damped block-Jacobi algorithm. We consider the system

$$Ky = b.$$

The matrix $K : m \times m$ here corresponds to the normal equations $K = AA^T$. This matrix is partitioned in blocks of size corresponding to the partitions of A, i.e. the row indices $\mathcal{M} = \{1, \ldots, m\}$ are partitioned in disjoint sets of indices $\mathcal{M}_i \subset \mathcal{M}$, $i = 1, \ldots, p$. For example if we consider 2 partitions, the system has the following block form

$$\begin{pmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

where $K_{ij} = A_i A_j^T$.

We define D as the block-diagonal of K with diagonal blocks $D_i = K_{ii} = A_i A_i^T$, the block-Jacobi algorithm then consists of the iteration

$$y_{Jacobi}^{(k+1)} = y^{(k+1)} = y^{(k)} + D^{-1}(b - Ky^{(k)}). \tag{A.1}$$

And the damped block-Jacobi iteration is

$$\begin{aligned} y^{(k+1)} &= (1-\omega)y^{(k)} + \omega y_{Jacobi}^{(k+1)} \\ &= (1-\omega)y^{(k)} + \omega(y^{(k)} + D^{-1}(b - Ky^{(k)}) \\ &= y^{(k)} + \omega D^{-1} r^{(k)} \text{ with } r^{(k)} = b - Ky^{(k)}. \end{aligned}$$

Considering the iterates $x^{(k)} = A^T y^{(k)}$, we recognise an iteration of the BC algorithm.

To get a more compact form of this iteration, we introduce the restriction operators $R_i : m_i \times m$ from $\mathcal{M}$ to $\mathcal{M}_i$

$$\begin{pmatrix} 0 & \dots & 0 & I_{m_i} & 0 & \dots & 0 \end{pmatrix},$$

as well as their transpose which are extension operators from $\mathcal{M}_i$ to $\mathcal{M}$ with $i \in \{1, \dots, p\}$. We then have the equalities

$$K_{ii} = R_i K R_i^T \text{ and } R_i^T K_{ii}^{-1} R_i = \begin{pmatrix} 0 & & & & \\ & \ddots & & & \\ & & K_{ii}^{-1} & & \\ & & & \ddots & \\ & & & & 0 \end{pmatrix}.$$

Using these definitions, we can rewrite (A.1) as

$$\begin{aligned} y^{(k+1)} &= y^{(k)} + \omega D^{-1} r^{(k)} \\ &= y^{(k)} + \omega \begin{pmatrix} K_{11}^{-1} & & \\ & \ddots & \\ & & K_{pp}^{-1} \end{pmatrix} r^{(k)} \\ &= y^{(k)} + \omega (\sum_{i=1}^{p} R_i^T K_{ii}^{-1} R_i) r^{(k)} \\ &= y^{(k)} + \omega (\sum_{i=1}^{p} R_i^T (R_i K R_i^T)^{-1} R_i) r^{(k)} \\ &= y^{(k)} + \omega M_{ASM}^{-1} r^{(k)}. \end{aligned}$$

We recognise in M_{ASM}^{-1} the abstract form of a damped additive Schwartz method iteration matrix, applied on the normal equations system, and to which the block-Jacobi method is equivalent. See section 1.2 in [40] for a more detailed proof of this equivalence. This interpretation gives a wider interpretation on the solution of the system with the block Cimmino iterations. Also, this naturally points to the solution of discretised PDEs, together with domain decomposition. As a matter of fact, the partitioning of the matrix can be performed using the geometry of the PDE problem, with the usual purpose of DDM to balance the workload inside partitions – subdomains – while minimising the links between them – interfaces.

A.2 Impact of the block size for the accelerated block Cimmino

Matrix	Type	1	2	4	8	16	32	64	128	256	512
deltaX	LS	265	233	183	143	107	76	47	27	18	12
Hardesty2	LS	727	701	699	629	645	341	186	107	65	43
image_interp	LS	635	633	610	506	300	163	93	55	35	22
LargeRegFile	LS	44	35	33	30	29	26	23	22	22	
Delor338K	under	148	88	56	44	37	34	32	30	24	16
neos	under	1092	971	818	588	424	269	181	130		
sctap1-2r	under	502	297	145	56	20	10	6	3	4	
stat96v3	under	126	101	79	60	46	36	28	21	16	11

Matrix	Type	1	2	4	8	16	32	64	128	256	512
deltaX	LS	72	65	58	57	54	60	75	89	129	191
Hardesty2	LS	2 850	2 840	3 280	3 530	4 800	3 850	4 280	5 140	6 560	9 910
image_interp	LS	496	533	594	564	457	409	462	574	811	1 160
LargeRegFile	LS	597	463	457	481	659	881	1 560	2 980	6 520	–
Delor338K	under	122	82	63	63	76	119	229	442	791	1 270
neos	under	2 020	2 200	1 970	1 660	1 670	1 590	2 200	3 250	–	–
sctap1-2r	under	99	67	37	18	8	7	8	12	61	–
stat96v3	under	455	383	340	330	347	418	663	1 030	1 690	2 740

Table A.1 Application of the block Cimmino accelerated method on rectangular matrices with an increasing block size for the block-CG algorithm in power of 2 from 1 to 256. The first table displays the number of iterations for convergence and the second the total time in seconds for the block-CG iterations.

A.3 Handling unsymmetric square matrices with the ABCD-Solver

The choice of row or column partitioning in the case of unsymmetric square matrices is completely problem dependent. Here, we apply the iterative and augmented methods on all square unsymmetric matrices from the SuiteSparse Matrix Collection with more than 1 000 rows and columns. After removing the matrices for which the ABCD-Solver could not finish, either because the system was ultimately rank deficient or because of too high memory requirements, around 700 matrices were kept. Additionally, matrices from a same family were merged, e.g. adder_dcop_01 to adder_dcop_48, and the mean value taken for iterations and augmentation sizes. In the end, 158 and 238 classes of problems are considered respectively for BC and ABCD. The goal of this study is to get a qualitative evaluation of how much does the choice on partitioning orientation favours BC or ABCD. The partitioner used is GRIP [126] for BC, and PaToH [32] for ABCD, see Section III.1.2 for more details. We use for BC a block size of 4, and for ABCD the technique from [47] giving smallest augmentation for each matrix.

In Figure A.1, the *x-axis* gives the relative difference in number of iterations, and the *y-axis* gives the relative difference in augmentation size between column and row partitioning. A positive value means that the row partitioning gave a smaller value for one or the other measure. The first observation is that some matrices only converged using a column partitioning, thus the cluster on the right of the figure. Additionally, the points are slightly shifted towards the bottom, giving a slight advantage for the row partitioning when using ABCD. Statistically, these results are confirmed when counting the case where one or the other partitioning is best for each approach. This would tend to say that a row partitioning is better for the iterative BC method and a column partitioning is better for the pseudo-direct ABCD method.

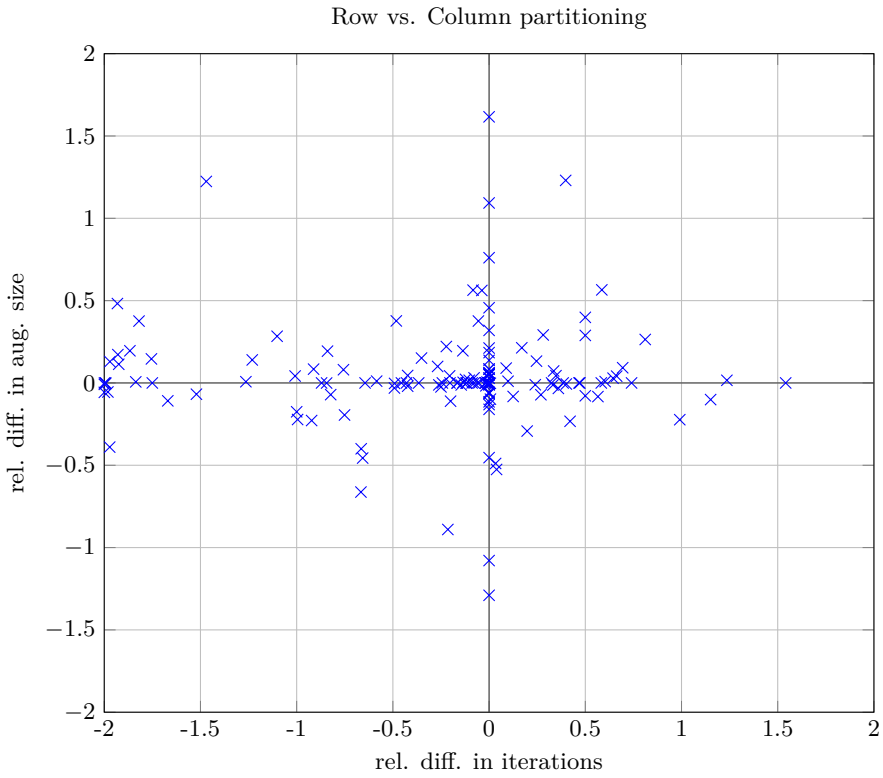


Figure A.1 Relative difference between row and column partitioning for *(x-axis)* the number of iterations, and *(y-axis)* the size of the augmentation. These results are obtained for 158 and 238 classes of problems, resp. for BC and ABCD, from matrices extracted from the SuiteSparse Matrix Collection.

Appendix B

Parallel implementation

B.1 Preprocessing

B.1.1 PaToH vs GRIP partitioner

The choice of the partitioning between GRIP and PaToH is also problem dependent. Here, we apply the iterative and augmented methods on all matrices (square or rectangular) from the SuiteSparse Matrix Collection with more than 1 000 rows and columns. After removing the matrices for which the ABCD-Solver could not finish, either because the system was ultimately rank deficient or because of too high memory requirements, around 1000 matrices were kept. Additionally, matrices from a same family were merged, e.g. adder_dcop_01 to adder_dcop_48, and the mean value taken for iterations and augmentation sizes. In the end, 361 and 329 classes of problems are considered respectively for BC and ABCD. The goal of this study is to get a qualitative evaluation of how much does the choice on partitioning GRIP among and PaToH favours BC or ABCD. We use for BC a block size of 4, and for ABCD the technique from [47] giving smallest augmentation for each matrix.

Figures B.1 shows the results for both BC and ABCD applied on each class problems. In x-axis, the difference in number of iterations for BC between the use of GRIP and the use of PaToH, scaled by the mean value. In y-axis, the difference in augmentation size for ABCD between the use of GRIP and the use of PaToH, scaled by the mean value. A positive value shows a better result from the use of PaToH, i.e. a smaller number of iteration, or augmentation size.

We observe that most green en red points are placed in the upper left part of the figure. This means that the GRIP partitioner has the advantage in terms of iterations and PaToH in terms of augmentation size. Only the underdetermined case is less clear,

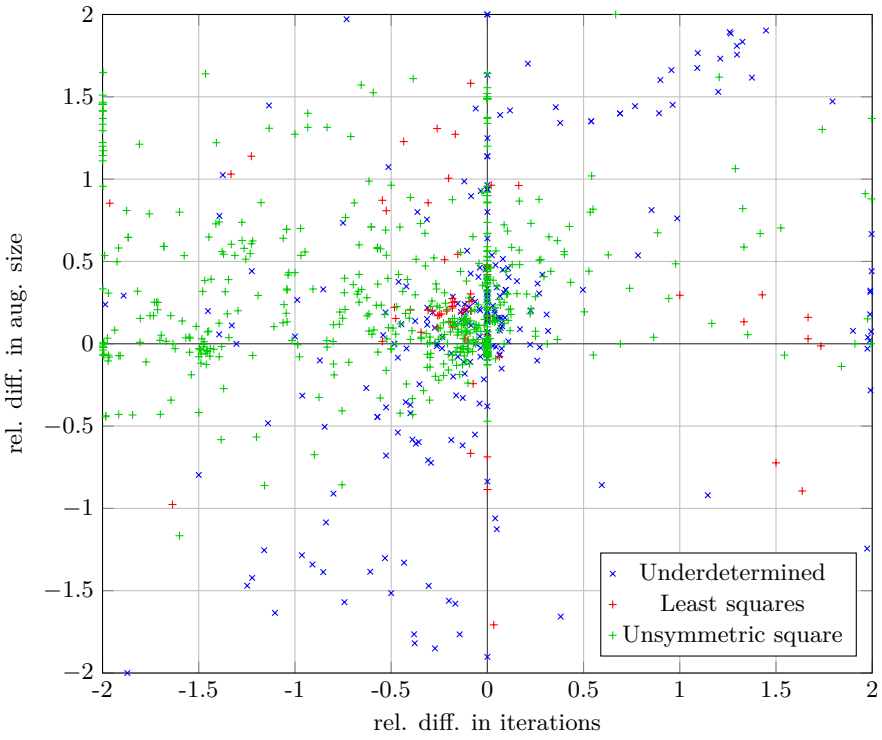


Figure B.1 Relative difference between the GRIP and the PaToH partitioners in terms of (x-axis) iterations and (y-axis) total time in seconds for the block Cimmino iterative method applied on 325 classes of problems from the SuiteSparse Matrix Collection. Positive values show a smaller number of iterations (resp. smaller execution time) when using the PaToH partitioner.

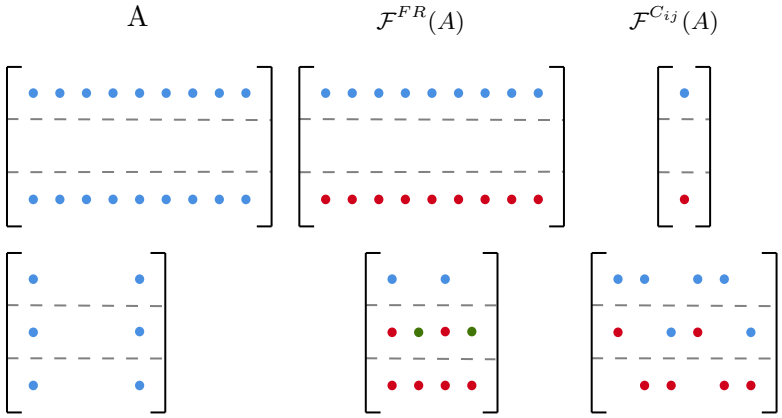


Figure B.2 Two matrices with respectively dense rows and dense columns, and partitioned in 3 blocks of rows. The corresponding augmentation blocks obtained with $\mathcal{F}^{FR}$ and $\mathcal{F}^{C_{ij}}$ are displayed.

but looking at the actual results the conclusion is still the same.

B.1.2 Matrix structure and augmentations $\mathcal{F}^{C_{ij}}$ and $\mathcal{F}^{FR}$

The choice of one or the other technique is entirely dependent on the structure of the matrix. We distinguish the 2 extreme cases of a matrix with dense rows and a matrix with dense columns. Both cases are illustrated in Figure B.2 with the corresponding augmentations obtained with both methods. In the presence of dense rows

1. the size of the interconnection blocks is large,
2. the normal equations $A_i A_j^T$ will not fill-in,
3. the number of interconnected partitions should be small as such rows will tend to be grouped.

Due to all these reasons, after filtering zero-rows/columns from the normal equations, the size of the augmentation given by $\mathcal{F}^{C_{ij}}$ stays small compared to $\mathcal{F}^{FR}(A)$. On the contrary, with dense columns

1. the size of the interconnection blocks is small,
2. the normal equations $A_i A_j^T$ will completely fill-in,
3. the number of interconnected partitions is large.

In this case, the lower number of duplications with respect to the number of interconnected partitions should make the augmentation from $\mathcal{F}^{FR}(A)$ smaller. Of course, these cases are caricature. The reality stands in the middle and both approaches can give similar results. Our goal here was to extract a rule of thumb based on structures that can be typical of a type of problem, either square, underdetermined or overdetermined.

We apply the two augmentation techniques on the test matrices bayer01, deltaX and sctap1-2r and give the results in Table B.2, together with the sparse structure of the 3 matrices in Figure B.3. sctap1-2r is the typical matrix displaying a dense column where $\mathcal{F}^{C_{ij}}(A)$ is large and dense. As for deltaX, with a mixture of dense rows and columns, and bayer01, with diagonal sparse features, both approaches naturally give similar results. The choice of augmentation is then guided on the one hand by the resulting conditioning of S, where $\mathcal{F}^{C_{ij}}$ tends to be worse as it relies on normal equations, and on the execution time of the solver which is heavily dependent on the size and density of S. In conclusion the augmentation method applied is problem dependent.

Matrix	**m** ($\times 10^6$)	**n** ($\times 10^6$)	**elts per row**	**#Parts**	**Problem**
`deltaX`	0.07	0.02	3.61	8	Counter Example
`sctap1-2r`	0.03	0.06	6.46	8	Linear Programming
`bayer01`	0.06	0.06	4.76	8	Chemical Process

Table B.1 Characteristics of the test matrices. n: the order of the matrix, nnz: the number of nonzero values in the matrix.

Matrix	**Aug.**	**S**			**Total (s)**
		m	*elts per row*	$\kappa(S)$	
`sctap1-2r`	C_{ij}	30339	7585.75	$1.08 \cdot 10^9$	248.30
	FR	238	461.50	$3.36 \cdot 10^6$	2.59
`deltaX`	C_{ij}	10389	2642.08	$2.38 \cdot 10^{15}$	30.25
	FR	10703	2701.5	$8.82 \cdot 10^{15}$	30.54
`bayer01`	C_{ij}	327	92.66	$1.47 \cdot 10^8$	2.83
	FR	257	69.33	$1.88 \cdot 10^7$	2.54

Table B.2 Application of ABCD with the augmentation methods $\mathcal{F}^{C_{ij}}$ and $\mathcal{F}^{FR}$. The size and number of elements per row of the matrix S are given as well as its condition number, and the sequential execution time to compute the solution.

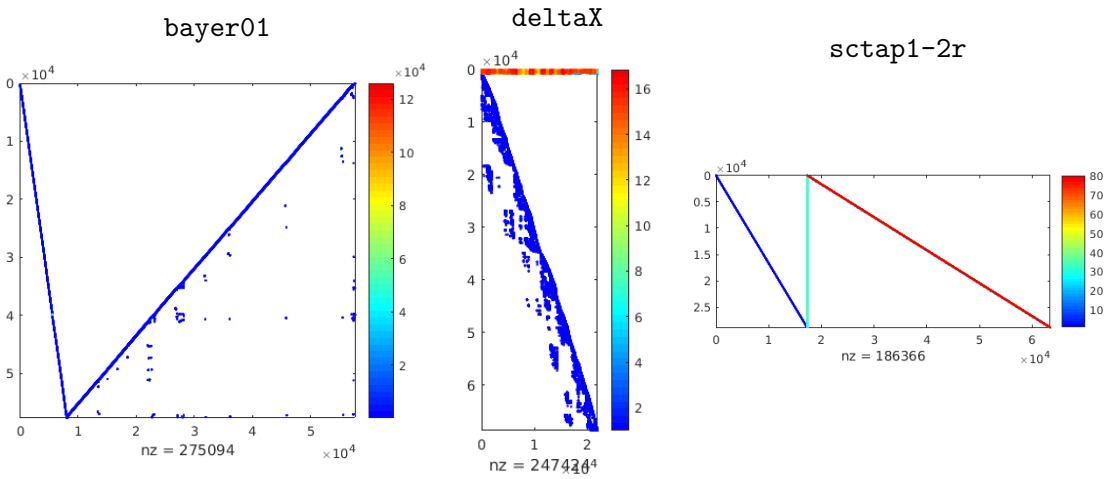


Figure B.3 Sparse structure of the test matrices.

B.2 Architecture aware placement of masters and workers

Here, we give the architecture aware algorithms for the placement of masters, Algo. 12, and the placement of workers, Algo. 13.

Algorithm 12 Placement of masters

Input: $nb\_masters$
Input: *map* containing for each node the rank of its processes. The nodes are numbered in descending order of number of processes
Output: *ptype*: the type of each process (0 for master, 1 for worker)

1: $master = 1$, $node = 1$, $direction = 1$
2: $ptype(proc) = 1$, $\forall proc$
3: **while** $master \leq nb\_masters$ **do**
4: **if** $\exists proc \in map(node)$ not assigned **then**
5: $ptype(proc) = 0$
6: $master++$
7: **else**
8: **break**
9: **end if**
10: **if** First node **then**
11: $direction++$
12: **else if** Last node **then**
13: $direction--$
14: **end if**
15: $node = node + direction$
16: **end while**

Algorithm 13 Placement of workers

Input: nb_procs, $nb\_masters$
Input: map: containing for each node the rank of its unassigned processes
Input: $numworkers$: number of workers required by each master. The masters are ordered in descending order of number of workers.
Input: M_node: the node of each master
Output: $workers$: the workers of each master

1: $workers[master] = \{\}$, $\forall master$
2: $nonFullNode = 0$
3: **for** $master = 1..nb\_masters$ **do**
4: $node = M\_node[master]$
5: **while** $numworkers[master] > 0$ **do**
6: **if** $node$ is fully assigned **then**
7: Find next non-empty $node$ fully assigned
8: **end if**
9: assigned $proc \in map[node]$ to $workers[master]$
10: $numworkers[master] - -$
11: **end while**
12: **end for**

Appendix C

Relaxed augmented block Cimmino method inspired from multigrid

Multigrid-inspired relaxed ABCD]Solution of full rank systems

C.1 Overdetermined augmentation as a multigrid scheme

In this section, we are augmenting the original system $Ax = b$ with additional variables corresponding to a restriction of the full rank operator A. We can solve the overdetermined augmented system

$$\begin{pmatrix} \overline{A} \\ W \end{pmatrix} x = \begin{pmatrix} A \\ P^T A \end{pmatrix} x = \begin{pmatrix} b \\ P^T b \end{pmatrix} \tag{C.1}$$

using the row partitioned block Cimmino since A is full rank and the system stays consistent.

We remind that implictely, the block Cimmino method get the solution in the two steps

$$AA^T y = b, \tag{C.2}$$

$$x = A^T y, \tag{C.3}$$

with y an intermediary variable, solution of the normal equations. We can build the simple 2-grid cycle in Algo. 14, with no post-smoothing, to solve the normal equations (C.2). In this algorithm, P and R are the prolongation an restriction operators. Also, $Smooth$ is the application of a smoother which we will be specified below, and $A_c = RAA^T P$ is the Galerkin operator.

Additionally, from (C.3), we can rewrite Algo. 14 w.r.t. $x^{(k)}$ by following the steps

Algorithm 14 2MG: 2-levels Multigrid algorithm where "Smooth" is the application of any smoother.

Input: initial guess $y^{(0)}$, number of cycles μ
Output: solution y
1: **for** k in 1..μ **do**
2: $y^{(k+1)} = Smooth(AA^T, y^{(k)}, b)$
3: $r^{(k+1)} = R(b - AA^T y^{(k)})$
4: $y^{(k+1)} = y^{(k)} + PA_c^{-1} r^{(k+1)}$
5: **end for**

- use the restriction operator $R = P^T$,
- smooth on $x^{(k)}$ instead of $y^{(k)}$, since we often can consider that $\Big(Smooth(AA^T, y^{(k)}, b) \iff Smooth($ e.g. for the block Kaczmarz smoother,
- Keep the coarse grid correction using the normal equations, as done for $y^{(k+1)}$, and get the next iterate with $x^{(k+1)} = A^T y^{(k+1)}$. We obtain

$$\begin{aligned} x^{(k+1)} &= A^T y^{(k+1)} \\ &= A^T y^{(k)} + A^T P A_c^{-1} P^T (b - AA^T y^{(k)} \\ &= x^{(k)} + A^T P (P^T AA^T P)^{-1} P^T (b - Ax^{(k)}) \\ &= x + (P^T A)^+ P^T b + (P^T A)^+ P^T A x^{(k)} \\ &= x + (P^T A)^+ P^T b + \mathcal{P}_{\mathcal{R}(P^T A)} x^{(k)} \end{aligned} \tag{C.4}$$

Considering that we use the block Kaczmarz iterations as smoothers on $x^{(k)}$ with a partitioning of the matrix A in blocks of rows A_i, we obtain the Algo. 15.

Algorithm 15 MG-Kaczmarz: Kaczmarz equivalent of the 2-levels Multigrid algorithm without post-smoothing.

Input: initial guess $x^{(0)}$, number of iterations μ
Output: solution x
1: **for** k in 1..μ **do**
2: $\delta^{(0)} = x^{(k)}$
3: **for** i in 1..p **do**
4: $\delta^{(i)} = \delta^{(i-1)} + A_i^+ (b_i - A_i \delta^{(i-1)})$
5: **end for**
6: $x^{(k+1)} = \delta^{(p)} + (P^T A)^+ (P^T b - P^T A \delta^{(p)})$
7: $x = \delta$
8: **end for**

Now, as explained in [27] (chapter 5), we can see the multigrid schemes as successive

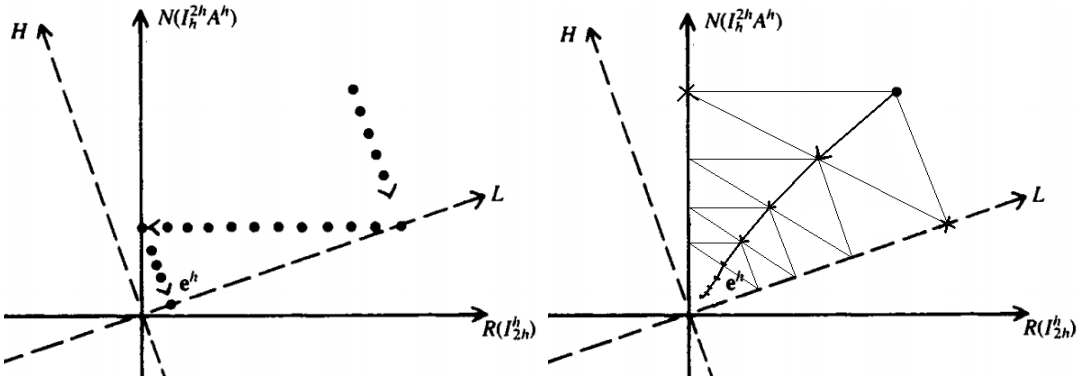


Figure C.1 Illustration of *(Left)* classical and *(Right)* additive Multigrid V-cycle iterations in terms of successive projections on non-orthogonal subspaces: ($\mathcal{L}$,$\mathcal{H}$) corresponds to the low/high-frequency modes subspaces and ($\mathcal{N}$,$\mathcal{R}$) corresponds to $\mathcal{N}(RA)$, with $R = P^T A$, and $\mathcal{R}(P)$. *Source:* [27].

applications of projectors. Consider the decomposition of the space

1. into the orthogonal subspaces $\mathcal{H}$ and $\mathcal{L}$ where $\mathcal{H}$ is the subspace corresponding to the high-frequency components of the error and $\mathcal{L}$ the subspace corresponding to the low frequency components,
2. into the orthogonal subspaces $\mathcal{N}(RA) = \mathcal{N}(P^T A)$ and $\mathcal{R}(P)$, noted $\mathcal{N}$ and $\mathcal{R}$.

$\mathcal{L}$ and $\mathcal{R}$ should be more nearly aligned as the range of the prolongation ideally corresponds to the low-frequency modes which we target on the coarse grid. Then

1. the smoother which quickly damps the high-frequency modes of the error can be viewed as an approximate projection of the subspace $\mathcal{L}$ (once the high-frequency part is removed, the point is near stationary),
2. the coarse grid correction, $I - PA_c^{-1}P^T A$, is a projection on the subspace $\mathcal{N}(RA) = \mathcal{R}(P)^\perp$, the orthogonal complement of $\mathcal{R}(P)$.

Applying a product of projections in an iterative way to converge to the solution of a linear system has a name: the block Kaczmarz iterations. This interpretation of the algorithm is illustrated in the left Figure C.1.

In Algo. 15, we can interpret the coarse grid correction as the presence of an additional partition for Kaczmarz, i.e. the partition $P^T A$. Finally, while the block Kaczmarz apply a product of projections, we can instead take the sum of projections, still including the additional partition $P^T A$. We then obtain block Cimmino iterations applied on the overdetermined system (C.1). This iterative scheme is finally equivalent to an additive 2-level cycle, see right Figure C.1.

C.2 C-ABCD parallelism with the construction of W

Here we present a possible parallelisation inside the ABCD-Solver to integrate the C-ABCD approach with an explicit construction of the closure equations W. The C-ABCD method combines the augmentation from ABCD, applied on the smaller matrix AP partitioned with the partitioning as in as A, and the conjugate gradient from BC, and with the addition of a step following the factorisation of the projection systems from the augmented partitions $\bar{A}_i$. This new step is the approximation of the block W and the right hand side f thanks to the variant of the block-CG algorithm introduced in Algo. 11. The Analysis and Factorisation phases of the direct solver MUMPS are then applied on the projection system built from the block W in order to integrate it as a partition in the global BC iterations.

An MPI process, called the master of W, needs to be reserved for the application of the direct solver MUMPS on W. Additionally, if there are processes left as workers, some can be assigned to the master of W for the application of MUMPS on the system associated to the projection onto the range of ^{T}. The process distributing the workers to the masters takes into account the estimation of workload from the Analysis phase of MUMPS, applied on the projection systems from the partitions $\bar{A}_i$. As this process is performed before we start building W, we use as workload estimation for the projection system based on W the maximum workload associated to a partition multiplied by a relaxation parameter, e.g. $\rho = 1.5$. Figure C.2 presents the parallelisation scheme of the ABCD-Solver for the application of the C-ABCD method.

The issue with this scheme is that the analysis, and factorisation of W are performed only using the master reserved for W and its workers. We are thus adding, between the factorisation of the projection systems from $\bar{A}_i$ and the conjugate gradient iterations, a step where most processes stay idle, thus breaking the potential performance.

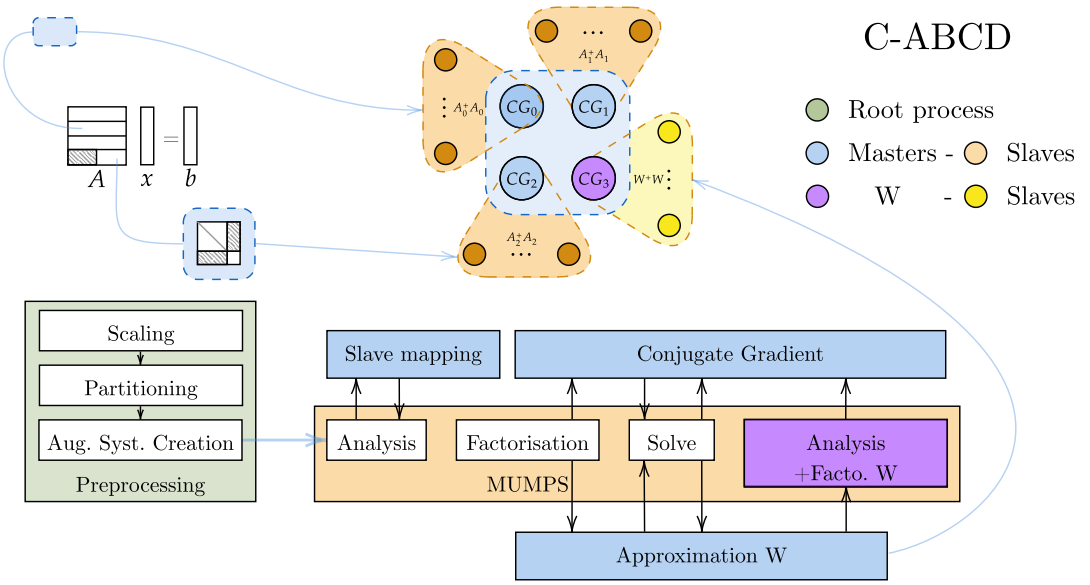


Figure C.2 Hybrid parallelisation scheme of the ABCD-Solver with the approximation of W and its inclusion as a partition in the global block-CG.

Appendix D

Résumé étendu [1]

> Mais enfin, ça fait 15 lieues que vous nous pétez les noyaux avec vos bestioles : les moutons, les chèvres, les poules, vous croyez que ça nous intéresse ça ?
> Oh la la, mais c'est pas vrai ! Les poules c'est plus ce que c'était, les chèvres c'est pas rentable, maintenant les moutons c'est fastidieux ! Vous savez même pas ce que ça veut dire fastidieux !
>
> Arthur, Kaamelott

L'intérêt de la simulation numérique est de mieux comprendre la physique derrière des systèmes physiques complexes. Suivant le schéma classique de la Figure D.1, ces simulations sont souvent basées sur des modèles d'équations aux dérivées partielles (EDP) couplées avec des conditions au bord [111]. Des méthodes de discrétisation, comme la méthode des éléments finis, appliquées à ces équations donnent de grands systèmes linéaires creux de la forme $Ax = b$ à résoudre [53, 64]. La solution de ces systèmes linéaires est alors utilisée afin de faire tourner la simulation numérique qui donne à son tour des informations sur le fonctionnement du système physique.
Dans ce travail, nous nous intéressons à la solution des systèmes linéaires creux qui est l'un des thèmes majeurs en calcul scientifique haute performance. Ces dernières

[1] Requis pour la rédaction du manuscrit en anglais. Ce résumé représente le contenu de la soutenance de thèse: `https://www.youtube.com/watch?fbclid=IwAR0nkXSJTiFi_T85KFQQHzc9ESzGU_IONUx72UrLmWgwAp-gCZ3YhU9x8Hw&v=jv5gCjFu8UU`.

décennies, des recherches poussées on été effectuées en vue de créer des méthodes pour la résolution de ces systèmes linéaires, appelés solveurs, pouvant être exécutées efficacement en parallèle sur des supercalculateurs.

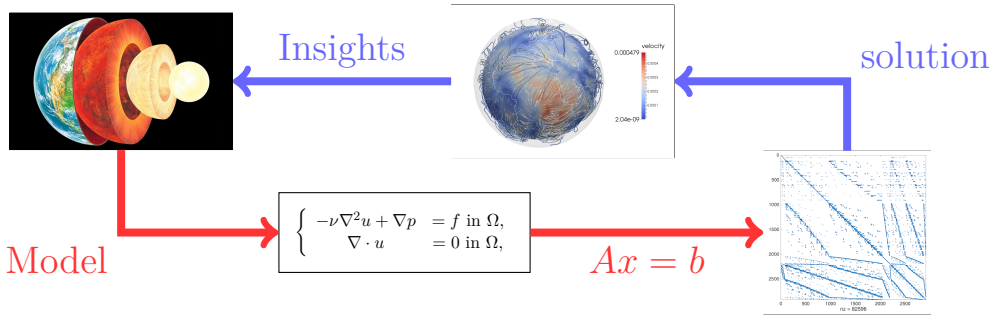


Figure D.1 The numerical simulation process involves the solution of sparse linear systems from the discretization of PDEs.

Historiquement, nous distinguons 2 classes de solveurs [128]. Les méthodes directes sont basées sur des techniques d'élimination de variables, menant à une factorisation de la matrice A, afin de résoudre le système linéaire [46, 61]. Les méthodes itératives quant à elles font successivement évoluer une approximation de la solution jusqu'à obtenir la précision désirée [75, 114]. Là où les méthodes directes se montrent robustes par rapport aux propriétés numériques du problème à résoudre, leur coût en calcul et mémoire est élevé comparé à celui des méthodes itératives, qui elles présentent une convergence très variable selon le problème. Introduites plus récemment, les méthodes hybrides ont été crées afin de bénéficier des avantages des 2 types de méthodes, directes et itératives, pour la solution de très grands systèmes sur des architectures de calcul parallèle.

Nous distinguons deux grandes classes de méthodes hybrides: les méthodes multigrilles et les méthodes de décomposition de domaine. La Figure D.2 présente le plan général du travail de thèse. Dans le chapitre 1, nous présentons un travail de recherche concernant l'amélioration de la scalabilité d'un schéma multigrille à de très grandes échelles, et plus précisément au niveau de la résolution du problème linéaire sur la grille la plus grossière. Dans la suite, nous étudions des méthodes de décomposition de domaine basées sur la méthode de projection par ligne Cimmino. Dans le chapitre 2, nous étendons une méthode bloc-itérative et une méthode pseudo-directe, originellement développées pour les systèmes carrés non-symétriques, à la solution de systèmes rectangulaires de rang pleins. L'implémentation parallèle de ces méthodes dans le ABCD-Solver est ensuite introduite dans le Chapitre 3, avec des méthodes de pré-traitement permettant d'améliorer l'efficacité de ces méthodes. Finalement, nous faisons le lien avec les méthodes

multigrilles en introduisant une méthode intermédiaire, entre les méthodes bloc-itérative et pseudo-direct, utilisant un niveau de grille grossière pour obtenir une convergence rapide et linéaire pour la solution de problèmes provenant de problèmes d'équations aux dérivées partielles.

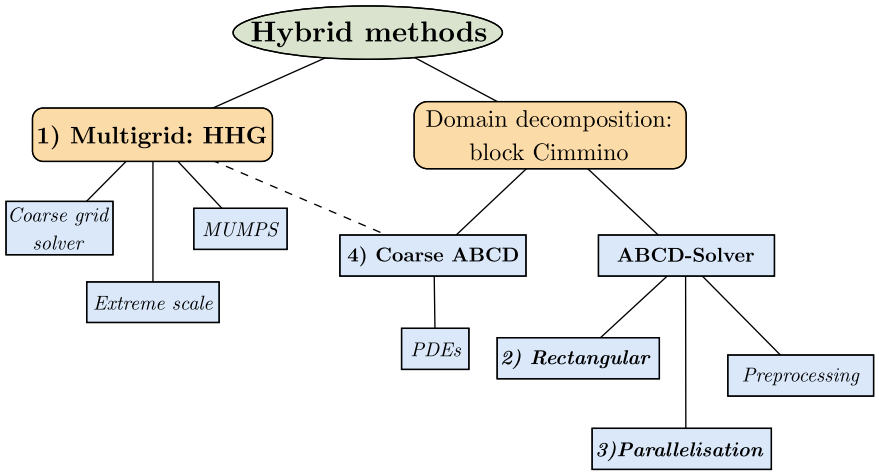


Figure D.2 In the PhD, we explore the world of hybrid methods for the solution of sparse linear systems.

D.1 Multigrille à très grande échelle

Dans le Chapter I, nous nous intéressons à la solution d'un problème exprimé sur une coquille sphérique, inspiré de la simulation de la convection du manteau terrestre, modélisé par les équations de Stokes généralisées:

$$\begin{aligned} -\operatorname{div}\left(\tfrac{\nu}{2}(\nabla\mathbf{u}+(\nabla\mathbf{u})^{\top})\right)+\nabla\mathbf{p} &= \mathbf{f} \quad \text{in } \Omega, \\ \operatorname{div}(\mathbf{u}) &= 0 \quad \text{in } \Omega, \\ \mathbf{u} &= \mathbf{g} \quad \text{on } \partial\Omega, \end{aligned} \tag{D.1}$$

avec $\mathbf{u}$ la vitesse, $\mathbf{p}$ la pression et f le forçage. Ces équations sont couplées à des conditions aux bords simplifiées, i.e.

- au niveau de la surface des conditions de Dirichlet utilisant des données réelles sur la vitesse du manteau,
- au niveau de la limite coeur-manteau des conditions de glissement libre. A noter qu'il s'agit là d'une simplification par rapport à la physique réelle du problème).

D.1.1 Problèmes à point de selle

Sur la coquille sphérique, nous construisons un maillage initial basé sur une division radiale et tangentielle de l'espace, voir Figure D.3. Une hiérarchie de grilles emboîtées est alors construite par raffinements successifs du maillage initial. Sur ces niveaux de grille, le problème de Stokes (D.1) est discrétisé en utilisant des éléments finis, et des opérateurs de prolongation sont construits, ici basés sur une interpolation linéaire. Selon la taille du maillage initial, nous obtenons 3 différentes tailles de problèmes résumées dans le Table D.1 avec jusqu'à 10^{11} degrés de libertés (DDL).

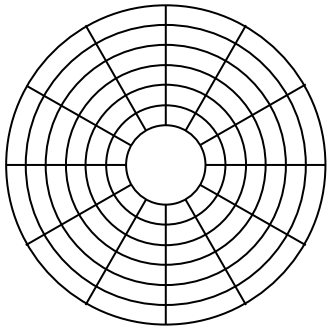


Figure D.3 Coupe transversale du maillage initial avec 6 divisions radiales et 12 divisions tangentielles.

Table D.1 Taille des problèmes fins obtenus à partir de différents maillages initiaux, avec le nombre de tétrahèdres et la résolution correspondantes. La taille du problème sur la grille grossière est également donnée.

Tetrahèdres	**Résolution** (km^2)	**DDL**	**DDL coarse**
1920	6.89	$5.37 \cdot 10^9$	$9.22 \cdot 10^4$
15360	3.44	$4.29 \cdot 10^{10}$	$6.96 \cdot 10^5$
43200	2.30	$1.21 \cdot 10^{11}$	$1.94 \cdot 10^6$

D.1.2 Problème de scalabilité

La question centrale ici est comment pouvons nous résoudre de tels problèmes efficacement, sachant que nous avons à disposition le supercalculateur Hazel Hen, classé 43ème du classement TOP500 de juin 2020. Ce supercalculateur contient presque 8 000 noeuds de calcul, et afin d'utiliser au mieux cette puissance de calcul parallèle, i.e. être scalable, nous devons utiliser des solveurs avec une complexité de calcul linéaire. Nous nous intéressons alors à des méthodes hybrides appelées méthodes multigrilles[27, 129]. En utilisant une

hiérarchie de grilles de taille plus ou moins grande, les méthodes multigrilles obtiennent la solution des systèmes linéaires grâce à deux principes. Tout d'abord, il a été observé que certaines classes de méthodes itératives, appelées lisseurs, atténuent rapidement les composantes d'erreur caractérisées par une haute fréquence d'oscillation, alors que les composantes à basse fréquence, i.e. lisses, restes pratiquement inchangées. De plus, les fonctions lisses sont bien représentées sur une grille grossière, où elles apparaissent comparativement plus oscillatoire. Ainsi, l'application d'une méthode itérative sur ces fonctions au niveau de la grille grossière devient à nouveau efficace. A l'intérieur d'un processus itératif global, les méthodes multigrilles utilisent ces deux aspects afin de calculer la solution du système linéaire en se déplaçant d'un niveau de grille à l'autre dans un cycle. Au niveau de la grille la plus grossière, le système est généralement considéré comme assez petit pour pouvoir être résolu avec un solveur direct sans utiliser trop de mémoire.

Cependant, alors que notre supercalculateur Hazel Hen possède près de 1 000 TB de mémoire, cela ne correspond qu'à une dizaine de vecteurs de taille 10^{13}, soit la plus grande taille de système linéaire résolue à ce jour. La seconde contrainte pour résoudre les problème du Table D.1 est alors d'utiliser une implémentation sans matrice assemblée. Ici, nous utilisons le framework multigrille HHG (Hierarchical Hybrid Grids)[18, 85]. Dans [66], les auteurs utilisent un solveur multigrille *"all-at-once"*, i.e. traitant simultanément la vitesse et la pression, couplé avec un lisseur de type Uzawa. Les auteurs conduisent alors une étude de scalabilité faible pour un problème similaire à (D.1). Les résultats de cette étude sont présentés en Figure D.4 et il est alors observé que l'efficacité parallèle, calculée par rapport au temps moyen du plus petit problème, se dégrade lorsque la taille du problème augmente. Alors que le temps nécessaire au traitement de la grille fine reste stable, le solveur appliqué sur la grille grossière est de plus en plus lent. Dans ce Chapter I, un premier but est de trouver une solution scalable sur la grille grossière. De plus, la viscosité μ utilisée dans le problème (D.1) peut être considérée simplement constante, ou nous pouvons considérer une viscosité variant soudainement afin de refléter plus fidèlement la limite lithosphère-asthénosphère du manteau terrestre 410km. Ce changement de coefficient de viscosité peut avoir un impact important sur l'efficacité du solveur sur la grille grossière et notre second but est de trouver un solveur robuste sur la grille grossière.

D.1.3 Solveur sur la grille grossière

La première approche pour résoudre le problème sur la grille la plus grossière est la méthode de minimisation du résidu préconditionnée (PMINRES), native dans le framework

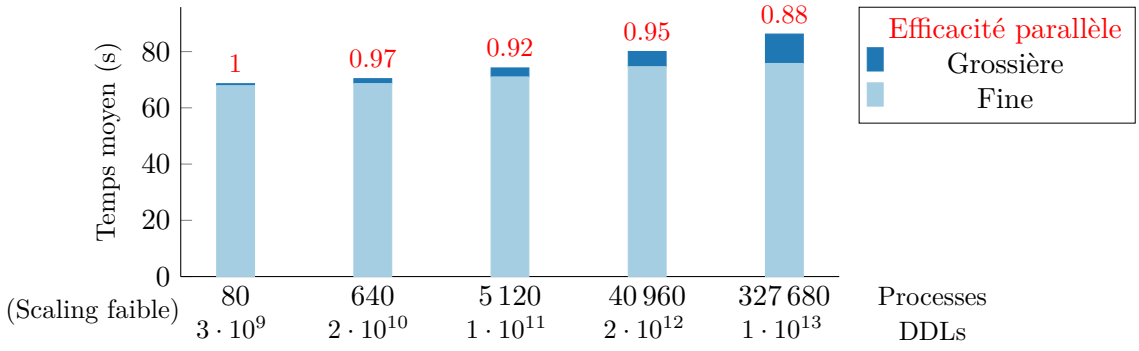


Figure D.4 Scalabilité faible pour un problème similaire à (D.1). Nous distinguons le temps moyen pour le traitement des grilles fine et de la grille la plus grossière. L'efficacité parallèle est calculée par rapport au temps moyen du plus petit problème.

HHG. Cette approche a l'avantage d'être facilement parallélisée et utilise l'implémentation sans matrice assemblée de HHG. Par contre, la convergence de cette méthode dépend beaucoup du problème. En Figure D.5, nous montrons une étude de scalabilité faible effectuée sur les 3 tailles de problèmes à notre disposition avec une viscosité constante (*iso-viscous*), soit une viscosité avec saut (*jump-410*). Nous observons que le temps nécessaire pour traiter la grille grossière augmente avec la taille du problème. C'est d'autant plus vrai lorsque l'on considère une viscosité variante pour laquelle l'efficacité parallèle du solveur multigrille global descend alors en dessous de 80%.

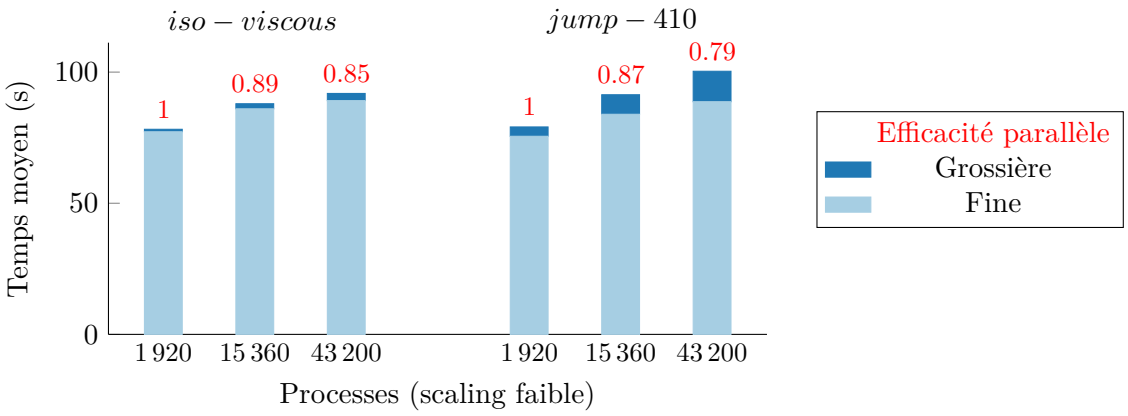


Figure D.5 Scalabilité faible pour le problème (D.1) avec viscosité (Gauche) *iso-viscous* et (Droite) *jump-410*. Nous distinguons le temps moyen pour le traitement des grilles fine et de la grille la plus grossière. L'efficacité parallèle est calculée par rapport au temps moyen du plus petit problème.

Nous proposons comme alternative l'utilisation d'un solveur direct approximé, ici le *MUltifrontal Massively Parallel Solver* (MUMPS)[2] [6]. Ce solveur, basé sur une élimination gaussienne, permet de résoudre des systèmes linéaires en parallèle grâce à la méthode multifrontale. Après une phase de *Setup*, pendant laquelle le solveur analyse puis factorise la matrice ($A = LU$), une phase de calcul de la *solution*, très peu chère en calcul et en mémoire, permet d'obtenir la solution pour un second membre spécifique en utilisant la factorisation de la matrice. Le Table D.2 montre les résultats obtenus avec MUMPS sur les matrices correspondant à la grille grossière du problème le plus large que nous traitons, avec viscosité constante ou non. Nous distinguons les temps de setup et de solution, et montrons dans chaque cas la précision de la solution calculée en terme de résidu scalé. Nous observons alors que MUMPS, au contraire de PMINRES, est très robuste par rapport à la viscosité en terme de temps de calcul et de précision de la solution. Cette robustesse a un prix, d'une part MUMPS nécessite une matrice totalement assemblée, ce qui n'est pas un problème puisque le problème grossier est relativement petit, et d'autre part le setup est longue, à elle seule cette phase prend 180s alors que le temps total pour PMINRES dans HHG est de 165.5s.

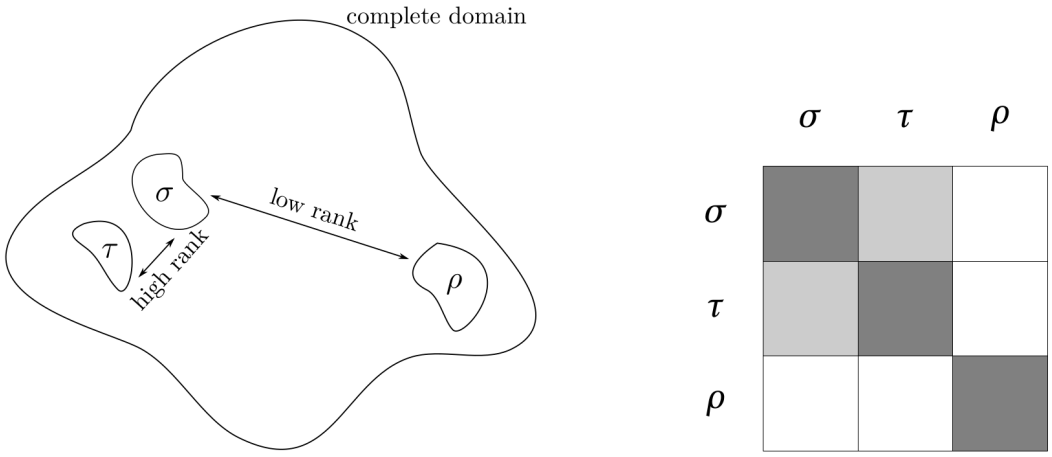
Table D.2 MUMPS on the largest problem ($1.94 \cdot 10^6$ *DOFs*) with $jump - 410$.
FYI: PMINRES runtime is 165.5s

Type	**Time (s)**		**Scaled residual**
	Setup	Solve	
$iso - viscous$	180.3	0.55	$8 \cdot 10^{-19}$
$jump - 410$	175.6	0.56	$2 \cdot 10^{-18}$

Ce prix peut être compensé. Tout d'abord, le schéma multigrille se fait au travers de plusieurs itérations d'un cycle-V dans lequel la matrice grossière reste constante. La factorisation peut alors être calculée une seule fois, son prix est divisé par le nombre d'itérations où seule la phase de solution peu chère reste à calculer en fonction du second membre. De plus, MUMPS offre un mécanisme appelé *approximation par blocs de rang faible* (BLR)[1, 4]. Son principe est d'approximer des blocs loin de la diagonale dans la matrice par des blocs de rang faible, en utilisant le fait que deux variables éloignées ont généralement peu d'effet l'une sur l'autre, voir Figure D.6. Grâce à ce mécanisme, il est possible de contrôler la précision de la solution en obtenant une réduction du coup de la factorisation. En effet, une approximation de la solution est suffisante sur la grille la plus grossière, e.g. PMINRES est arrêté avec une précision de 10^{-3}. Lorsque BLR donne une solution inférieur à 10^{-8}, il est alors possible d'utiliser le mode simple précision

[2] `http://mumps-solver.org/`

de MUMPS pour un gain en temps supplémentaire, sans perte de précision. Dans ce contexte, cette solution peut être idéale.



(a) Strong and weak interactions between clusters in the geometric domain.

(b) Corresponding block-clusters in the matrix.

Figure D.6 Source: thèse de Theo Mary [100]

Implémenter une méthode multigrille de manière à ce qu'elle se comporte bien sur des systèmes de calculs très larges est difficile[109, 110]. Dans ce cas, en partant d'une grille très fine, résolue avec beaucoup d'unités de calcul (coeurs), et en se déplaçant vers une grille plus grossière, le nombre de variables par coeur devient de plus en plus petit, voir Figure D.7. L'efficacité en calcul se détériore alors rapidement si tous les coeurs sont utilisés à tous les niveau. L'agglomération est une solution très simple à ce problème

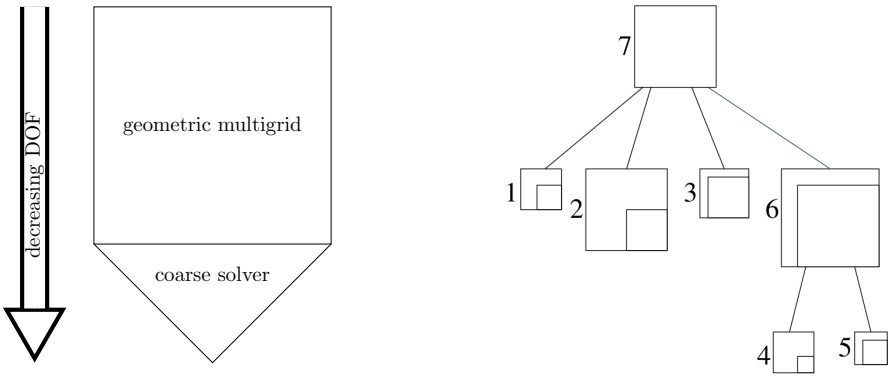


Figure D.7 *(Gauche)* Les niveaux de grille grossiers ont de moins en moins de DDL par coeur, *(Droite)* ce qui entraîne une baisse de la granularité des sous-problèmes résolus dans MUMPS.

qui consiste à utiliser sur la grille grossière seulement un sous-ensemble m de tous les

coeurs $|P|$ obtenue via un facteur de réduction r tel que $m = |P|/r$. Nous proposons 2 stratégies d'agglomération, voir Figure D.8:

1. *Schéma Maître-Ouvrier*: parmi tous les coeurs utilisés pour le traitement des grilles fines, un sous-ensemble est sélectionné sur lequel les DDL de la grille grossière sont rassemblés et traités avec le solveur direct.
2. *Superman*: un certain nombre de noeuds est dédié dès le départ au traitement de la grille grossière. Ce schéma a l'avantage que la factorisation de la matrice du niveau grossier peut être effectué en parallèle de la première branche du premier cycle multigrille. Au prix d'un (petit) nombre de coeur additionnels, le coût de la factorisation est alors virtuellement réduit (voire supprimé).

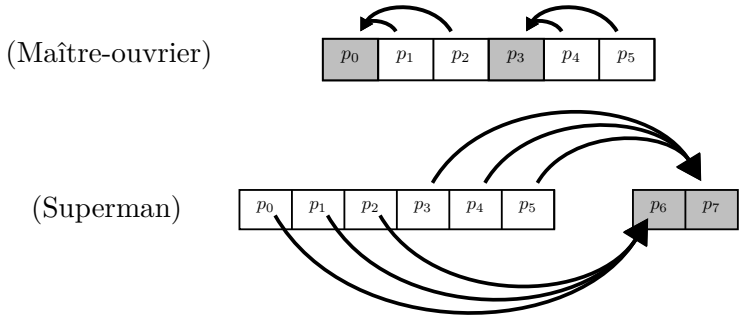


Figure D.8 Schémas d'agglomération proposés.

Le problème de ce schéma d'agglomération est que le choix du facteur de réduction r doit se faire au cas par cas. Dans la Figure D.9, nous montrons le temps d'exécution de MUMPS sur la matrice grossière pour les 3 tailles de problème avec un facteur de réduction qui augmente. A partir de ces résultats, nous pouvons obtenir le facteur donnant l'exécution la plus rapide et l'utiliser pour nos tests. Nous pouvons alors introduire l'approximation BLR et l'utilisation du calcul simple précision. Pour MUMPS appliqué au problème de plus grande taille, le Table D.3 présente la réduction du temps d'exécution et la précision obtenue avec ces approximations. Nous observons que BLR permet de diviser le temps d'exécution de la factorisation par 3 au prix de quelques secondes de plus dans l'analyse de la matrice, pour calculer les blocs de rangs faibles, et d'une solution de précision 10^{-6}. L'utilisation du calcul simple précision peut alors être activé pour une réduction supplémentaire de 30% de la factorisation sans changer la solution.

Enfin, nous pouvons inclure MUMPS avec approximation à l'intérieur du framework HHG pour une étude de scalabilité faible sur nos 3 tailles de problème. Les résultats sont présentés en Figure D.10 où nous comparons le temps d'exécution en utilisant les

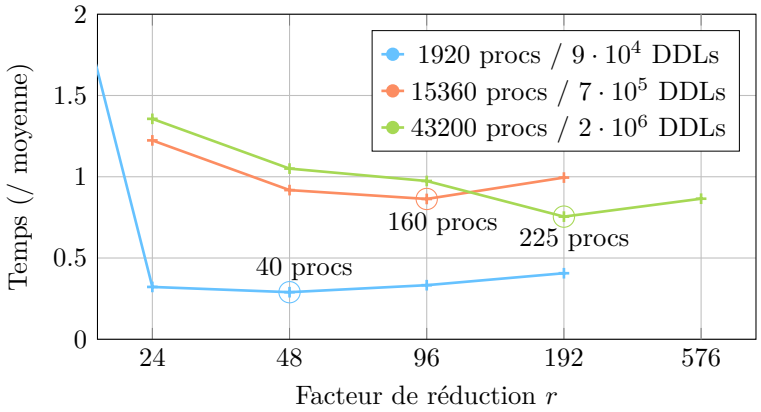


Figure D.9 Temps d'exécution de MUMPS pour les 3 tailles de problème selon le facteur de réduction utilisé pour l'agglomération.

Table D.3 MUMPS sur le problème le plus gros ($1.94 \cdot 10^6$ *DOFs*) with $jump - 410$.
FYI: PMINRES runtime is 165.5s

BLR ϵ	**Analyse**	**Factorisation**		**Solution**	**Résidu scalé**
	Temps (s)	*Flops (%)*	*Temps (s)*	*Temps (s)*	
Rang plein	41.0	100	134.6	0.56	$2 \cdot 10^{-18}$
10^{-5}	47.6	13	37.0	0.30	$2 \cdot 10^{-6}$
10^{-5} + simple	47.7	13	25.6	0.27	$1 \cdot 10^{-6}$

différents solveurs sur la grille grossière: *(P)* PMINRES, MUMPS approximé couplé avec *(M)* l'agglomération Maître-Ouvrier et *(S)* l'agglomération Superman. En utilisant MUMPS avec l'agglomération Maître-Ouvrier, le temps de traitement de la grille la plus grossière est réduit de 50% grâce à l'approximation BLR et le calcul simple précision[30]. Additionnellement, grâce à l'agglomération Superman, le temps du setup peut-être virtuellement caché puisque effectué en parallèle avec la première branche du premier cycle multigrille. Le résultat est une réduction du temps de traitement de la grille grossière de 80% par rapport à l'utilisation de PMINRES, ce qui donne une amélioration de l'efficacité parallèle du framework HHG de 79% à 88%.

Cette approche est une contribution importante pour la simulation numérique et, dans notre cas, pour la simulation du manteau terrestre [16].

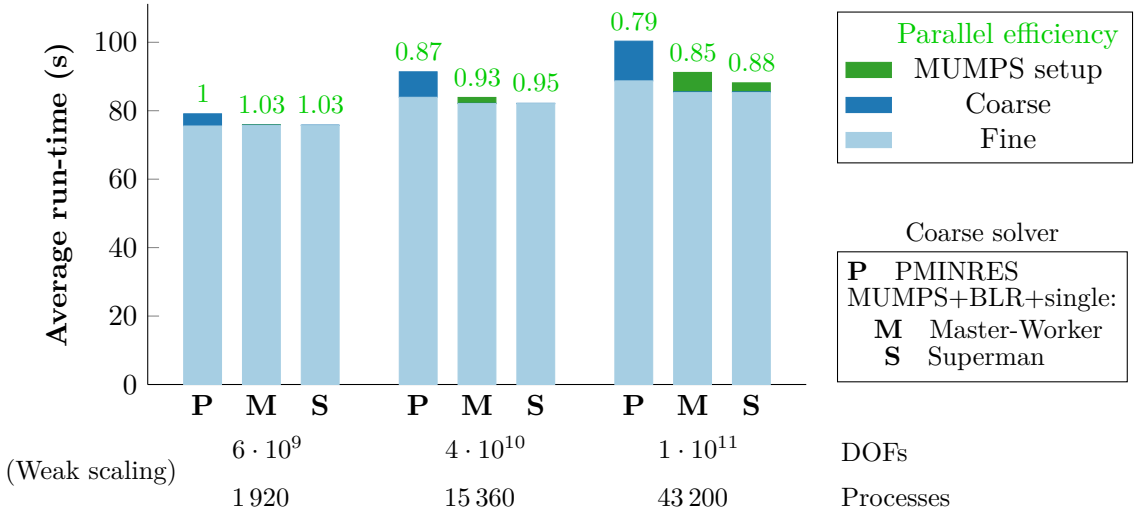


Figure D.10 Comparaison de 3 solveurs sur la grille grossière dans HHG pour les 3 tailles du problème (D.1): PMINRES (P); MUMPS utilisant BLR et simple précision couplé avec agglomération Maître-Ouvrier (M) ou agglomération Superman (S).

D.2 Méthodes Cimmino par blocs, itérative et pseudo-directe

L'autre grande classe de méthodes hybrides sont les méthodes de décomposition de domaine (DDM) [40, 98, 127] qui décomposent le problème en sous-problèmes résolus indépendamment, afin d'être efficaces en terme de calcul parallèle. Nous nous intéressons plus particulièrement à la méthode de projection par bloc de lignes *bloc Cimmino*[34, 52, 88], pouvant être interprétée en tant que méthodes de décomposition de domaine[40]. Une itération de la méthode Cimmino est très simple à visualiser et nous en montrons un exemple en 2 dimensions dans la Figure D.11. Dans ce cas, trouver la solution d'un système linéaire revient à trouver l'intersection de 2 lignes, chacune correspondant à une équation. Appliquer une itération de Cimmino consiste alors à calculer la projection d'un itéré courant sur chacune de ces lignes puis à prendre une somme pondérée de ces projections pour obtenir l'itéré suivant.

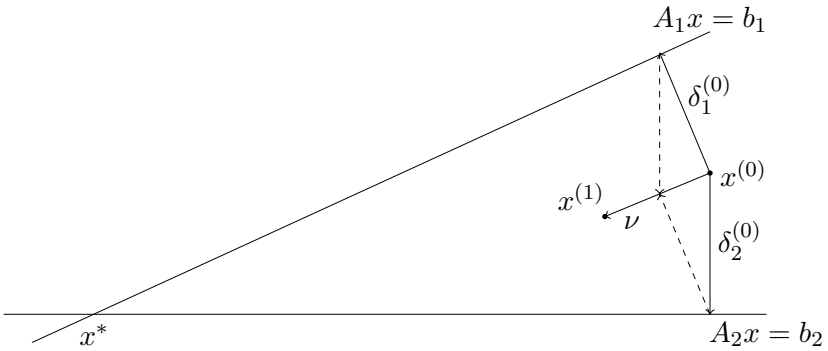


Figure D.11 Itération en 2D de la méthode de projection par ligne Cimmino. *Source:* [133]

D.2.1 Accélération de la méthode Cimmino par blocs

Ce processus peut être généralisé très facilement. Considérons la matrice carrée, inversible $A \in \mathbb{R}^{n \times n}$, partitionnée en p blocs de ligne tel que l'on résout le système partitionné:

$$\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_p \end{bmatrix} x = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p, \end{bmatrix} \tag{D.2}$$

avec x et b des vecteurs de taille n. Dans ce cas, une ligne du problème 2D devient un sous-espace engendré par une partition, et l'application d'une itération de Cimmino revient toujours à calculer une somme pondérée des projections de l'itéré courant. Cette itération est exprimée par

$$\begin{aligned} \delta^{(i)} &= A_i^+(b_i - A_i x^{(k)}),\ i = 1, \ldots, p, \\ x^{(k+1)} &= x^{(k)} + \omega \sum_{i=1}^{p} \delta^{(i)}. \end{aligned} \tag{D.3}$$

Cette méthode peut-être interprétée comme une méthode de décomposition de domaine, plus précisément une méthode de Schwartz abstraite additive sur les équations normales, puisque Cimmino par bloc revient à appliquer un Jacobi par bloc sur les équations normales. Comme la convergence de cette méthode est souvent lente, nous considérons le

point fixe des itérations (D.3) pour obtenir le nouveau système équivalent

$$Hx = K \quad \text{avec} \quad \begin{cases} H = \sum_{i=1}^{p} \mathcal{P}_{\mathcal{R}(A_i^T)} = \sum_{i=1}^{p} A_i^+ A_i, \\ K = \sum_{i=1}^{p} A_i^+ b_i. \end{cases}$$

La *matrice d'itération* H est symétrique positive définie (SPD) et, afin d'accélérer les itérations de Cimmino par bloc, nous pouvons utiliser un algorithme de gradient conjugué pour le résoudre. Une des principale contribution du Chapter II est alors d'étendre cette approche, originellement pensée pour les systèmes carrés, à la solution de systèmes rectangulaires.

Une bonne nouvelle pour commencer, l'approche précédente est directement valable les systèmes sous-déterminés de rang *ligne* plein, avec un partitionnement en blocs de *ligne*. Quant aux systèmes sur-déterminés, il faut considérer une matrice de rang *colonne* plein, avec un partitionnement en blocs de *colonnes*. Afin d'unifier les approches pour ces 2 types de systèmes, notamment dans la notation, nous considérons que la matrice A est toujours sous-déterminée et de rang ligne plein. Nous cherchons alors à calculer soit la solution de norme minimale d'un problème basé sur A, soit la solution du problème de moindre carré basé sur A^T, i.e. où chaque solution est calculée grâce au pseudo-inverses

Sous-déterminé: (solution de norme-minimale)	**Sur-déterminé:** (problème de moindres carrés)
$\min_{x \in \mathbb{R}^n} \|x\|_2$ tel que $Ax = b$	$\min_{\widetilde{x} \in \mathbb{R}^n} \left\| A^T \widetilde{x} - \widetilde{b} \right\|_2$
$\implies x_{mns} = A^+ b = A^T (AA^T)^{-1}$	$\implies x_{ls} = (A^T)^+ \widetilde{b} = (AA^T)^{-1} A\widetilde{b}$

de Moore-Penrose A^+ et $(A^T)^+$. Nous remarquons alors que pour le choix spécifique de second membres $b = A\widetilde{b}$, les solutions des systèmes sous- et sur-déterminés sont liés par la simple relation $x_{mns} = A^T x_{ls}$. Nous utilisons ce simple fait seulement pour simplifier les calculs et notations.

Nous pouvons maintenant développer la même accélération de la méthode Cimmino par bloc que précédemment en considérant les problèmes partitionnés

$$\min_{x \in \mathbb{R}^n} \|x\|_2 \text{ tel que } \begin{bmatrix} A_1 \\ \vdots \\ A_p \end{bmatrix} x = \begin{bmatrix} b_1 \\ \vdots \\ b_p \end{bmatrix} \qquad \min_{x \in \mathbb{R}^n} \left\| \begin{bmatrix} A_1^T & \dots & A_p^T \end{bmatrix} \begin{bmatrix} \widetilde{x}^1 \\ \vdots \\ \widetilde{x}^p \end{bmatrix} - \widetilde{b} \right\|_2$$

sur lesquels nous pouvons appliquer l'itération correspondante de Cimmino par blocs

Sous-déterminé:

$$\begin{aligned}\delta^{(i)} &= A_i^+(b_i - A_i x^{(k)}),\ \forall i,\\ x^{(k+1)} &= x^{(k)} + \omega \sum_{i=1}^{p} \delta^{(i)},\end{aligned}$$

Sur-déterminé:

$$\begin{aligned}\delta^{(i)} &= {A_i}^+ A_i r^{(k)}\\ x_i^{(k+1)} &= x_i^{(k)} + \omega (A_i^T)^+ r^{(k)},\ \forall i,\\ r^{(k+1)} &= r^{(k)} - \omega \sum_{i=1}^{p} \delta^{(i)}.\end{aligned}$$

En considérant à nouveau le point fixe des itérations, nous obtenons le nouveau système à résoudre $Hx = k$ avec où $D = blkdiag(A_1 A_1^T, \ldots, A_p A_p^T)$. Nous pouvons alors

$$\begin{aligned}H^{row} &= \sum_{i=1}^{p} \mathcal{P}_{\mathcal{R}(A_i^T)} = \sum_{i=1}^{p} A_i^+ A_i\\ &= A^T D^{-1} A\end{aligned}$$

$$\begin{aligned}H^{col} &= \begin{bmatrix}(A_1^T)^+\\ \vdots\\ (A_p^T)^+\end{bmatrix} \begin{bmatrix}A_1^T & \ldots & A_p^T\end{bmatrix}\\ &= D^{-1} A A^T\end{aligned}$$

considérer la résolution de ces systèmes à l'aide d'un gradient conjugué car

- H^{row} est cette fois symétrique **semi-**définie positive, dans un système qui reste consistent. Cette matrice correspond à la somme des projections sur les partitions.
- H^{col} n'est pas symétrique mais est symétrisable au sens de Hageman&Young [75]. Cette matrice correspond à une concaténation des pseudo-inverses des partitions.

Dans le but d'accélérer un peu plus la convergence de l'algorithme, nous utilisons un algorithme de gradient conjugué par bloc stabilisés aux systèmes rectangulaires. Le but de cet algorithme, que nous avons étendu aux problèmes de moindres carrés, est de réduire les possibles plateaux dans la convergence du gradient conjugué classique, voir Figure D.12 pour un exemple. Ces plateaux sont notamment dûs à des groupes de petites valeurs propres dans le spectres de H. Même si les plateaux sont réduits, chaque itération est alors plus chère mais grâce à l'utilisation de librairies très efficaces pour le calcul d'opérations matrice-vecteur, nous obtenons une amélioration du temps de calcul comparé au gradient conjugué classique pour un choix de taille de bloc raisonnable. Dans cet algorithme, que ce soit pour la solution de problèmes sous- ou sur-déterminés, il est nécessaire à chaque itération de calculer une somme de projections sur les partitions, i.e. $\mathcal{P}_{\mathcal{R}(A_i)^T} = A_i^+ A_i$, qui sont obtenues via la solution des systèmes linéaires

$$\begin{pmatrix} I_n & A_i^T \\ A_i & O_{m_i} \end{pmatrix} \begin{pmatrix} u_i \\ v_i \end{pmatrix} = \begin{pmatrix} r \\ z_i \end{pmatrix}, \tag{D.4}$$

avec un solveur direct parallèle. Grâce à l'indépendance entre les projections, et 2 niveaux de parallélisme inhérents au solveur direct, nous obtenons naturellement un schéma de parallélisation hybrides pour l'implémentation.

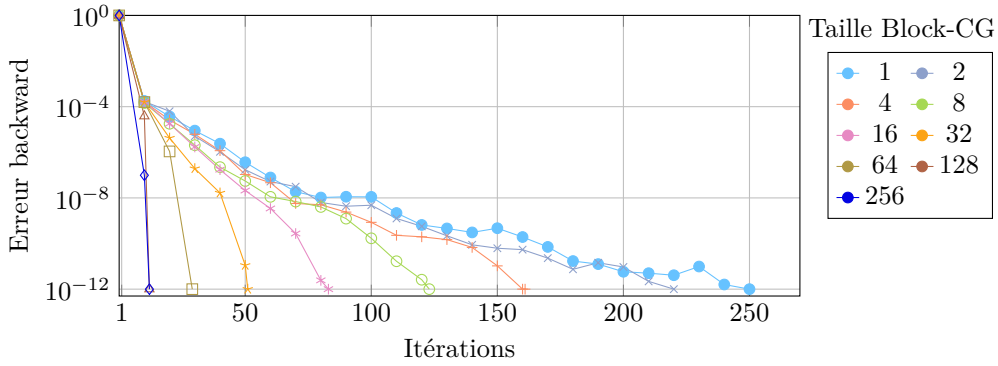


Figure D.12 Convergence de BCG: `deltaX` ($7 \cdot 10^4 \times 2 \cdot 10^4$) avec 16 partitions

Nous obtenons ainsi la méthode Cimmino par blocs accélérée (BC) pour la solution de systèmes rectangulaire. Pourtant, malgré l'utilisation d'un paradigme par blocs, et même en considérant l'utilisation de pré-traitements du systèmes, la convergence de cette méthode reste problème dépendante et peut être caractérisée par de long plateaux. En effet, le spectre de la matrice d'itération H dépend de l'ouverture des angles entre les sous-espaces engendrés par les partitions. Une alternative basée sur l'augmentation du système original avec des variables et contraintes additionnelles a été proposée dans [46]. Cette augmentation permet d'orthogonaliser mutuellement les partitions, garantissant ainsi la convergence en une seule itération de la méthode BC.

D.2.2 Méthode Cimmino par blocs augmentée

Nous considérons encore une matrice A sous-déterminée de rang ligne plein et partitionnée par blocs de lignes A_i. Le principe de l'augmentation [46] est le bloc $\mathcal{F}(A)$ aux colonnes de la matrice. Nous obtenons la matrice augmentée $\overline{A} = \begin{bmatrix} A & \mathcal{F}(A) \end{bmatrix}$ dans laquelle les partitions augmentées $\overline{A}_i$ sont mutuellement numériquement orthogonales, i.e. $\forall\ i, j = 1, \ldots, p,\ i \neq j,\ \overline{A}_i \overline{A}_j^T = A_i A_j + \mathcal{F}(A)_i \mathcal{F}(A)_j = 0$. La Figure D.13 montre l'augmentation obtenue à partir d'une matrice bloc tri-diagonale. Il y a plusieurs techniques d'augmentation, et la plus simple duplique les blocs d'intéraction de chaque couple de partition avec un signe moins sur la deuxième partition.

Une fois la matrice augmentée, le système est fermé à l'aide d'un bloc additionnel simple $Y = \begin{bmatrix} 0 & Y \end{bmatrix}$. Et comme ce bloc n'est pas orthogonal à $\overline{A}$, Y^T est projeté sur le

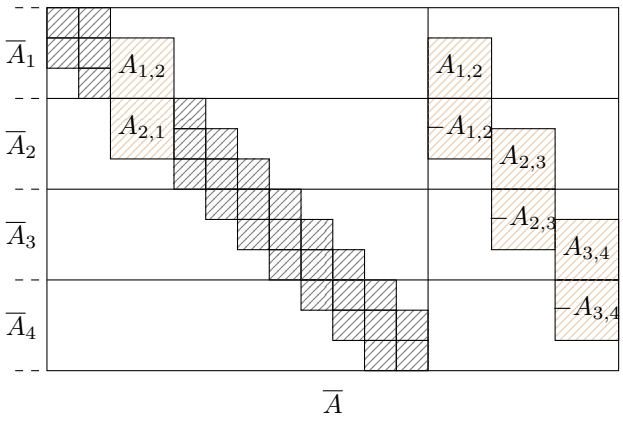


Figure D.13 Augmentation d'une matrice tri-diagonale par bloc.

noyau de $\overline{A}$ pour obtenir la matrice augmentée

$$\begin{bmatrix} \overline{A} \\ W \end{bmatrix} = \begin{bmatrix} A & \mathcal{F}(A) \\ B & S \end{bmatrix} \quad avec \quad \begin{cases} W & = & \begin{bmatrix} B & S \end{bmatrix} = Y(I - \overline{A}rP), \\ \overline{A}rP & = & \mathcal{P}_{\mathcal{R}(\overline{A}^T)}. \end{cases} \tag{D.5}$$

La bonne nouvelle c'est que grâce à l'orthogonalité entre partitions, le projecteur orthogonal sur $\mathcal{R}(\overline{A}^T)$ devient une somme de projecteurs orthogonaux, i.e. $\mathcal{P} = \sum_{i=1}^{p} \mathcal{P}_{\mathcal{R}(\overline{A}_i^T)}$. Le même schéma de parallélisation que la méthode itérative reste donc envisageable. De plus, nous obtenons une nouvelle matrice S qui a de bonnes propriétés. Tout d'abord cette matrice est SPD et correspond aux équations normales du bloc W ($S = WW^T$). De plus, si nous considérons les équations normales de la matrice augmentée avec seulement Y, nous obtenons

$$\begin{bmatrix} \overline{A} \\ Y \end{bmatrix} \begin{bmatrix} \overline{A} \\ Y \end{bmatrix}^T = \begin{bmatrix} \overline{A}_1\overline{A}_1^T & & & \mathcal{F}(A)_1 \\ & \ddots & & \vdots \\ & & \overline{A}_p\overline{A}_p^T & \mathcal{F}(A)_p \\ \mathcal{F}(A)_1^T & \dots & \mathcal{F}(A)_p^T & I_q \end{bmatrix}. \tag{D.6}$$

La forme de ces équations normales, une matrice bloc diagonale bordée, est typique des méthodes de décomposition de domaine. Il est alors naturel de calculer le complément de Schur de ces équations, $Schur = I_q - \sum_{i=1}^{p} \mathcal{F}(A)_i^T(\overline{A}_i\overline{A}_i^T)^{-1}\mathcal{F}(A)_i = S$, qui correspond tout simplement à la matrice S du schéma d'augmentation. Là où la méthode Cimmino par bloc correspond à une méthode de Schwartz abstraite additive, la méthode Cimmino par bloc augmentée est une méthode de complément de Schur, également sur les équations

normales. L'augmentation sert alors à condenser l'information sur l'interface entre les sous-domaines correspondant aux partitions dans la matrice.

Maintenant que le système est augmenté, nous allons voir comment reconstruire la solution du système originale à partir de la solution du système augmenté. Pour commencer, comme dans [46], nous considérons une matrice A carrée inversible et le système augmenté

$$\begin{bmatrix} \overline{A} \\ W \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} A & C \\ B & S \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ f \end{bmatrix}, \tag{D.7}$$

où y et f sont respectivement des variables et second membre additionnels. Le choix spécifique $f = -Y\overline{A}^{+}b$ permet de forcer $y = 0$. Nous garantissons ainsi que la solution du système augmenté est $\begin{bmatrix} x_{mns} \\ 0 \end{bmatrix}$ où x_{mns} est la solution du système original. Grâce à l'orthogonalité entre $\overline{A}$ et W, cette solution est obtenue en une seule itération de Cimmino par bloc, i.e.

$$\begin{bmatrix} x_{mns} \\ 0 \end{bmatrix} = \overline{A}^{+}b + W^{+}f. \tag{D.8}$$

De plus, comme $W = Y(I - \overline{P})$ est de rang plein, que $S = WW^T$, et en utilisant $f = -Y\overline{A}^{+}b$, on obtient la meilleure forme pour la solution

$$\begin{aligned} \begin{bmatrix} x_{mns} \\ 0 \end{bmatrix} &= \overline{A}^{+}b - W^T(WW^T)^{-1}Y\overline{A}^{+}b, \\ &= \overline{A}^{+}b - (I - \overline{P})Y^T S^{-1}Y\overline{A}^{+}b. \end{aligned} \tag{D.9}$$

La question du Chapter II est alors la même, comment étendre cette approche développée pour des systèmes carrés à des systèmes rectangulaires ? Concernant les systèmes sous-déterminés, l'approche est directement valable ! La seule différence est au niveau théorique, puisqu'il faut démontrer que les solutions x_{mns} et $\begin{bmatrix} x_{mns} \\ 0 \end{bmatrix}$ sont de norme minimale. Quant aux systèmes sur-déterminés, comme dit précédemment, nous utilisons la transposée du système augmenté et résolvons le problème de moindres carrés

$$\min_{(\widetilde{x},\widetilde{y}) \in \mathbb{R}^{n} \times \mathbb{R}^{q}} \left\| \begin{bmatrix} A^T & B^T \\ C^T & S \end{bmatrix} \begin{bmatrix} \widetilde{x} \\ \widetilde{y} \end{bmatrix} - \begin{bmatrix} \widetilde{b} \\ \widetilde{f} \end{bmatrix} \right\|_2. \tag{D.10}$$

A nouveau, le choix spécifique $\widetilde{f} = S^{-1}Y\overline{A}\left(\overline{A}^T\right)^{+}\begin{bmatrix} \widetilde{b} \\ 0 \end{bmatrix}$ permet d'annuler les variables additionnelles ($\widetilde{y} = 0$). Grâce à l'orthogonalité entre $\overline{A}^T$ et W, nous obtenons la solution

de ce système augmenté en une itération et la solution du système original x_{ls} est construite selon

$$\begin{aligned} x_{ls} &= \left(\overline{A}^T\right)^+ \begin{bmatrix} \widetilde{b} \\ \widetilde{f} \end{bmatrix} \\ &= \left(\overline{A}^T\right)^+ \begin{bmatrix} \widetilde{b} \\ 0 \end{bmatrix} + \left(\overline{A}^T\right)^+ Y^T S^{-1} Y \overline{A}^+ \overline{A} \begin{bmatrix} \widetilde{b} \\ 0 \end{bmatrix} \end{aligned} \tag{D.11}$$

Pour toute forme de matrice, nous obtenons ainsi la *méthode Cimmino par bloc augmentée* (ABCD) étendue dans le Chapter II à la solution de systèmes rectangulaires. Comme illustré dans la Figure D.14, alors que la méthode itérative accélérée Cimmino par bloc (BC) a une convergence très dépendante de l'ouverture des angles entre les partitions, dans la méthode ABCD permet d'obtenir la solution en une itération grâce à l'orthogonalité mutuelle entre les espaces engendrés par les partitions augmentées. C'est une méthode pseudo-directe.

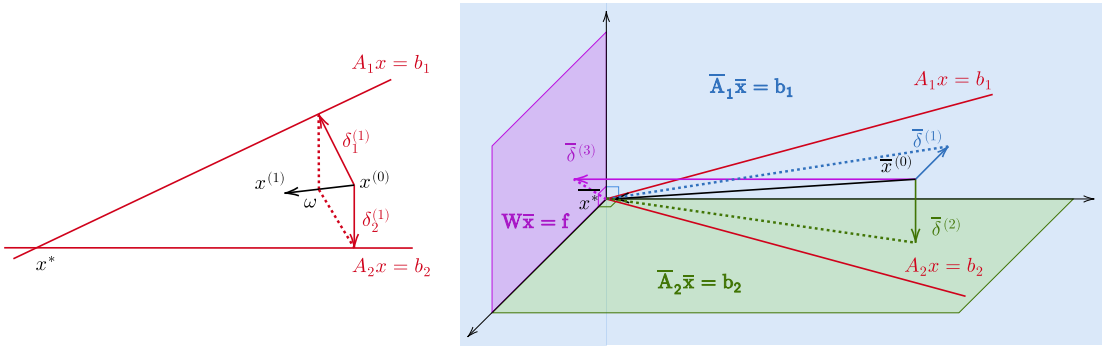


Figure D.14 Cimmino par bloc vs Cimmino par bloc augmenté

La forme de la solution de norme minimale (D.9) pour les systèmes sous-déterminés, et de la solution (D.9) des problèmes de moindres carrés ont une forme très similaire. Ces solutions sont exprimées en 2 parties, une pour $\overline{A}$ et une pour W, et ne font intervenir que quelques éléments:

1. le calcul de pseudo-inverses indépendants, comme dans BC. En effet, dans les solutions apparaissent des pseudo-inverses $\overline{A}^+$ qui, grâce à l'orthogonalité entre partitions, peuvent être décomposés en éléments indépendants. Nous avons par exemple $\overline{A}^+ b = \sum_{i=1}^{p} \overline{A}_i^+ b$.
2. l'inverse du complément de Schur S, qui peut être obtenu en résolvant un système Sz=f avec un solveur direct parallèle. La résolution d'un tel système n'est possible

que si S, et donc l'augmentation, sont de petite taille. La Figure D.15 présente les tailles d'augmentations, relativement à la plus grande dimension du système original, obtenues pour tous les systèmes de plus de 1000 lignes ou colonnes extraits de la SuiteSparse Matrix Collection. Nous observons alors que deux-tiers des systèmes, toutes structures confondues, ont une taille d'augmentation inférieure à 20% de la taille du système original. Ce seuil est considéré comme généralement acceptable dans [46], et la solution du complément de Schur est réaliste. Le point clé de la méthode ABCD est alors la construction de manière "embarrassingly parallel" du complément de Schur à l'aide des projections indépendantes.

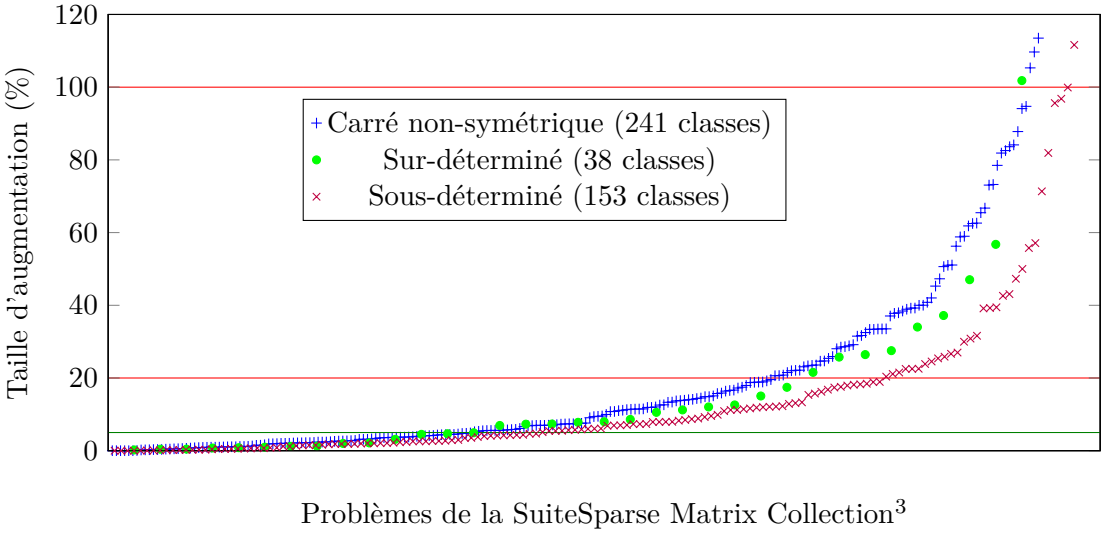


Figure D.15 Distribution de la taille d'augmentation, relatif à la taille du système original, pour 38 problèmes sur-déterminés *(points verts)*, 153 problèmes sous-déterminés *(croix rouges)* et 241 problèmes carrés non-symétriques *(signes plus bleus)* problèmes, classés par taille d'augmentation croissante. Les seuils de 5%, 20% et 100% sont indiqués.

Le calcul de ces éléments permet d'utiliser le même type de schéma de parallélisation que BC. L'implémentation des méthodes BC et ABCD pour la solution de systèmes rectangulaires a été développé pendant la thèse dans le ABCD-Solver[4] [133] et nous allons maintenant détailler cette parallélisation.

[4]http://abcd.enseeiht.fr/

D.3 Implémentation parallèle: ABCD-Solver

Afin d'améliorer les propriété numériques des méthodes itératives et pseudo-directes, des techniques de pré-traitement sont présentées dans la première partie du Chapter III. Après avoir atténué la disparité dans les facteurs d'échelle de la matrice via une normalisation simultanée des lignes et des colonnes dans le cas des matrices carrées [113], les différentes méthodes de partitionnement existantes sont discutées [42, 112]. Nous nous intéressons en particulier à des méthodes de partitionnement de graphes qui cherchent, soit à accélérer la convergence de la méthode BC, avec le partitionneur GRIP [126] prenant en compte les valeurs dans les équations normales, ou à diminuer la taille de l'augmentation, avec le partitionneur d'hyper-graphes PaToH PaToH [32]. Nous introduisons ensuite une nouvelle technique d'augmentation de la matrice qui permet de diminuer encore la taille d'augmentation.

D.3.1 Schéma de parallélisation

Les méthodes BC et ABCD sont implémentées en parallèle dans le ABCD-Solver, suivant le même schéma de parallélisation basé sur l'utilisation de processus MPI [71] (distribué) et de threads OpenMP [37] (mémoire partagée). Tout commence par la phase de pré-traitement séquentielle durant laquelle, en particulier, la matrice est partitionnée, éventuellement augmentée (dans ABCD), et un système est créé pour chaque partition afin de calculer la projection correspondante avec le solveur direct MUMPS. Le solveur suit alors un schéma de parallélisation classique en DDM: des processus MPI, appelés maîtres, reçoivent une ou plusieurs partitions et calculent en parallèle, de manière indépendante, les projections associées à l'aide d'un solveur direct. Ces projections locales sont alors sommées grâce à des communications MPI entre maîtres afin de mettre l'itéré courant à jour. Nous proposons un nouvel algorithme de distribution des partitions pour la méthode itérative qui permet, en plus d'équilibrer la charge de travail entre les processus, de diminuer la communication entre ceux-ci afin de diminuer le temps de calcul. En plus de l'indépendance entre les partitions, 2 niveaux de parallélisme sont ajoutés par le solveur direct utilisé pour les projections. Les noyau de calcul pour les opérations denses utilisés par le solveur direct ont une parallélisation en mémoire-partagée. De plus, chaque potentiel processus sans partition, appelé ouvrier, peut être assignée à un maître afin de paralléliser l'exécution du solveur direct. Nous montrons que distribuer les maîtres sur l'architecture de calcul, plutôt que de les grouper, permet de diminuer les accès mémoire concurrents et ainsi de réduire le temps d'exécution. Cette parallélisation est résumée dans la Figure D.16.

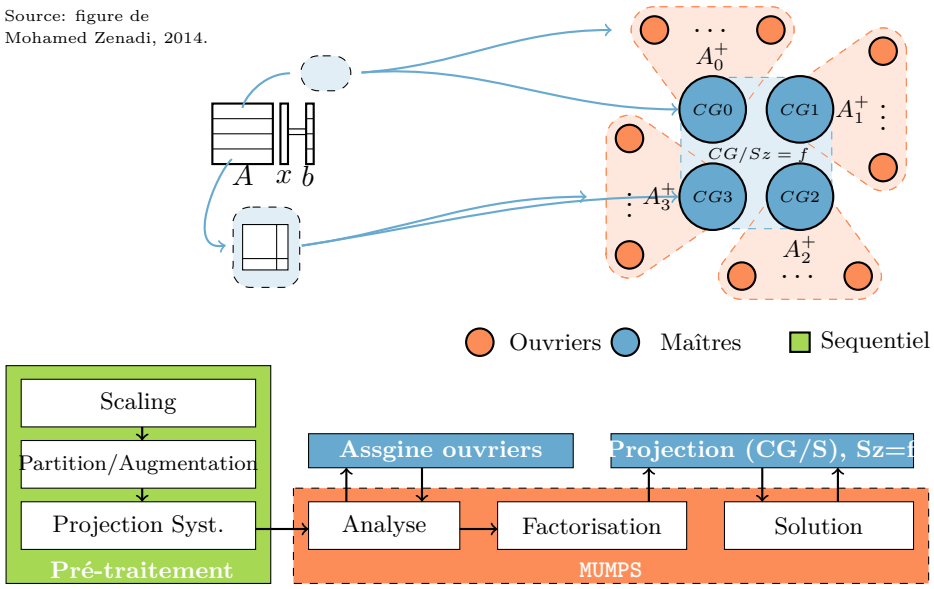


Figure D.16 ABCD-Solver: Schéma de parallélisation.

D.3.2 Efficacité parallèle

Enfin, la grande question est quelle est l'efficacité du ABCD-Solver, que ce soit la méthode BC ou la méthode ABCD ? Afin d'estimer cette efficacité, nous comparons notre approche avec des solveurs directs de l'état de l'art en tournant sur toutes les matrices de la SuiteSparse Matrix Collection de plus de 1000 lignes ou colonnes. Concernant les matrices carrées, nous effectuons une comparaison avec le solveur MUMPS[5] [6], qui utilise une décomposition LU et la méthode multifrontale dans une parallélisation hybride MPI-OpenMP. Nous effectuons alors des tests sur le cluster Meggie[6] avec 64 processus MPI et 4 OpenMP par MPI. Quant aux matrices rectangulaires, nous effectuons une comparaison avec QR-MUMPS [29], un solveur direct pour les matrices rectangulaires, basé sur la décomposition QR et la méthode multifrontale, avec une parallélisation OpenMP. Nous effectuons alors des tests sur le cluster Kraken [7] avec 18 OpenMP pour QR-MUMPS, et 9 MPI $\times$ 2 OpenMP pour le ABCD-Solver, afin de bénéficier tout de même du parallélisme hybride pour ce dernier. Les résultats de ces tests sont présentés dans la Figure D.17 dans 2 graphes indiquant la différence relative entre le solveur direct et le ABCD-Solver en terme de temps d'exécution (en abscisse), et de mémoire (en ordonnée). Une valeur positive donne l'avantage au ABCD-Solver. Ce que nous observons tout d'abord c'est

[5] `http://mumps-solver.org/`
[6] RRZE, FAU: `https://www.anleitungen.rrze.fau.de/hpc/meggie-cluster/`
[7] CERFACS: `https://cerfacs.fr/en/cerfacs-computer-resources/`

qu'en terme de temps d'exécution, les solveurs directs ont l'avantage dans la plupart des cas. En effet, seulement 17% des tests donnent l'avantage au ABCD-Solver. Pour ce qui est de la consommation en mémoire, ABCD-Solver a l'avantage en général: MUMPS a besoin de plus de mémoire dans 75% des cas, et QR-MUMPS dans 50% des cas.

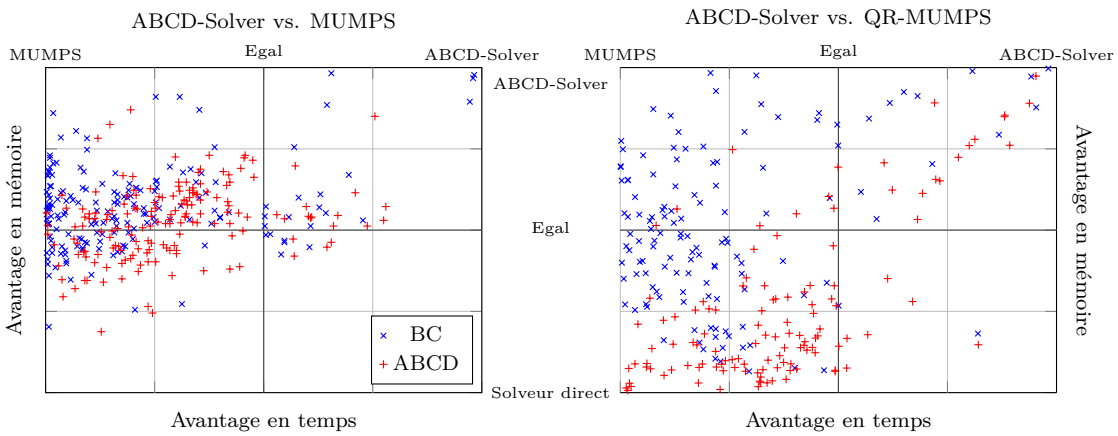


Figure D.17 Différence relative entre l'ABCD-Solver et *(Gauche)* MUMPS ou *(Droite)* QR-MUMPS en terme *(x-axis)* de temps d'exécution et *(y-axis)* de mémoire pour 187 et 128 classes de problèmes de la SuiteSparse Matrix Collection. Une valeur positive indique un avantage pour l'ABCD-Solver (temps ou mémoire plus petit).

Comme l'extension du ABCD-Solver à la solution de systèmes rectangulaires est la principale contribution des Chapter II et III, nous nous concentrons alors sur la comparaison avec QR-MUMPS. Dans la Figure D.18, les tests les plus larges (¿ 1s pour la résolution) sont mis en évidence avec le plus meilleur solveur dans chacun de ces cas. Quelles sont les conditions dans lesquelles chaque solveur est le plus rapide ? Prenons trois matrices typiques.

Dans le cas de la matrice `neos` (losange sur le graphe), la factorisation de QR-MUMPS est particulièrement coûteuse (196s et 2.2 GB). Pour la méthode itérative BC, chaque itération est rapide mais la convergence n'est obtenue qu'après 920 itérations pour 754s au total. Dans ce cas, la méthode ABCD est la meilleure car le complément de Schur est petit (10% de la taille originale) et, même si S reste un peu dense, la résolution du système associé est rapide et moins coûteuse en mémoire que QR-MUMPS (92s et 2GB).

Considérons maintenant la matrice `LargeRegFile` (triangle sur le graphe) qui est le complet opposé du cas précédent. Ici, ABCD donne un complément de Schur toujours petit (8% de la taille originale) mais si dense que la factorisation avec MUMPS requiert

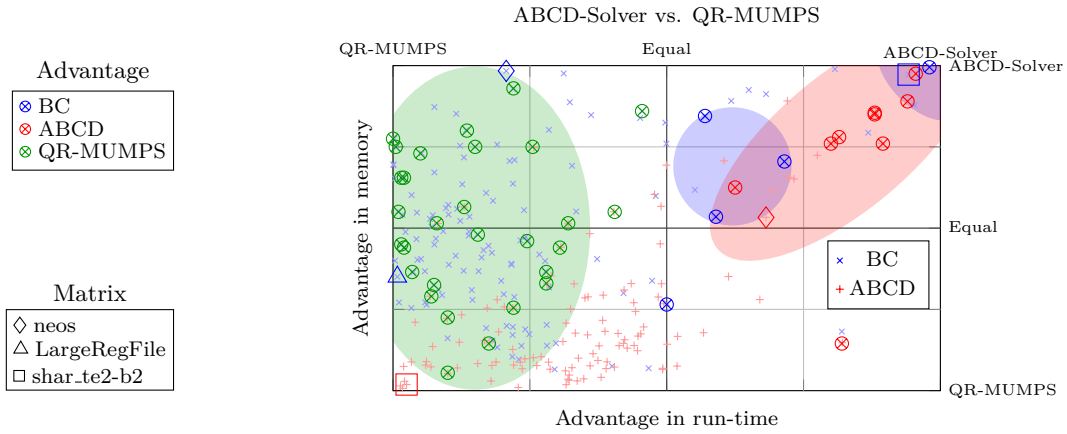


Figure D.18 Différence relative entre l'ABCD-Solver et QR-MUMPS en terme *(x-axis)* de temps d'exécution et *(y-axis)* de mémoire. Les plus grosses classes de problème (temps > 1s) pour lesquelles chaque solveur est meilleur sont indiquées, ainsi que 3 exemples spécifiques.

trop de mémoire pour la machine à notre disposition. Quant à la méthode BC, la convergence est très rapide avec seulement 28 itérations, mais chaque itération est très lente pour un total de 182s avec 153 MB. Dans ce cas, QR-MUMPS est imbattable puisque la solution est obtenue en seulement 6s pour une consommation mémoire de 81 MB (même moins que BC).

Pourquoi une telle différence ? Pour les deux matrices précédentes, nous montrons en Figure D.19 les équations normales après reordering en utilisant un simple algorithme SymAMD (Symmetric Approximate Minimum Degree). Pour la matrice `neos`, la structure obtenue est typiquement une matrice qui provoquerait une grosse utilisation mémoire pour un solveur direct (effet de fill-in), et une grosse complexité en terme de calcul (beaucoup de pivoting). Quant à la matrice `LargeRegFile`, nous obtenons une forme en tête de flèche parfaite, ce qui explique que QR-MUMPS est imbattable dans ce cas. Malheureusement, ABCD-Solver ne peut faire usage de telles méthodes de pré-traitement.

Finalement, nous considérons la matrice `shar_te2-b2`. Pour cette matrice, ABCD donne une taille d'augmentation très large ET très dense ce qui donne un temps d'exécution énorme (2 311s) et une très grande consommation mémoire (51 GB). Pour QR-MUMPS, la factorisation est très coûteuse (193s et 1 GB). Enfin, cette matrice est parfaite pour la méthode itérative BC pour qui la convergence est très rapide (11 itérations en 2.4s). Cette dernière méthode est alors imbattable dans ce cas, mais c'est loin d'être une généralité et n'arrive que dans des cas très spécifiques.

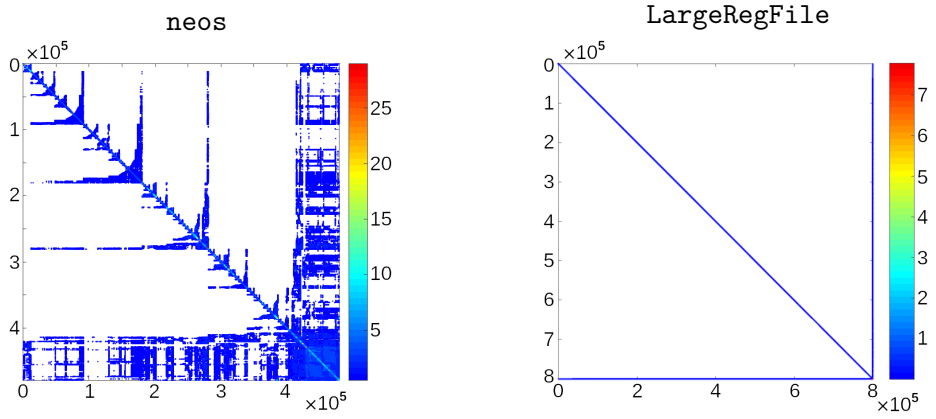


Figure D.19 Structure des équations normales après reordering grâce à l'algorithme SymAMD (Symmetric Approximate Minimum Degree).

Nous avons donc développé une méthode itérative et une méthode pseudo-directe basées sur un paradigme par bloc. La méthode ABCD nécessite que le complément de Schur calculé ait une taille raisonnable. Dans le chapitre suivant, nous proposons une méthode permettant de contrôler explicitement la taille de l'augmentation grâce à l'utilisation d'une grille grossière. Cependant, QR-MUMPS reste plus rapide dans la plupart des cas. Finalement, BC reste une bonne alternative avec une basse consommation en mémoire dans les cas où ABCD et QR-MUMPS seraient trop coûteux. Bien sûr, la convergence de BC reste problème dépendante.

D.4 Une approche pour le ABCD-Solver inspirée des méthodes multigrilles

Nous avons vu que la méthode ABCD peut être meilleure qu'un solveur direct compétitif mais que son efficacité dépend fortement du complément de Schur calculé. Considérons l'exemple très simple d'un problème de Poisson en 3D défini sur un cube. Le problème est discrétisé à l'aide de simples différences finies et on obtient un système linéaire de taille $N = 2^{12} = 4\,096 \approx 70 \cdot 10^9 DDLs$. Cette matrice est partitionné en utilisant la géométrie du système, i.e. nous divisons le cube en $l_p = 16$ blocs dans chaque direction. L'augmentation obtenue avec ABCD, qui correspond à une condensation de l'information sur l'interface entre les partitions, a alors une taille du même ordre que les faces entre les petits cubes. Sur chaque face, nous avons $f = \left(\frac{N}{l_p}\right)^2$ DDLs, et la taille d'augmentation

finale est

$$taille(Aug.) \approx \sum_{i=1}^{p} 6 \times f = 6p(\frac{N}{l_p})^2 = \frac{6N^3}{16^2} = \mathcal{O}(N^3)\ ! \tag{D.12}$$

C'est-à-dire que l'augmentation obtenu a une taille du même ordre que le nombre total de point dans le système original, ce qui n'est pas acceptable et nous avons besoin de réduire/contrôler cette taille.

D.4.1 Problèmes d'EDPs et multigrille

Nous considérons seulement des systèmes linéaires venant de la discrétisation de problèmes aux dérivées partielles (EDPs) définis sur un domaine carré ou sur un domaine en L. Sur ces domaines, une hiérarchie de 6 grilles imbriquées, de la plus grossière $l = 0$ à la plus fine $L = 5$, sont définies à partir d'une grille initiale, voir le haut de la Figure D.20. De plus, un partitionnement est défini en se basant sur la géométrie du domaine, voir le bas de la Figure D.20.

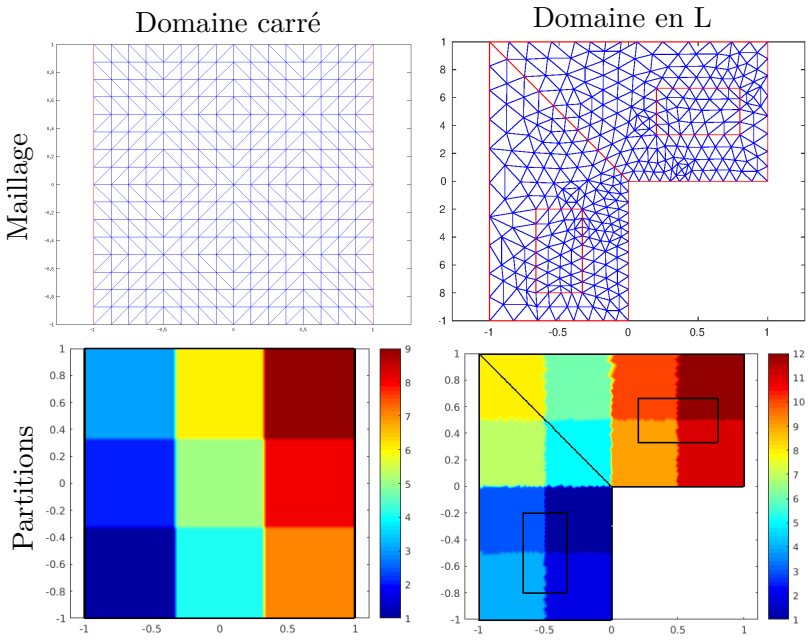


Figure D.20 Maillage et partitionnement des domaines carré et en L.

A partir de ces domaines et grilles, nous nous concentrons sur 3 problèmes, voir Figure D.21:

1. un problème de Helmholtz défini sur le domaine carré,
2. un problème de convection-diffusion sur le domaine carré, avec convection domi-

nante,

3. un problème de diffusion défini sur le domaine en L, avec diffusivité très hétérogène.

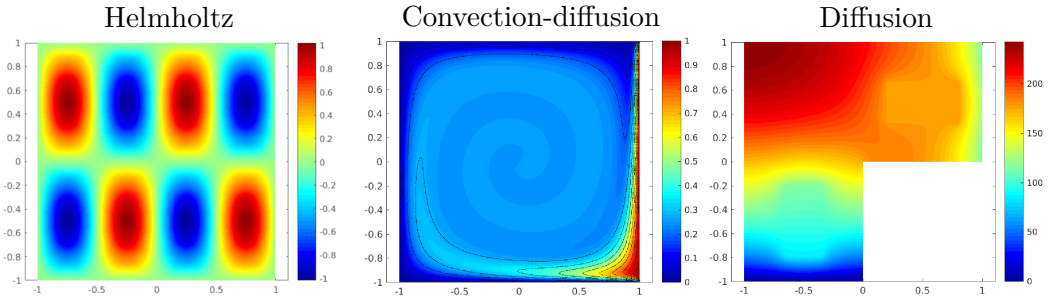


Figure D.21 Solution des problèmes d'EDPs.

Ces problèmes sont discrétisés à l'aide d'éléments finis P1 et des opérateurs de prolongations P_l^{l+1} sont définis entre 2 niveaux consécutifs l et $l+1$ avec une interpolation bilinéaire. Comme fait dans les méthodes multigrilles, nous avons alors à disposition une hiérarchie de systèmes linéaires et une idée naturelle est que le choix d'un niveau grossier pour l'augmentation permettrait de contrôler la taille du complément de Schur.

D.4.2 Augmentation relâchée et construction de la solution

Considérons une matrice carrée A inversible provenant de la discrétisation d'EDPs et partitionnée suivant la géométrie du système. Plutôt que d'applique la technique d'augmentation de ABCD directement sur A, nous l'appliquons sur la matrice $A.V$, où V est une matrice sur-déterminée avec peu de colonnes, afin d'obtenir la matrice augmentée:

$$\overline{A} = \begin{bmatrix} A & \mathcal{F}(AV) \end{bmatrix} \quad \text{avec} \quad V \in \mathbb{R}^{m \times m_c}, \quad m_c \ll m.$$

Dans cette matrice, les partitions augmentées ne seront pas strictement orthogonale. Le but est simplement d'obtenir une augmentation de petite taille et dans laquelle les angles entre les sous-espaces engendrés par les partitions sont plus ouverts que dans le système original afin d'avoir une convergence de la méthode Cimmino par bloc plus rapide. En réalité, l'orthogonalité strict est vrai dans une sous-image de A, i.e. pour la matrice $\begin{bmatrix} AV & \mathcal{F}(AV) \end{bmatrix}$.

Dans notre cas, un choix naturel pour la matrice V vient alors des méthodes multigrilles. En choisissant un niveau de grille grossière spécifique l et en considérant que nous résolvons le niveau fin L, nous pouvons définir l'opérateur de prolongation entre ces 2 niveaux comme le produit des prolongations intermédiaires, i.e. $P = P_{L-1}^{L} \dots P_l^{l+1}$. A noter également que si nous choisissons le niveau fin comme niveau grossier ($l = L$), alors

$P = I$ et nous récupérons la méthode ABCD classique. Pour les 3 problèmes considérés, nous donnons dans le Table D.4 la taille de P et la taille de l'augmentation $\mathcal{F}(AP)$ obtenues pour chaque choix de grille grossière possible. Nous observons que la taille de P est divisée par 4 en choisissant le niveau grossier consécutif, ce qui est normal car le problème est en 2D et le nombre de points est divisé par 4 entre 2 niveaux. Lorsque nous choisissons le prochain niveau grossier, l'augmentation est quant à elle divisée par 2. Une fois de plus ceci est attendu car l'augmentation a la même taille que les interfaces entre sous-domaines et l'interface est en 1D dans nos problèmes.

Table D.4 Size of the prolongator P and augmentation $\mathcal{F}(AP)$ depending on level

Grid levels	**Helmholtz**		**Convection-Diffusion**		**Diffusion**	
	P	$\mathcal{F}(AP)$	P	$\mathcal{F}(AP)$	P	$\mathcal{F}(AP)$
5 (ABCD)	87 424	9 329	65 025	4 112	65 025	4 096
4	21 952	4 766	16 129	2 072	16 129	2 064
3	5 536	2 477	3 969	1 048	3 969	1 044
2	1 408	1 339	961	536	961	536
1	364	764	225	280	225	280
0	97	471	49	152	49	152

Afin de calculer la solution du système, nous n'avons qu'à appliquer le reste de la méthode ABCD avec partitionnement par blocs de ligne. D'abord, le système est fermé avec des contraintes additionnelles $W = Y(I - \overline{P})$, orthogonales à $\overline{A}$, avec $Y = \begin{bmatrix} 0 & Y \end{bmatrix}$. Nous résolvons alors le système augmenté

$$\begin{bmatrix} \overline{A} \\ W \end{bmatrix} \begin{bmatrix} x^* \\ 0 \end{bmatrix} = \begin{bmatrix} A & \mathcal{F}(AP) \\ B & S \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ f \end{bmatrix} \tag{D.13}$$

Ensuite, nous pouvons choisir un second membre $f = -Y\overline{A}^{+}b$ tel que les variables additionnelles soient annulées. Nous obtenons donc comme dans ABCD la solution x^* du système original grâce à la solution du système augmenté

$$\begin{bmatrix} x^* \\ 0 \end{bmatrix} = \overline{A}^{+}b + W^{+}f. \tag{D.14}$$

D.4.3 Construction itérative de la solution

Tout est ainsi identique à la méthode ABCD... Sauf que les partitions ne sont plus mutuellement orthogonales. Ainsi nous ne pouvons plus calculer les deux parties de la

réponse comme précédemment:

1. $\overline{A}^{+}b \neq \sum_{i=1}^{p} \overline{A}_i^{+} b_i$ et pour calculer cette partie il est maintenant nécessaire de faire converger la méthode Cimmino par bloc complètement pour obtenir cette partie de la solution. La bonne nouvelle est que nous espérons une convergence linéaire et rapide grâce à l'augmentation.
2. $\mathcal{P}_{\mathcal{R}(\overline{A}^T)} \neq \sum_{i=1}^{p} \mathcal{P}_{\mathcal{R}(\overline{A}_i^T)}$ et pour calculer $W^{+}f$, l'idée est la même.

D.4.3.a Solution part $\overline{A}^{+}b$

Nous examinons maintenant la convergence de la méthode BC appliquée à la matrice augmentée $\overline{A}$ qui permet de calculer d'une part $\overline{A}^{+}b$, et d'autre part les projections sur l'image de $\overline{A}^T$. Nous nous attendons à observer une convergence linéaire et rapide puisque l'orthogonalité est vraie sur une sous-image de A correspondant au niveau grossier choisi, et l'espoir est que ce soit suffisant pour ouvrir les angles entre les sous-espaces engendrés par les partitions. Considérons maintenant les 3 problèmes d'EDPs introduit ci-dessus, et prenons leur discrétisation sur la grille fine $L = 5$ pour former le système $Ax = b$ où A est partitionnée en blocs de lignes suivant la géométrie du problème. Nous utilisons alors le niveau $l = 2$ comme niveau grossier afin d'appliquer l'augmentation relâchée, avec 4 niveaux de grilles dans ce cas. La Figure D.22 montre le profile de convergence de la méthode BC appliquée au système original, et au système augmenté $\overline{A}$. Nous prenons comme critère d'arrêt une erreur backward [9] de 10^{-10}. Sans augmentation, nous observons une convergence caractérisée par de longs plateaux et donc imprévisible. Au contraire, quand la matrice est augmentée sur le niveau grossier 2, la convergence devient linéaire et rapide. Dans le cas du problème de diffusion, la convergence sur A est obtenue après plus de 2 000 itérations, et est réduite à 126 itérations après augmentation avec 1 339 colonnes, soit 2% de la taille du système original. Cela signifie que les angles entre les partitions sont ouverts très rapidement grâce à l'information provenant de la grille grossière. Cela implique aussi que nous pouvons construire $\mathcal{P}_{\mathcal{R}(\overline{A}^T)}$ appliqué à n'importe quel vecteur avec une convergence rapide de la méthode itérative.

Bien sûr, nous nous attendons à ce qu'un choix de grille plus grossier apporte moins d'informations et une convergence plus lente. Un compromis est alors à trouver entre une petite taille d'augmentation et une convergence rapide. Dans les premiers tests (Figure D.22), nous avons utilisé le niveau de grille $l = 2$ pour vérifier la robustesse de notre approche. La Figure D.23 montre la taille d'augmentation obtenue, ainsi que le nombre d'itérations pour la convergence de BC appliqué sur $\overline{A}$, lorsque l'on varie le niveau de grille grossière choisi du plus grossier ($l = 0$) au plus fin ($l = 6$). Entre un

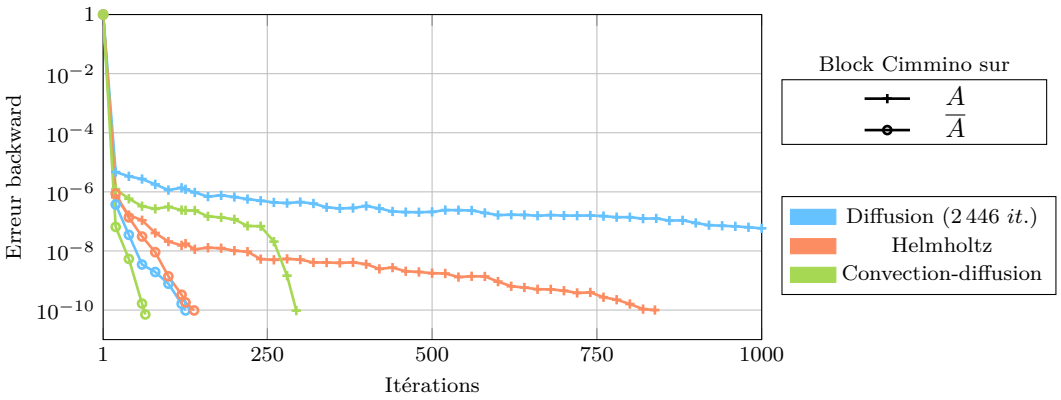


Figure D.22 Itérations de bloc Cimmino avec et sans augmentation, en utilisant le niveau $l = 2$ (4 niveaux)

niveau l et un niveau $l+1$, la taille d'augmentation est divisée par 2, comme les interface sont en 1D dans ces problèmes 2D, et le nombre d'itérations est augmenté par un facteur d'approximativement 2. Au travers du choix d'une grille plus ou moins grossière pour

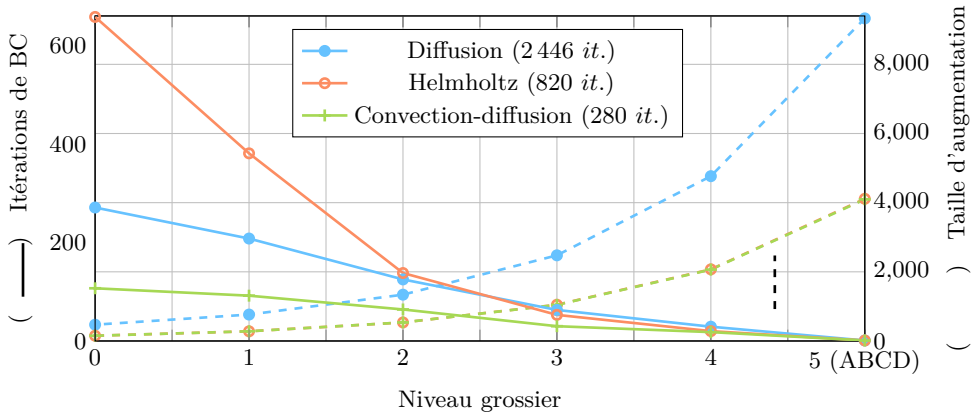


Figure D.23 Block Cimmino appliqué sur $\overline{A}$ avec différents choix de niveau grossier.

l'augmentation, nous introduisons une classe de méthodes avec convergence linéaire, à l'intermédiaire entre à un extrême la méthode ABCD ($l = L$), avec l'orthogonalité pure et la convergence en 1 seule itération, et à l'autre extrême la méthode BC, sans augmentation et avec une convergence en plateaux.

D.4.3.b Solution part $W^+ f$

Maintenant, il reste à s'occuper de la deuxième partie de la solution, dépendante des contraintes additionnelles W et du complément de Schur S. Comme dit précédemment,

les blocs W et S ne peuvent plus être construit avec une simple somme de projections comme dans ABCD, à cause de la perte d'orthogonalité entre les partitions. Nous devons approximer W et S itérativement. Ici, nous discutons deux approches pour cette construction itérative:

1. *Une approximation des contraintes additionnelles via Cimmino par bloc:* Rappelons que ces contraintes sont exprimées par

$$W^T = (I_{\overline{m}} - \overline{P})Y^T = Y^T - \overline{P}Y^T,$$

où $Y = \begin{bmatrix} 0 & I_q \end{bmatrix}$. La projection $\overline{P} = \overline{A}^+ \overline{A}$ peut être calculée par l'application de la méthode BC sur $\overline{A}$ qui, comme vu précédemment, a une convergence rapide et linéaire. Notons $Z = \overline{A}^+ \overline{A} Y^T$, la projection de Y^T sur $\mathcal{R}(\overline{A}^T)$, nous pouvons calculer Z via la solution de norme minimale du système sous-déterminé avec multiple second membres

$$\overline{A}Z = \overline{A}Y^T,$$

qui peut être obtenue avec la méthode BC à nouveau [51]. Même si la convergence de cette méthode sur $\overline{A}$ devrait être linéaire et rapide, il faut l'appliquer à autant de vecteurs que la taille d'augmentation. De même, et c'est l'inconvénient principal, l'approximation de S devient alors généralement très dense, comme cela résulte d'une méthode itérative qui mélange l'information graduellement. En considérant que la taille de S est raisonnable, le coût additionnel induit par la construction et la solution d'un système avec une telle matrice pourrait être compensé en considérant des problèmes dépendants du temps, avec plusieurs second membres successifs.

2. *Approximation implicite du complément de Schur:* Dans une seconde approche, nous appliquons implicitement l'inverse du complément de Schur en utilisant une méthode itérative appliquée à l'équation

$$Sz = f.$$

Pour résoudre cette équation avec une méthode de Krylov, e.g. un gradient conjugué préconditionné, il suffit de pouvoir appliquer S à chaque itération. Comme nous avons

$$S = Y(I_{\overline{m}} - \overline{P})Y^T = I_q - Y\overline{P}Y^T,$$

la difficulté principale pour appliquer S à un vecteur v est le calcul de la projection $\overline{P}Y^T v$ qui peut être calculée grâce à la méthode BC à chaque itération. Nous obtenons alors un schéma d'itérations internes-externes. Le point crucial est alors

la construction d'un bon préconditionneur pour S qui permettrait d'avoir une convergence rapide du solveur de Krylov. Le préconditionnement efficace de ce système est le sujet de nos recherches actuelles.

D.5 Conclusion

Ces travaux de thèse ne sont qu'un élément dans la course actuelle pour arriver à l'échelle du calcul exascale. Dans ce but, ma première contribution est l'apport d'une solution robuste et scalable pour la solution du problème sur la grille grossière à de très grandes échelles dans le framework multigrille HHG. Ceci n'est bien sûr qu'un élément que nous avons lié avec des techniques sans matrice assemblée et des méthodes de discrétisation flexibles. Ma deuxième contribution est l'apport de méthodes robustes pour des problèmes complexes, que même les méthodes multigrilles ne peuvent résoudre. Dans ce contexte, j'ai développé les méthodes itératives BC et pseudo-directe ABCD pour la solution des systèmes rectangulaires de très grande taille. Enfin, dans le cas où ces deux méthodes ne pourraient obtenir un résultat dans un temps et avec une consommation mémoire raisonnable, j'ai proposé une nouvelle méthode dans laquelle nous appliquons la méthode ABCD en considérant une représentation de basse résolution des interfaces entre sous-domaines. Cette augmentation relâchée permet d'obtenir une convergence rapide et linéaire de la méthode Cimmino par bloc. Une question très importante reste en suspend: quelle est l'interprétation en terme d'EDPs de cette nouvelle approche. Un début de piste serait de faire le lien entre cette méthode ABCD relâchée et des méthodes existantes de décomposition de domaine à 2 niveaux.

Abstract: In scientific computing, the numerical simulation of systems is crucial to get a deep understanding of the physics underlying real world applications. The models used in simulation are often based on partial differential equations (PDE) which, after fine discretisation, give rise to huge sparse systems of equations to solve. Historically, 2 classes of methods were designed for the solution of such systems: direct methods, robust but expensive in both computations and memory; and iterative methods, cheap but with a very problem-dependent convergence properties. In the context of high performance computing, hybrid direct-iterative methods were then introduced in order to combine the advantages of both methods, while using efficiently the increasingly large and fast supercomputing facilities. In this thesis, we focus on the latter type of methods with two complementary research axis.

In the first chapter, we detail the mechanisms behind the efficient implementation of multigrid methods. These methods use several levels of increasingly refined grids to solve linear systems with a combination of fine grid smoothing and coarse grid corrections. The efficient parallel implementation of such a scheme is a difficult task. We focus on the solution of the problem on the coarse grid whose scalability is often observed as limiting at very large scales. We propose an agglomeration technique to gather the data of the coarse grid problem on a subset of the computing resources in order to minimise the execution time of the overall scheme. Combined with an approximate direct solver, we demonstrate an increased overall scalability of the multigrid scheme when using our approach compared to classical iterative methods, when the problem is numerically difficult. At extreme scale, this study is carried in the HHG framework (Hierarchical Hybrid Grids) for the solution of a Stokes problem with jumping coefficients, inspired from Earth's mantle convection simulation. The direct solver used on the coarse grid is MUMPS, combined with block low-rank approximation and single precision arithmetic.

In the following chapters, we study some hybrid methods derived from the classical row-projection method block Cimmino, and interpreted as domain decomposition methods. These methods are based on the partitioning of the matrix into blocks of rows. Due to its known slow convergence, the original iterative scheme is accelerated with a stabilised block version of the conjugate gradient algorithm. While an optimal choice of block size improves the efficiency of this approach, the convergence stays problem dependent. An alternative solution is then introduced which enforces a convergence in one iteration by augmenting the linear system with additional variables and constraints. These two approaches are extended in order to compute the minimum norm solution of indefinite systems and the solution of least-squares problems. The latter problems require a partitioning in blocks of columns. We show how to improve the numerical properties of the iterative and pseudo-direct methods with scaling, partitioning and better augmentation methods. Both methods are implemented in the parallel solver ABCD-Solver (Augmented Block Cimmino Distributed solver) whose parallelisation we improve through a combination of load balancing and communication minimising techniques.

Finally, for the solution of discretised PDE problems, we propose a new approach which augments the linear system using a coarse representation of the space. The size of the augmentation is controlled by the choice of a more or less refined mesh. We obtain an iterative method with fast linear convergence demonstrated on Helmholtz and Convection-Diffusion problems. The central point of the approach is the iterative construction and solution of a Schur complement.

Keywords: high-performance computing, hybrid methods, multigrid methods, agglomeration, full rank linear systems, augmented block Cimmino