N^o D'ORDRE: 2133

THÈSE

présentée

pour obtenir

LE TITRE DE DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE

Spécialité: INFORMATIQUE

par

Stéphane Pralet

CERFACS

Ordonnancement sous contraintes et séquencement en algèbre linéaire creuse parallèle.

Constrained orderings and scheduling for parallel sparse linear algebra.

Thèse présentée le 21 septembre 2004 devant le jury composé de:

P. R. Amestoy	Professeur, ENSEEIHT	Directeur de thèse
I. S. Duff	Professeur, CERFACS et RAL	
J.-Y. L'Excellent	Chargé de recherche, INRIA	
E. Ng	Directeur de recherche, LBNL	Rapporteur
A. Pothen	Professeur, ODU	Rapporteur
J. Roman	Professeur, ENSEIRB	Président

Thèse préparée au CERFACS, CERFACS Report Ref: TH/PA/04/105

Résumé

Nous nous intéressons à la résolution de systèmes linéaires creux de grande taille par des solveurs creux directs opérant en trois phases qui sont l'analyse, la factorisation et la résolution.

L'analyse est le lieu de prétraitements et doit dans la mesure du possible assurer à la fois des facteurs aussi creux que possible et une factorisation numériquement stable. La factorisation doit exploiter l'indépendance des calculs pour être efficace dans un environnement parallèle distribué. Cette étude contribue à l'amélioration de ces comportements sur des classes de problèmes connues comme étant difficiles ou mal appréhendées par des stratégies classiques.

Dans une première partie, nous développons des techniques de prétraitements numériques et structurels pour les matrices symétriques indéfinies. Nous étudions aussi de manière plus prospective des approches de factorisation LDL^T avec pivotage statique et l'élaboration d'ordonnancements pour les systèmes augmentés.

Dans une deuxième partie, nous présentons des techniques d'ordonnancements pour les matrices très non symétriques visant à la fois à réduire le remplissage et à stabiliser la factorisation. Ces ordonnancements reposent sur des métriques hybrides prenant en compte des informations structurelles et numériques.

Dans une troisième partie, nous discutons des stratégies de séquencement des tâches dans un solveur multifrontal parallèle, MUMPS. Dans un premier temps, nous essayons de prendre en compte l'hétérogénéité des architectures des machines cibles. Dans un second temps, nous prenons en compte à la fois des critères de charge de travail et de mémoire pour une prise de décision dynamique optimale.

Mots-clés: calcul distribué, calcul parallèle, élimination de Gauss, matrices creuses, maximum matching, méthode multifrontale, ordonnancement, séquencement de tâches.

Abstract

We consider the three different phases (analysis, factorization, solution) for the direct solution of large sparse systems of linear equations.

During the analysis phase, preprocessing is applied in order to, on the one hand, permute the matrix to decrease the number of nonzeros in the factors and, on the other hand, to determine pivots that ensure as much as possible a stable factorization. During the factorization phase, the independence of the computations must be exploited to achieve a good performance on a distributed memory parallel computer. Our study contributes to the improvement of these aspects for some classes of matrices which are known for being difficult or for which default strategies clearly do not perform well.

In the first part of our thesis, we develop numerical and structural preprocessing strategies for symmetric indefinite matrices. We also study, in a more prospective way, static pivoting approaches for LDL^T factorizations and orderings for augmented systems.

In the second part, we present orderings for highly unsymmetric matrices that aim at decreasing the fill-in and at stabilizing the factorization. These orderings are based on hybrid metrics which take into account both structural and numerical information.

In the last part, we discuss task scheduling strategies in a parallel multifrontal solver, MUMPS. We study two kinds of strategies. Firstly, we investigate strategies that take into account heterogeneous architectures. Secondly, we study strategies that mix information about workload and memory to make optimal dynamic decisions.

Keywords: distributed computing, Gaussian elimination, maximum matching, multifrontal method, ordering, parallel computing, sparse matrices, tasks scheduling.

Remerciements

Merci Clarisse, tu m’as soutenu et fait preuve de beaucoup de patience pendant ces deux années, cette thèse est un peu la tienne.

Je tiens à remercier pour ce qu’ils m’ont apporté aussi bien sur le plan professionnel que sur le plan humain:

- Patrick Amestoy, mon directeur de thèse, professeur à l’ENSEEIHT, avec qui j’ai eu le plaisir de travailler. Je lui suis reconnaissant pour sa disponibilité, ses conseils et son ouverture d’esprit. Merci d’avoir toujours mis en avant les qualités de mon travail, ce qui m’a remotivé dans les moments de doute.
- Iain Duff, mon chef d’équipe, professeur au RAL, qui m’a fait bénéficier de ses connaissances et de son expérience. Tout en me laissant libre dans ma recherche, il a su s’impliquer dans mon travail. De plus, si l’anglais de cette thèse a des allures d’écossais, c’est grâce à lui.
- Esmond Ng, directeur de recherche au LBNL, et Alex Pothen, professeur à l’ODU, qui m’ont fait l’honneur d’être rapporteurs de ma thèse et de s’intéresser à mon travail. Leurs remarques pertinentes ont permis d’améliorer de manière significative la présentation de ma thèse.
- Jean Roman, professeur à l’ENSEIRB, qui m’a fait l’honneur de présider mon jury.
- Jean-Yves L’Excellent, chargé de recherche à l’INRIA, avec qui j’ai eu le plaisir de collaborer sur les aspects mumpsiens de ma thèse. Merci pour l’intérêt porté à mon travail et pour la lecture attentive de ma thèse.
- Sherry Li, chercheuse au LBNL, à qui j’ai pu rendre visite à Berkeley et avec qui j’ai eu la chance de collaborer.
- John Gilbert, professeur à l’UCSB, pour son accueil chaleureux à Santa Barbara et qui à travers des discussions informelles m’a apporté du recul sur mon travail, ce qui a fait progresser ma recherche.
- Serge Gratton, senior dans l’équipe ALGO, qui s’est toujours montré disponible pour d’innombrables discussions. J’ai pu profiter de la finesse de ses connaissances et de la pertinence de ses remarques.
- Abdou Guermouche, post-doctorant à l’INRIA, qui a grandement contribué au séquencement hybride.
- Toute l’équipe ALGO pour son ambiance de travail et plus généralement le CERFACS.

Contents

Introduction

We are interested in the direct solution of large sparse linear systems of equations, $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is large and sparse, $\mathbf{b}$ is the right-hand side and $\mathbf{x}$ is the vector of unknowns. A matrix is *sparse* if advantage can be taken by avoiding storage of or explicit computation on some of its zero entries. Clearly it does not only depend on the number of nonzeros but also on the pattern of the matrix and on the target machine.

Let us make the following conventions: $\mathbf{L}$ denotes a lower triangular matrix, $\mathbf{U}$ an upper triangular matrix and $\mathbf{D}$ a block diagonal matrix with blocks of size 1 or 2. Direct methods compute a Cholesky factorization if $\mathbf{A}$ is symmetric positive definite, an $\mathbf{LDL^T}$ factorization if $\mathbf{A}$ is symmetric indefinite, and an $\mathbf{LU}$ factorization in the unsymmetric case. Direct solvers try to preserve the zero pattern and to benefit from the independence of some computations in parallel environments. So called *three-phase approaches* have become very popular:

- The *analysis phase* applies numerical and/or structural pre-treatments and prepares for the operations of the factorization.
- The *factorization phase* tries to follow the decision of the analysis. Nevertheless, it must be able to dynamically adapt to unpredicted numerical difficulties and load variations of the experimental environment.
- The *solution phase* performs a forward and a backward substitution and optionally calls an iterative method to refine the solution.

In this introduction we discuss the analysis and the factorization phases in order to state the outline of our thesis. We try to emphasize some points at the end of each section that will guide us in the presentation of the different chapters. At the end of this introduction we describe briefly some graphs representations that are used in the context of sparse matrix computation and do a short introduction to multifrontal methods that will be useful for the late chapters.

Analysis phase

Numerical pre-treatments

The goal of the analysis phase is to prepare for the factorization. Some pre-treatments consider the values of the matrix. They aim at applying simple linear transformations to the original system so that the new one is more likely to be numerically friendly. A

first example of numerical preprocessing consists of applying *scaling* factors: diagonal matrices $\mathbf{D}_r$ and $\mathbf{D}_c$ are computed such that $\mathbf{D}_r\mathbf{A}\mathbf{D}_c$ has good numerical properties. Examples of scaling are proposed in [19, 24, 87, 93].

A second example of numerical preprocessing consists in permuting large entries to the diagonal [40, 41].

This last preprocessing has the drawback of restricting the choice of the pivots to the diagonal of the permuted matrix. Moreover it generally perturbs the symmetry and so is not well designed for symmetric matrices.

Structural pre-treatments

Other pre-treatments only consider the structure of the matrix. In most cases they consist of an ordering of the diagonal entries (ie they compute a symmetric permutation). Criteria for ordering the pivots depend on the objective for the factorization. Commonly the target is either to decrease the fill-in and/or the number of operations or to partition the pattern of the matrix into sets corresponding to balanced independent tasks (this last characteristic is necessary in parallel environments for the design of scalable factorizations).

Greedy algorithms work on local heuristics such as [4, 5, 12, 56, 85, 92] and have been developed successfully to improve the sparsity of the factors. More global decisions compute partitions using nested dissection algorithm [53]. Hybrid multilevel approaches [18, 90] have shown their efficiency. Examples of software for graph partitioning are CHACO [69], METIS [73], SCOTCH [88] and PORD [95]. They first apply multiple nested dissection and then use greedy algorithms on the smallest subgraphs.

Most of the orderings are done on a symmetric pattern. Recently, some orderings have been developed to take asymmetry into account [12]. Nevertheless, note that the pivots are still selected from the diagonal.

Factorization Phase

Numerical and static pivoting

The factorization follows as much as possible the estimations of the analysis. If some variables are not eliminated because of numerical issues, their elimination is delayed. It increases the fill-in in the factors but conserves the backward stability of the factorization in most cases. An alternative is to eliminate the pivots chosen by the analysis after adding small perturbations to the diagonal. In this last case, the factorization does *static pivoting* [77].

The static pivoting approach has been developed for unsymmetric matrices and must be coupled with unsymmetric preprocessings such as the maximum transversal. It has not been designed for symmetric indefinite matrices.

Parallel issues

Different approaches have been developed to schedule a sparse factorization in a parallel environment. Some direct solvers have chosen an "all static" approach. The tasks are mapped and the factorization cannot adapt itself to any variation of the load and has to perform static pivoting when numerical problems occur. It is the choice of PaStiX [70] and PSPACES [66]. On a dedicated machine, after being tuned, these approaches are efficient and highly scalable. Another approach performs a partial static mapping and still offers flexibility to adapt the workloads of each process and to perform numerical pivoting. It is the choice in MUMPS [7, 8]. These methods have the advantage of being more machine independent and it has been shown that they are competitive [9, 67]. In this last case many difficulties arise.

To address non-uniform memory access multiprocessors, algorithms in [11] for both the static and the dynamic scheduling need to be revisited to take account of the non-uniform cost of communication.

As the execution of the factorization is not fully determined, it is more difficult to evaluate the memory requirements and to correct the mistakes or the unbalances due to the variations of the load of the machine.

Problems addressed

We make links between different phases of sparse solvers. We emphasize the weaknesses of existing approaches on symmetric indefinite matrices and on structurally unsymmetric matrices and stress that parallel implementations need to be improved.

For the analysis phase, we can afford neither to have too dense factors nor to have bad numerical pivots. Usually this coexistence is maintained thanks to basic manipulations: a maximum weighted matching is computed so that the corresponding permutation will place large entries on the diagonal. If, however, this permutation is applied to a symmetric matrix the resulting permutation will not normally preserve symmetry. That is why in the first and in the second chapter of our thesis we consider ways of implementing preordering and scaling for symmetric systems and show the effect of using this technique with multifrontal codes for sparse symmetric indefinite systems.

Moreover in the case of very unsymmetric matrices it seems to us unreasonable to limit the ordering choices to the diagonal (ie to compute a symmetric permutation). That is why in the fourth chapter of our thesis we develop an approach that still preserves good numerical pivots and that has the freedom to select off-diagonal entries. By mixing the two objectives we show that we can reduce the amount of fill-in in the factors and reduce the amount of numerical problems during factorization.

For the factorization, we have mentioned that it must adapt dynamically to the numerical problems. Static pivoting has been developed for unsymmetric matrices but has not been studied for symmetric indefinite problems. The third chapter of our thesis starts this study.

In order to get better scalability, the fifth chapter revisits algorithms of parallel multifrontal factorization for both the static and the dynamic scheduling to address clusters of SMP (Symmetric Multi-Processor) architectures.

The purpose of the sixth chapter is to design a dynamic scheduling strategy that takes into account both workload and memory information. The originality of our approach is that we base our estimations (work and memory) on a static optimistic scenario during the analysis phase. We then use it during the factorization phase to constrain the dynamic decisions.

Sparse matrices, graphs and matchings

Graphs are often useful in the context sparse matrix computation. In this section we present graphs that can be associated with a sparse matrix $A = (a_{ij})$ of order n. Depending on the problem addressed, one representation may be more useful than another. We also define different kinds of matchings that are used in the first four chapters of the thesis.

Sparse matrices and graphs

The **directed graph associated with** A, $G = (V, E)$, is defined as follows. V represents the nodes (or vertices) and we often set $V = \{1, \ldots, n\}$. $E \subset V \times V$ is the set of edges. There is an edge (i, j) from node i to node j if and only if $a_{ij} \neq 0$.

If A is symmetric then G becomes the **undirected graph associated with** A: the edges are undirected since $(i, j) \in E$ implies $(j, i) \in E$.

The **bipartite graph associated with** A is defined as the undirected graph $G = (V_r, V_c, E)$ where V_r (resp. V_c) is the set of row (resp. column) vertices and $E \subset V_r \times V_c$ is the set of edges. There is an edge between the i^{th} row vertex of V_r and the j^{th} column vertex of V_c if and only if $a_{ij} \neq 0$.

For each type of graph the edges may be weighted. In such cases, we speak about **weighted graphs**.

Matchings

General graphs

Let $G = (V, E)$ be an undirected graph. Let $S \subset E$ be a set of edges. We have $S = \{(i_1, j_1), \ldots, (i_{|S|}, j_{|S|})\}$ where $|S|$ denotes the size of S, $i.e.$, the number of edges in S.

S is said to be a **matching** if and only if no two of its edges have the same vertex as an endpoint.

S is said to be a **maximal matching** if and only if for all edges $e \in E \setminus S$, $S \cup \{e\}$ is not a matching.

S is said to be a **maximum cardinality matching** if it is a matching of maximum size.

When the edges of G are weighted, a weight can be associated with S. Sometimes, we may be interested in the edge of minimum magnitude $\min_{(i,j)\in S} |a_{ij}|$ or in the sum of the magnitudes of the edges of S (that is $\sum_{(i,j)\in S} |a_{ij}|$). However, in the context of this thesis, we are interested in the product of the magnitudes and so we define the weight of S as

$$\omega(S) = \prod_{(i,j)\in S} |a_{ij}|.$$

S is said to be a **maximum weighted matching of maximum cardinality** if it maximizes the weight function ω among the set of maximum cardinality matchings.

In this thesis we will abbreviate the two above terminologies and we will use the terms **maximum matching** to refer to a maximum cardinality matching and **maximum weighted matching** to refer to a maximum weighted matching of maximum cardinality.

Bipartite graphs

When G is a bipartite graph, we will say that S is a **symmetric matching** if S is a matching and if for any edge $(i, j) \in S$ between the i^{th} row vertex and the j^{th} column vertex, the corresponding symmetric edge (j, i) between the j^{th} row vertex and the i^{th} column vertex is also in S.

The notion of **symmetric maximum matching** and **symmetric maximum weighted matching** are restrictions of the previous definition to symmetric matchings.

If G is a bipartite graph, we use the notation $S = I \otimes J$ where I is the sequence $(i_1, \ldots, i_{|S|})$ and J is the sequence $(j_1, \ldots, j_{|S|})$. I is said to be the set of matched rows and J the set of matched columns.

In this case, the size of S defines the **structural rank** of A. If the size of the maximum cardinality matching is lower than n then the matrix is said to be structurally singular. Note that structural singularity implies singularity since whatever are the values of the nonzeros entries of A at least one column is a linear combination of the others.

Use of matching techniques in the thesis

In the chapters about symmetric indefinite matrices, we will consider maximum weighted matchings and discuss algorithms to obtain a symmetric matching which is not too far (in terms of weight) from a symmetric maximum weighted matching.

In the fourth chapter, we consider unsymmetric matrices and their associated bipartite graph. We will be interested either in finding a maximum matching or in finding a maximum weighted matching.

Basic introduction to multifrontal methods

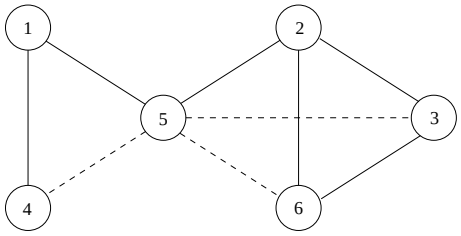
Symbolic factorization

We first consider the structurally symmetric case and suppose that the sequence of pivots is fixed. Usually, the analysis performs the symbolic factorization by manipulating undirected graphs. For example, the graph $G(\mathbf{A})$ associated with $\mathbf{A}$ has an edge between nodes i and j if $a_{ij} \neq 0$ (see Figures 1.a and 1.b). The symbolic factorization has to predict the fill-in and the dependence of the computations. It is often convenient in practice to work with $G(\mathbf{A})$. Excellent descriptions and data structures are given in [54].

Each time a pivot is selected its adjacent nodes which correspond to variables that are not yet eliminated are connected each other. They form what is called a *clique*. Note that an edge between two nodes implies that they cannot be eliminated independently. For example in Figure 1.b, after eliminating pivot 2, edges between nodes 5 and 6 and between nodes 3 and 5 are added. Then nodes 3, 5 and 6 form a clique and their elimination must be done sequentially.

$$\begin{pmatrix} X & & & X & X & \\ & X & X & & X & X \\ & X & X & & F & X \\ X & & & X & F & \\ X & X & F & F & X & F \\ & X & X & & F & X \end{pmatrix}$$

(1.a) Pattern of a structurally symmetric matrix and fill-in in its factors. An X corresponds to an original entry and a F to a fill-in in the factors.



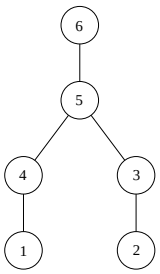
(1.b) Undirected graph of matrix of Figure 1.a and fill-in. Dashed edges correspond to fill-in in the factors.

Figure 1: Example of symbolic factorization on a structurally symmetric matrix.

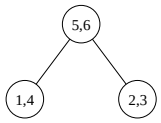
The dependences can be represented by a tree which is called the *elimination tree* [81] (see Figure 2.a). The elimination tree represents the order in which the matrix can be factorized, ie the order in which the unknowns from the underlying linear system of equations can be eliminated. For a dense matrix the elimination tree is a chain and defines a complete ordering of the eliminations. However, for a general sparse matrix, the definition yields only a *partial* ordering which allows some freedom for the order in which pivots can be eliminated. Indeed two variables none of which is the descendant of another can be eliminated at the same time.

The structurally unsymmetric case is clearly more complicated. The analysis has to predict the structure of $\mathbf{L}+\mathbf{U}$ (assuming that no numerical pivoting will occur during the factorization). A first solution is to work on the pattern of $\mathbf{A}+\mathbf{A^T}$ and so to get an upper bound on the structure of the factors. A second solution computes the exact structure of the factors. A directed graph or a bipartite graph is preferred and the elimination tree is generalized to an *elimination directed acyclic graph* (denoted as *edag*) [57, 58] which describes the dependence of the computations.

One central concept of the sparse computation is to group (or *amalgamate*) variables



(2.a) Elimination tree of $\mathbf{A}$.



(2.b) An assembly tree compatible with the elimination tree of $\mathbf{A}$.

Figure 2: Examples of trees.

with the same or nearly the same sparsity structure to create bigger *supervariables* or *supernodes* [44, 83] in order to make use of efficient dense matrix kernels. The amalgamated elimination tree is called the *assembly tree* (see Figure 2.b) and the amalgamated *edag* is called the *assembly directed acyclic graph*.

Finally, the symbolic factorization evaluates the storage requirements and prepares the integer and real data structures for the factorization.

Factorization Phase

The factorization follows as much as possible the estimation of the analysis. We briefly describe the organization of supernodal approaches and then we discuss multifrontal approaches.

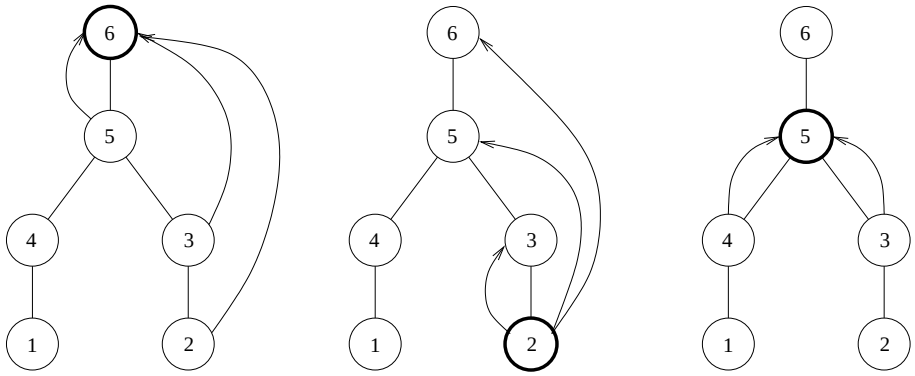


Figure 3: Updates in left-looking, right-looking and multifrontal factorization. The bold nodes represent the current pivot and the arrows symbolize the update. In the left-looking approach, column 6 is updated by the contribution of nodes 2, 3 and 5 before the elimination using pivot 6. In the right-looking approach, the pivot 2 is eliminated and then updates of columns 3, 5 and 6 are done. In the multifrontal approach, the frontal matrix of node 5 is built including the contributions of children 3 and 4, then elimination is performed and the Schur complement is computed.

The scheme in Figure 3 shows the main differences between the left- or right-looking supernodal approaches and the multifrontal approaches. In a *left-looking* approach, the

columns of the pivots are first updated by the contributions from the previous eliminations. Then the pivots are selected and their columns of factors are computed.

In a *right-looking approach* the pivots are first selected, their columns are then computed and finally the entries that are affected by these eliminations are updated.

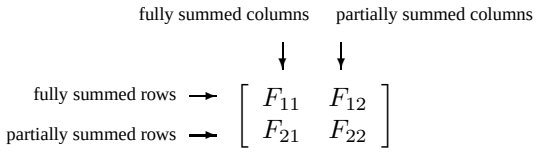
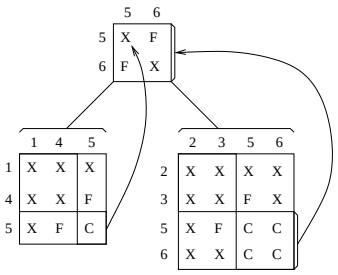


Figure 4: A frontal matrix.

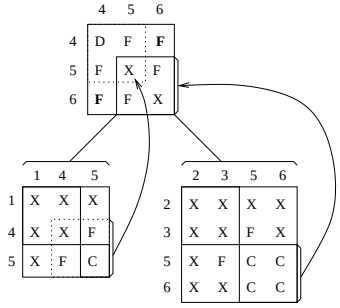
The *multifrontal* method was initially developed for indefinite sparse symmetric linear systems [44] and the extension to unsymmetric matrices [45] followed.

The work associated with an individual node of the assembly tree corresponds to the factorization of a so called *frontal matrix*. Frontal matrices are always considered as dense matrices and we can make use of efficient BLAS kernels and avoid indirect addressing, see for example [32].

Here, pivots can be chosen only from within the block of fully summed variables F_{11}. Once all eliminations have been performed, the Schur complement matrix $F_{22} - F_{21}F_{11}^{-1}F_{12}$ is computed and used to update later rows and columns of the overall matrix which are associated with the parent nodes. We call this Schur complement matrix the *contribution block* of the node. If some variables are not eliminated because of numerical issues, they are included in the contribution block and their elimination is postponed to the parent node or latter. These *delayed pivots* increase the fill-in in the factors, the number of operations and the factorization time (see Figure 5) but allow the factorization to compute accurate factors.



(5.a) No delayed pivots



(5.b) Pivot 4 delayed because of numerical issue. Bold F represent the fi ll-in due to delayed pivot (marked with a D).

Figure 5: Assembly tree and frontal matrices that appear during the factorization of **A**. Each X shows the place where the initial entries are assembled. An F corresponds to fi ll-in and a C corresponds to entries in the contribution blocks.

The notion of child nodes which send their contribution blocks to their parents leads to the following interpretation of the factorization process. When a node of the assembly tree is being processed, it assembles the contribution blocks from all its child nodes into its frontal matrix. Afterwards, the pivotal variables from the fully summed block are

eliminated and the contribution block computed. The contribution block is then sent to the parent node to be assembled once all children of the parent (which are the siblings of the current node) have been processed.

More recently the multifrontal factorization has been adapted in different ways to unsymmetric patterns. The work of [46] proposes a generalization of the elimination tree for symmetric indefinite matrices. This approach can be adapted in a straightforward way to unsymmetric matrices by introducing the augmented system associated with $\mathbf{A}$. Nevertheless, it has not yet proved its efficiency on a large range of matrices. The factorization of [13] allows the fronts to be rectangular and provides algorithms for generating them. It saves significant storage and improves performance over the symmetric version.

I

Chapter 1

Strategies for scaling and pivoting for sparse symmetric indefi nite problems

We consider ways of implementing preordering and scaling for symmetric systems and show the effect of using this technique with a multifrontal code for sparse symmetric indefi nite systems. After presenting a new method for scaling, we propose a way of using an approximation to a symmetric weighted matching to predefi ne 1×1 and 2×2 pivots prior to the ordering and analysis phase. We also present new classes of orderings called “(relaxed) constrained orderings" that mix structural and numerical criteria.

1.1 Introduction

We study techniques for scaling and choosing pivots when computing the LDL^T factorization of symmetric indefinite matrices where L is a lower triangular matrix and D is a block diagonal matrix with 1×1 and 2×2 blocks.

Our main contribution is to define a new method for scaling and a way of using an approximation to a symmetric weighted matching to predefine 1×1 and 2×2 pivots prior to the ordering and analysis phase.

We also present new classes of orderings called “(relaxed) constrained orderings" that select pivots during the symbolic Gaussian elimination using two graphs: the first one contains information about the structure of the reduced matrix and the second one gives information about the numerical values.

Prior to the LU factorization of an unsymmetric matrix A, `MC64` [40, 41] can be used to get a maximum weighted matching so that the corresponding permutation will place large entries on the diagonal. The matrix can then be scaled so that diagonal entries have modulus one and off-diagonals have modulus less than or equal to one. This has been found to greatly improve the numerical stability of the subsequent LU factorization. If, however, `MC64` is applied to a symmetric matrix the resulting permutation will not normally preserve symmetry. In this chapter, we examine ways in which symmetric preprocessing can be applied and in particular how `MC64` can be used while still preserving symmetry. Our preprocessing will be followed by a symmetric permutation in order to decrease the fill-in in the factors.

We will use our symmetric preprocessing with a symmetric multifrontal code `MA57` [37]

to validate our heuristics on real test problems that we have divided into two sets: the first consists of augmented matrices and the second contains general indefinite matrices.

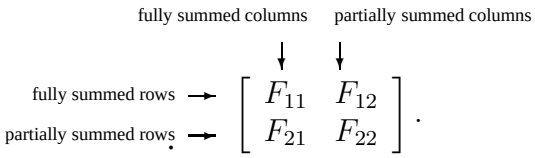
In Section 1.2, we present some general characteristics of multifrontal symmetric indefinite solvers that will be useful in understanding our preprocessing strategies and experimental results. In Section 1.3, we describe our experimental environment. Section 1.4 shows how `MC64` scaling can be used for the symmetric indefinite case and studies the effects on the factorization of different kinds of scaling. Section 1.5 discusses orderings to decrease the fill-in and to give good preselected pivots. Our experimental results are given in Section 1.6 and the best strategies are discussed. Finally, in Section 1.7, we summarize our results and consider future improvements.

We make extensive use of routines from HSL [72] (see Table 1.3.4). Any code with a name beginning with `MA` or `MC` is from HSL or is a derivative of an HSL code, for example `MC64` or `MA57`.

1.2 Symmetric indefinite multifrontal solvers and numerical pivoting

1.2.1 Multifrontal approach

Multifrontal methods [44, 45] use an elimination tree [81] to represent the dependencies of the computation. Each node of this tree is associated with a frontal matrix that is assembled (summed) by contributions from its children and the original matrix. It is of the form:



Then elimination operations are performed using pivots from within the fully summed block, F_{11}, and the Schur complement or contribution block $F_{22} \leftarrow F_{22} - F_{21}F_{11}^{-1}F_{12}$ is computed. In the symmetric case, $F_{12} = F_{21}^T$, the matrices F_{11} and F_{22} are symmetric, pivots are chosen from the diagonal as discussed in the following section, and operations and storage are about half that of the general case.

1.2.2 Numerical pivoting

In the unsymmetric case, at step k of the Gaussian elimination, the pivot (p, q) is selected from the fully summed rows and columns and the entries a_{ij} of the remaining submatrix are updated:

$$a_{ij}^{(k+1)} \leftarrow a_{ij}^{(k)} - \frac{a_{ip}^{(k)} a_{qj}^{(k)}}{a_{pq}^{(k)}}.$$

To limit the growth of the entries in the factors and thus to have a more accurate factorization, a test on the magnitude of the pivot is commonly used. a_{pq} can be selected

if and only if

$$|a_{pq}| \geq u \max_j |a_{pj}| \tag{1.2.1}$$

where u is a threshold between 0 and 1. Thanks to this criterion of selection the growth factor is limited to $1/u$.

In the symmetric indefinite case, we have to perform 1×1 and 2×2 pivoting if we want to keep the symmetry while maintaining stability. Pivot selection can be done using the Bunch-Parlett [21] or Bunch-Kaufman [20] algorithm or a variation proposed by [17]. In the context of sparse matrices, the criterion of the Duff-Reid algorithm ([44], as modified in [46]) can be used to ensure a growth factor lower than $1/u$ at each step of Gaussian elimination. A 1×1 diagonal pivot can be selected if and only if it satisfies the inequality (1.2.1). A 2×2 pivot $P = \begin{pmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{pmatrix}$ can be selected if and only if it satisfies:

$$|P^{-1}| \begin{pmatrix} \max_{k\neq p,q} |a_{pk}| \\ \max_{k\neq p,q} |a_{qk}| \end{pmatrix} \leq \begin{pmatrix} 1/u \\ 1/u \end{pmatrix} \tag{1.2.2}$$

where $|P^{-1}|$ denotes the matrix whose values are the absolute values of P^{-1} and u is a threshold between 0 and $\frac{1}{2}$ (the limit $\frac{1}{2}$ is needed to be sure that a pivot is available). During the elimination, it may not be possible to eliminate some fully summed variables. Elimination of these variables must then be delayed to the parent. This has the effect of causing extra fill-in and thus increases the memory and the number of operations. Too many delayed pivots can severely slow down the factorization.

1.3 Experimental environment

In this section, we present our experimental environment: test machine, sets of matrices and criteria of comparison. It will also be used in Chapters 2 and 3.

1.3.1 Test machine

Our experiments are conducted on one node of a COMPAQ Alpha Server SC45 at CERFACS. There are 4 GBytes of memory shared among 4 EV68 processors per node and we disable three of the processors so that we can use all the memory of the node with the remaining single processor. We use the Fortran 90 compiler, f90 version 5.5 with -O option. Our use of integer pointers restricts our array size to 2 GBytes. If the factorization needs to allocate an array larger than 2 GBytes or requires more than 30 minutes, we consider that the factorization is not successful.

1.3.2 Matrices

We conduct our experiments on a number of test problems that we divide into two sets. We decided to test our preprocessings on matrices of order between 10000 and 100000, and to restrict the number of matrices of the same type to 2 or 3 in each set in order to avoid class effects. Some matrices have an identification

number that will be used to represent them on the x-axis of some figures. Except for bloweybl, bloweybq and qpband that come from CUTEr, they are available from `ftp.numerical.rl.ac.uk/pub/matrices/symmetric/indef/` and correspond to a subset of the matrices collected by [61] for testing symmetric sparse solvers. The matrices come from the University of Florida collection (UF) [25], the Maros and Meszanos quadratic programming collection (M2) [84], the CUTEr optimization test set (CUTEr) [60] and from Kumfert and Pothen (KP) [75]. Some problems were generated by Andy Wathen (AW), Mario Arioli (MA), and Miroslav Tuma (MT). The c-* matrices were obtained from Olaf Schenk (OS) and are also available in [25]. These problems are described in Tables 1.3.1, 1.3.2 and 1.3.3.

Matrix	n	nnz	sprank	Origin
AUG2DC	30200	40000	20000	Expanded system–2D PDE (CUTEr)
AUG2D	29008	38416	19208	Expanded system–2D PDE (CUTEr)
AUG3D	24300	34992	11664	Expanded system–3D PDE (CUTEr)
DTOC	24993	34986	19994	Discrete-time optimal control (CUTEr)

Table 1.3.1: Augmented systems, $H = 0$ type (set 1).

Matrix	Id	n	nnz	Origin
A0NSDSIL	1	80016	200021	Linear Complementarity problem (CUTEr)
A2NNSNSL	2	80016	196115	Linear Complementarity problem (CUTEr)
A5ESINDL	3	60008	145004	Linear Complementarity problem (CUTEr)
AUG3DCQP	4	35543	77829	Expanded system–3D PDE (CUTEr)
BLOCKQP1	5	60012	340022	QP with block structure (CUTEr)
BLOWEYA	6	30004	90006	Cahn-Hilliard problem (CUTEr)
BRAINPC2	7	27607	96601	Biological model (CUTEr)
BRATU3D	8	27792	88627	3D Bratu problem (CUTEr)
CONT-201	9	80595	239596	KKT matrix–Convex QP (M2)
K1_SAN	10	67759 (1)	303364	Straz pod Ralskem mine model (MT)
NCVXQP1	11	12111	40537	KKT matrix–nonconvex QP (CUTEr)
NCVXQP5	12	62500	237483	KKT matrix–nonconvex QP (CUTEr)
NCVXQP7	13	87500	312481	KKT matrix–nonconvex QP (CUTEr)
SIT100	14	10262	34094	Straz pod Ralskem mine model (MT)
bloweybl	15	30003 (1)	60000	Cahn-Hilliard problem (CUTEr)
cvxqp3	16	17500	62481	Convex QP (CUTEr)
mario001	17	38434	114643	Stokes equation (MA)
olesnik0	18	88263	402623	Straz pod Ralskem mine model (MT)
qpband	19	20000	30000	QP (CUTEr)
stokes128	20	49666	295938	Stokes equation (MA)
stokes64	21	12546	74242	Stokes equation (AW)
tuma1	22	22967	50560	Mine model (MT)
tuma2	23	12992	28440	Mine model (MT)

Table 1.3.2: Augmented matrices of $H \neq 0$ type (set 2). (x): x is the structural deficiency of the matrix. Id: identification number.

Many of the test matrices correspond to augmented matrices of the form

$$\mathcal{K}_{H,A} = \begin{pmatrix} H & A \\ A^T & 0 \end{pmatrix}.$$

Matrix	Id	n	nnz	Origin
BOYD1*	24	93279	652246	KKT matrix–Convex QP (CUTEr)
DIXMAANL	25	60000	179999	Dixon-Maany optimization example (CUTEr)
HELM3D01	26	32226	230335	Helmholtz problem (MA)
LINVERSE	27	11999	53988	Matrix inverse approximation (CUTEr)
SPMSRTLS	28	29995	129971	Sparse matrix square root (CUTEr)
bcsstk35	29	30237	740200	Stiffness matrix–automobile seat frame (UF)
bcsstk39	30	46772	1068033	Shuttle solid rocket booster (UF)
bloweybq	31	10001	30000	Cahn-Hilliard problem (CUTEr)
c-68	32	64810	315403	Optimization model (OS)
c-71	33	76638	468079	Optimization model (OS)
copter2	34	55476	407714	Helicopter rota blade (KP)
crystk02	35	13965	491274	Stiffness matrix–crystal free vibration (UF)
crystk03	36	24696	887937	Stiffness matrix–crystal free vibration (UF)
dawson5	37	51537	531157	Aeroplane actuator system (UF)
qa8fk	38	66127	863353	FE matrix from 3D acoustics (UF)
vibrobox	39	12328	177578	Vibroacoustic problem (UF)

Table 1.3.3: General symmetric indefinite matrices (set 3). $*$: BOYD1 is in set 3 because the number of constraints is negligible. Id: identification number.

The set of matrices described in Table 1.3.1 are augmented systems with $H = 0$. We note that the matrices of this form are structurally singular unless A is square nonsingular. The set of matrices described in Table 1.3.2 are also of the above form but with $H \neq 0$. The third set, in Table 1.3.3, corresponds to general (nonzero diagonal) indefinite symmetric matrices. Note that we also include the BOYD1 matrix which is an augmented system in this set because its number of constraints (order of the zero block) is negligible.

1.3.3 Measures and methodology

Table 1.3.4 summarizes the characteristics of routines from HSL [72] that we use in the first three chapters of the thesis.

Code	Description
`MA47`	solves a sparse symmetric indefinite system, taking advantage of the extra sparsity available with 2×2 pivots with one or both diagonal entries of value zero (multifrontal method).
`MA57`	solves a sparse symmetric indefinite system (multifrontal method).
`MA67`	solves a sparse symmetric indefinite system, taking advantage of the extra sparsity available with 2×2 pivots with one or both diagonal entries of value zero (blocked conventional).
`MC21`	permutes a sparse matrix to put entries on the diagonal.
`MC30`	calculate scaling factors of a sparse symmetric matrix so that the scaled matrix has its entries near to unity in absolute value.
`MC64`	permutes a sparse matrix to put large entries on the diagonal, with an option to compute scaling factors.
`MC77`	calculates scaling factors of a sparse matrix so that the p-norms of all the rows and columns is approximately equal to 1.

Table 1.3.4: Description of some HSL routines.

In our earlier experiments [33], we clearly identified that MA57 was not tailored to augmented matrices with a zero $(1,1)$ block. These were much better handled by MA47 [48] which respects the two zero blocks that exist when 2×2 pivots with two zero entries on the diagonal (referred to as *oxo* pivots) are eliminated. A brief review of these results is done in Section 2.5.1. This is why we have decided not to try to improve MA57 on this set of matrices and why we will not use matrices of this type in the experiments of this chapter. Authors of [61] also reported some failures of MA57 on this class of problems. They found that MA47 and MA67 usually performed better on these problems. We ran the MA57 factorization with a pivoting threshold equal to 10^{-2} and oblige its analysis to merge a father and its child if both require less than 16 eliminations (ICNTL(12) = 16).

We compare our codes using the performance profiling system of [29]. These profiles aim at evaluating and comparing the performance of a set of solvers $\mathcal{S}$ on a test set of problems $\mathcal{P}$. For each solver $i\in\mathcal{S}$ and each problem $j\in\mathcal{P}$, we measure a statistic s_{ij} with the conventions that if the solver did not succeed in solving the problem then $s_{ij}=\infty$ and that the smaller is this statistic, the better is the code. For example, we can measure the factorization time, the number of nonzeros in the factors ... Let j be a problem and let $\hat{s}_j=\min_{i\in\mathcal{S}}\{s_{ij}\}$ be the best statistic obtained for this problem. For each $\alpha\geq 1$ and each solver $i\in\mathcal{S}$, we define

$$p_i(\alpha)=\frac{\text{size }\{j\in\mathcal{P}\text{ such that }s_{ij}\leq\alpha\hat{s}_j\}}{\text{size }\mathcal{P}},$$

as the fraction of the problem set that is solved by the solver i within a factor α of the best. $p_i(1)$ gives the fraction of examples for which solver i is the best and $\lim_{\alpha\to\infty}p_i(\alpha)$ gives the fraction of the problems for which the solver succeeded. In our experiments, for each solver i we will plot its performance profile p_i against α.

To compare our different approaches we will look at the factorization time, the memory needed to complete the factorization, and the reliability of the analysis in terms of memory forecasting. We would like to have an estimation of the required memory after the analysis that is not too far from that actually needed by the factorization. The quality of the analysis prediction will be assessed using the ratio between the memory actually used during the factorization and that predicted by the analysis.

It is standard practice to increase the storage estimated by the analysis and allocate rather more storage in the hope that the factorization will be successful even if some additional numerical pivoting occurs. The percentage by which we increase the estimate from the analysis will be called the memory relaxation. Clearly, this memory relaxation parameter is important for algorithms that are implemented in Fortran. However, although C or C++ implementations allow dynamic memory allocation their cost may be not negligible. Finally, note that the quality of the memory estimation is all the more critical in a distributed memory environment. For example, the buffers used for communications need to be well estimated. In our experiments, we use a memory relaxation of 20% and 50%. When we evaluate the analysis prediction, we only consider runs where the factorization can be performed within the relaxed analysis estimation. When we want to evaluate the factorization time without taking into account the reliability of the memory estimations, we allocate 4 GBytes of memory.

We will look at the precision of the solution and the number of iterative steps only when we will do static pivoting in Chapter 3.

Firstly, in Section 1.4, we study the influence of scaling on the MA57 factorization phase. Secondly, the effect of additional numerical and structural preprocessing are analysed when they are combined with different orderings (AMD [4] in 1.6.2.1, METIS [73] in 1.6.2.2 and AMF [85, 92] in 1.6.2.3). The best approaches will be discussed in Sections 1.6.1 and 1.6.2.4.

1.4 Scaling

In this section, we examine the effect of the scaling and identify the most robust approach. Discussions about scaling can be found in [38]. Firstly, we describe existing scaling approaches; secondly, we propose an alternative based on the symmetrization of an unsymmetric scaling; and finally we analyse our experimental results.

1.4.1 Existing symmetric scaling

Let A be a square symmetric sparse matrix. MC30 [24] scales A. By computing a diagonal matrix D so that the scaled matrix $B = DAD$ has its nonzero entries near to 1 in absolute value. In practice, it computes the solution of a linear least-squares problem that minimizes $\sum_{b_{ij}\neq 0}(\log|b_{ij}|)^2$.

If A is structurally nonsingular, MC77 [93] used with the p-norm, $||\ ||_p$, computes a sequence of matrices $D_r^{(k)}AD_c^{(k)}$ that converge to a matrix that has its column and row norms equal to 1 (which corresponds to a doubly stochastic matrix if $p = 1$: for each row (resp. column) the sum of the absolute values of the entries in this row (resp. column) is equal to 1). At each step of its iterative process, MC77 algorithm divides the entry in position (i, j) by $\sqrt{||r_i||_p\,||c_j||_p}$ where r_i (resp. c_j) represents the vector of the i^{th} row (resp. j^{th} column). If $p \neq \infty$, this limit is unique. If A is symmetric then this scaling is symmetric ($D_r^{(k)} = D_c^{(k)}$). We will study the effect of MC77 with $p = 1$ and $p = \infty$. These scalings will be identified by MC77one and MC77inf, respectively. Note that, when the matrix is structurally singular, the convergence of the algorithm is not guaranteed. Moreover, the convergence to a doubly stochastic matrix with $p = 1$ is slower than with $p = \infty$. In our experiments, we limit the number of MC77 steps to 20.

1.4.2 Adaptation of MC64 scaling

1.4.2.1 Maximum weighted matching

It is common to represent a symmetric matrix A of order n by a weighted undirected graph given by $G = (V, E)$, where $V = \{1, 2, ...n\}$ and each undirected edge (i, j) of E corresponds to the off-diagonal nonzeros a_{ij} and a_{ji} of A and has a weight of $2|a_{ij}|$ if $i \neq j$. If $i = j$, depending on the context we may or may not want to add a self-loop to G of weight $|a_{ii}|$. The matrix can also be represented by a bipartite graph $G = (R, C, E)$ where $(i, j) \in R \times C$ belongs to E if and only if $a_{ij} \neq 0$ and has a

weight of $|a_{ij}|$. In the following we will associate a weight ω with a set of edges $\mathcal{S}$, defined by:

$$\omega(\mathcal{S}) = \prod_{(i,j)\in\mathcal{S}} |e_{ij}|, \text{ where } |e_{ij}| \text{ is the weight of the edge } (i,j). \tag{1.4.1}$$

We define a **matching** on the bipartite graph as a set of edges, no two of which have the same vertex as an endpoint. A **maximum matching** is a matching with maximum cardinality that will be n if the matrix is structurally nonsingular. A **maximum weighted matching** is a maximum matching that realizes the maximum weight. We will later (in Section 1.5.1) relate matchings on the bipartite graph to matchings on another undirected graph that is a modified version of the undirected graph described above.

1.4.2.2 Symmetrization

When `MC64` is used with a product metric to define the weight of a set of edges, it returns a maximum weighted matching and thus provides a permutation that puts large entries on the diagonal of the matrix. It also returns a row and column scaling. This scaling is directly computed from the dual variables that are used during the maximum weighted matching algorithm. We now show how a symmetric scaling can be built from the `MC64` scaling. It will be called `MC64SYM`.

DEFINITION 4.1. *A matrix $B = (b_{ij})$ is said to satisfy the* `MC64` *constraints if and only if*

$$\exists \text{ a permutation } \sigma, \text{ such that } \forall i, |b_{i\sigma(i)}| = 1 \text{ and } \forall (i,j), |b_{ij}| \leq 1.$$

The above definition says that the magnitude of all the entries in the matrix B are less than or equal to 1 and that we can permute the columns of B so that it has entries of magnitude 1 on the diagonal.

PROPERTY 4.1. *Let $\mathcal{M}$ be the maximum matching of the symmetric matrix A returned by* `MC64` *and $D_r = (d_{r_i})$, $D_c = (d_{c_i})$ be the row and column scaling respectively. Let $D = (d_i) = \sqrt{D_r D_c}$. Then DAD is a symmetrically scaled matrix that satisfies the* `MC64` *constraints.*

PROOF. The entry of $|DAD|$ in position (i,j) is

$$|d_i d_j a_{ij}| = \sqrt{|d_{r_i} d_{c_j} a_{ij}|}\sqrt{|d_{r_j} d_{c_i} a_{ji}|} \leq 1$$

because it is the square root of the product of two entries of $|D_r A D_c|$. Let σ be the column permutation associated with $\mathcal{M}$. Thus, for all i, $|d_{r_i} d_{c_{\sigma(i)}} a_{i\sigma(i)}| = 1$. Let $\lambda_i = |d_{c_i} d_{r_{\sigma(i)}} a_{i\sigma(i)}| = |d_{c_i} d_{r_{\sigma(i)}} a_{\sigma(i)i}|$. $\forall i$, $\lambda_i \leq 1$ because it corresponds to an entry of $|D_r A D_c|$. Then

$$\prod_{1\leq i\leq n} \lambda_i = \left[\prod_{1\leq i\leq n} |d_{r_i}|\right]\left[\prod_{1\leq i\leq n} |d_{c_i}|\right]\left[\prod_{1\leq i\leq n} |a_{i\sigma(i)}|\right] = \prod_{1\leq i\leq n} |d_{r_i} d_{c_{\sigma(i)}} a_{i\sigma(i)}| = 1,$$

where we exchange the terms in the products to get the product of the scaled terms in the MC64 maximum matching. That is, the product of the numbers, λ_i, each of which is less than or equal to one, is one, so that $\lambda_i = 1,\ \forall i$ and so

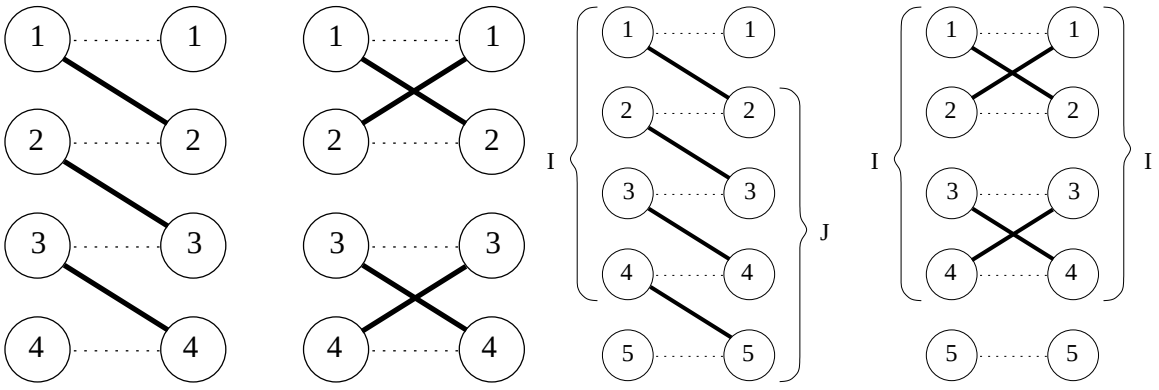
$$\forall i,\ |d_i d_{\sigma(i)} a_{i\sigma(i)}|^2 = |d_{r_i} d_{c_{\sigma(i)}} a_{i\sigma(i)}||d_{r_{\sigma(i)}} d_{c_i} a_{i\sigma(i)}| = 1 \times \lambda_i = 1.$$

□

1.4.2.3 Structurally singular matrices?

Structurally singular matrices are quite common in optimization. The notion of a maximum weighted matching with the product metric is not well defined for MC64. All the maximum matchings of MC64 will have zero weight. Moreover, it is impossible to find scaling factors that satisfy the MC64 constraints for structurally singular matrices (it is impossible to find a permutation σ that satisfies the relation of Definition 4.1).

Suppose that the weight of a matching on a structurally singular matrix is given by the product of the absolute values of the matched entries (which are nonzeros). Then we can define a maximum weighted matching on a structurally singular matrix as a matching of maximum weight among the matchings of maximum size. We did not modify the MC64 code to get this kind of matching, but we used a less complicated algorithm to get an approximation to a maximum weighted matching. We first apply MC64 to A and we get a maximum matching $\mathcal{M} = \mathcal{I} \otimes \mathcal{J}$ where $\mathcal{I}$ is the set of matched rows and $\mathcal{J}$ is the set of matched columns. Then we extract a structurally nonsingular symmetric submatrix $\tilde{A}$ from A using Property 4.2 below. We then apply MC64 this time to $\tilde{A}$.



(1.4.1.a) Example of even maximal open path. (1.4.1.b) Example of odd maximal open path.

Figure 1.4.1: Maximal open paths. Bold edges are edges of the matching. Dashed lines represent the diagonal of the matrix that may not be entries.

PROPERTY 4.2. *Let A be a symmetric matrix and let $\mathcal{M} = \mathcal{I} \otimes \mathcal{J}$ be a maximum matching of A. The restriction of A to $\mathcal{I} \times \mathcal{I}$ is structurally nonsingular.*

PROOF. We define the bipartite graph $G = (R, C, E)$ where $a_{ij} \neq 0$ if and only if $(i^{\mathcal{R}}, j^{\mathcal{C}}) \in E$. The notations $^{\mathcal{R}}$ (resp. $^{\mathcal{C}}$) will be added to a set of indices when we want to explicitly mention that it refers to a set of rows (resp. columns) included in R (resp. in C). A *path* in this graph from i_1 to i_k is defined by a sequence $(i_1, \ldots, i_k)$ where $(i_t^{\mathcal{R}}, i_{t+1}^{\mathcal{C}})_{t=1,\ldots,k-1} \in E$. This path is a *cycle* if $i_k \equiv i_1$. By definition, the *length* of

this path is k. We can similarly define a path using the maximum matching $\mathcal{M} \subset E$ where $(i_t^{\mathcal{R}}, i_{t+1}^{\mathcal{C}})_{t=1,\ldots,k-1} \in \mathcal{M}$. A *maximal open path of the matching* is defined as a path $(i_1, \ldots, i_k)$ where there are no v such that $(v^{\mathcal{R}}, i_1^{\mathcal{C}}) \in \mathcal{M}$ or $(i_k^{\mathcal{R}}, v^{\mathcal{C}}) \in \mathcal{M}$. All matching edges are either in cycles or in maximal open paths.

The length of a maximal open path must be odd since, if it were even the symmetry of the matrix would allow us to extend the matching by using the edges $(i_1^{\mathcal{R}}, i_2^{\mathcal{C}}), (i_2^{\mathcal{R}}, i_1^{\mathcal{C}}), (i_3^{\mathcal{R}}, i_4^{\mathcal{C}}), (i_4^{\mathcal{R}}, i_3^{\mathcal{C}}) \ldots, (i_{k-1}^{\mathcal{R}}, i_k^{\mathcal{C}}), (i_k^{\mathcal{R}}, i_{k-1}^{\mathcal{C}})$ thus contradicting that we have a maximum matching. Figure 1.4.1.a shows this extension for $k = 4$.

We now construct a maximum matching on $\mathcal{I}^{\mathcal{R}} \times \mathcal{I}^{\mathcal{C}}$ of the same cardinality as that on $\mathcal{I}^{\mathcal{R}} \times \mathcal{J}^{\mathcal{C}}$. Clearly any cycle gives corresponding matching edges in $\mathcal{I}^{\mathcal{R}} \times \mathcal{I}^{\mathcal{C}}$. Consider a maximal open path $(i_1, \ldots, i_k)$, $i_1, \ldots, i_{k-1} \in \mathcal{I}$, $i_k \in \mathcal{J} \setminus \mathcal{I}$. Then we can replace the matching edges corresponding to this path by $(i_1^{\mathcal{R}}, i_2^{\mathcal{C}}), (i_2^{\mathcal{R}}, i_1^{\mathcal{C}}), \ldots, (i_{k-2}^{\mathcal{R}}, i_{k-1}^{\mathcal{C}}), (i_{k-1}^{\mathcal{R}}, i_{k-2}^{\mathcal{C}})$ all of whose end points are in $\mathcal{I}$. It is illustrated by the example in Figure 1.4.1.b for $k = 5$.

This then gives us a maximum matching on $\mathcal{I} \times \mathcal{I}$ of cardinality $|\mathcal{I}|$. □

Let D_r and D_c be the scaling factors returned by `MC64` on the restriction of A to $\mathcal{I} \times \mathcal{I}$, $\tilde{A}$. We build D so that the entries of $|DAD|$ are less than 1 and entries in the matching are 1. If i corresponds to an index of $\tilde{A}$, we define $d_i = \sqrt{d_{r_i} d_{c_i}}$. If i does not correspond to the index in $\mathcal{I}$, we want all the entries of the i^{th} row (resp. column) of $|DAD|$ to be less than or equal to 1. We decide to take

$$d_i = \frac{1}{\max_{k \in index(\tilde{A})} |a_{ik} d_k|} \text{ with the convention } \frac{1}{0} = 1.$$

Thus for all $i \notin \mathcal{I}$ and for all $j \in \mathcal{I}$ we have $|d_i a_{ij} d_j| = \frac{|a_{ij} d_j|}{\max_{k \in \mathcal{I}} |a_{ik} d_k|} \leq 1$. Note also that the maximum entry in absolute value in each non-empty row/column is 1.

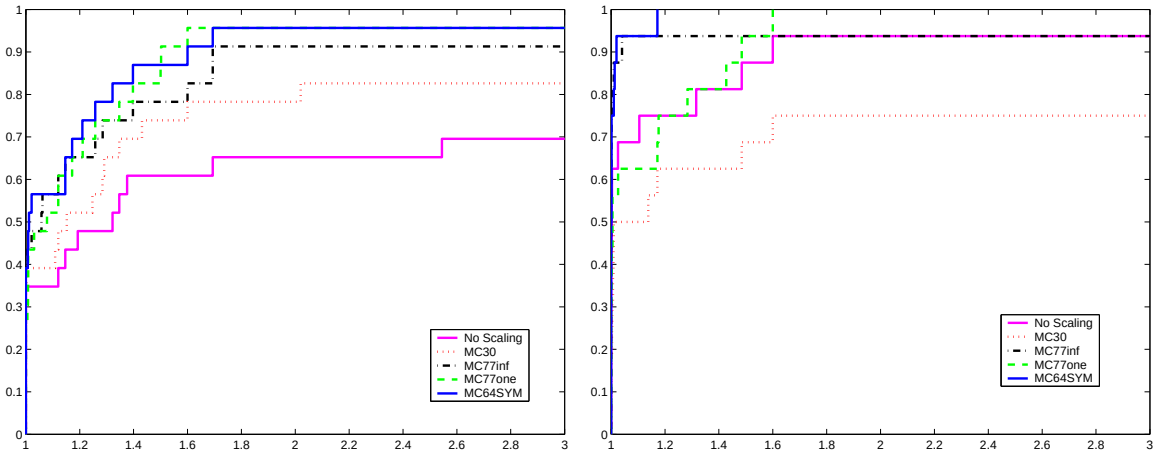
1.4.3 Influence of scaling

In this section we analyse the influence of the scaling on the factorization phase. As mentionned in Section 1.3.3 we compare our codes using the performance profiling system of [29].

It is important to note that scaling does not change the pivot order returned by the analysis but changes the selection of the numerically stable 1×1 and 2×2 pivots during the factorization. Scaling can have a profound effect on the subsequent factorization. A good scaling can avoid many numerical difficulties whereas a bad scaling can actually cause numerical problems, can produce a lot of delayed pivots when not necessary, can increase the memory requirements, and can consequently severely slow down the factorization.

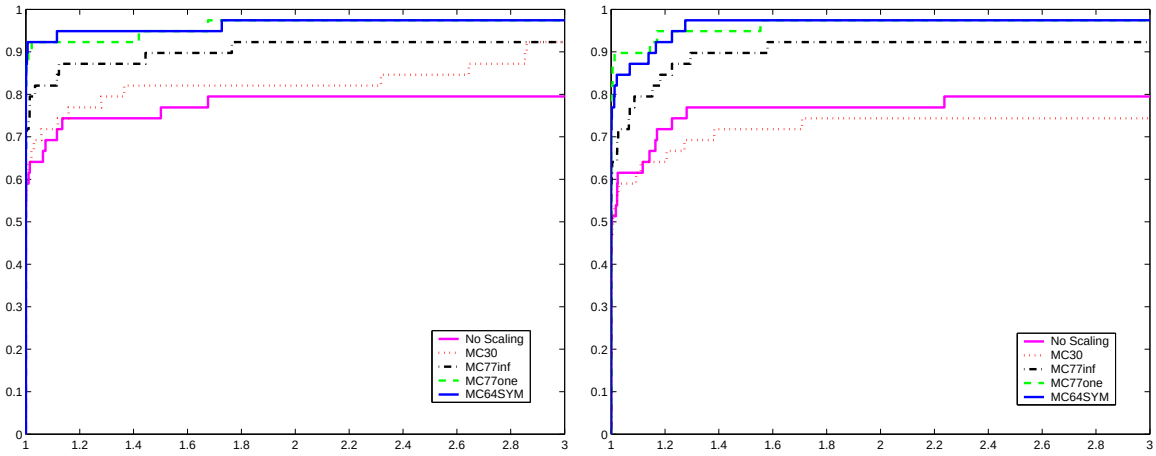
On sets 2 and 3, the approach without any scaling (No Scaling) fails on 6 matrices, the approaches using `MC30`, `MC77inf`, `MC77one` and `MC64SYM` fail on 3, 3, 1 and 1 matrices respectively. The use of scaling thus improves the robustness of the `MA57` factorization in terms of the number of successful computations. Table 1.4.1 shows that most of the time failures are due to numerical pivoting which increases the CPU time and memory requirement (C and M failures). It seems difficult to get a good solution for the BRATU3D problem, except with the `MC77one` scaling. Figure 1.4.2 compares the

influence of the different scalings on the CPU factorization time of MA57 on sets 1 and 2.



(1.4.2.a) set 2. (1.4.2.b) set 3.

Figure 1.4.2: Profile showing influence of scaling on CPU factorization time (AMD ordering).

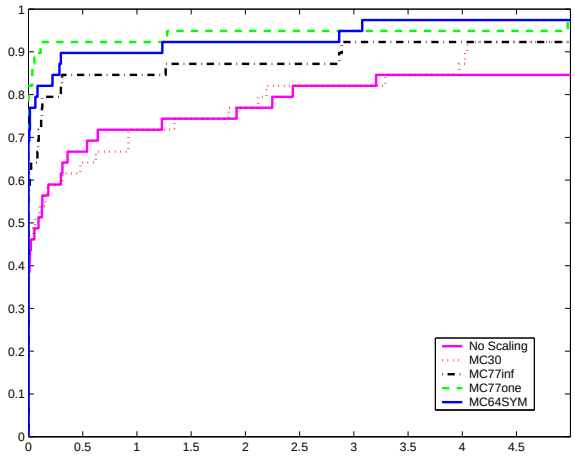


(1.4.3.a) Memory profile. (1.4.3.b) Total number of operations profile.

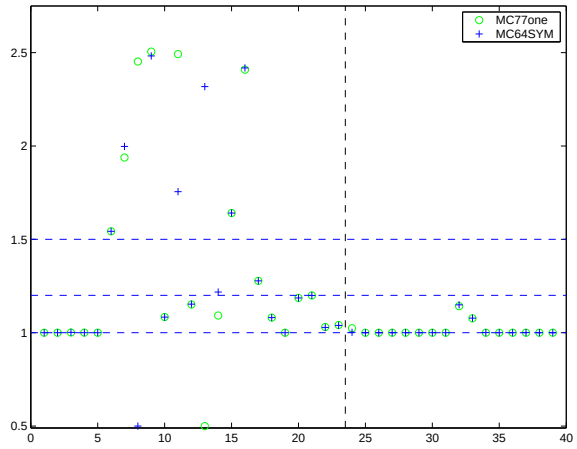
Figure 1.4.3: Profile showing influence of the scaling on memory and number of operations for the factorization (Sets 2 and 3 together).

On set 3 (Figure 1.4.2.b), the MC30 scaling degrades the MA57 performance, whereas the other scalings have a positive effect on these general indefinite matrices. The negative effect of MC30 on indefinite problems has also been observed by [62]. Contrary to the other scalings (MC64, MC77), MC30 does not take into account the nonzero structure when doing the scaling and uniformly tries to scale all entries to be as close to one as possible. We suspect that the structural decision of the analysis is less compatible with the MC30 scaling. For example, MC77 used with the 1-norm tends to decrease the magnitude of the entries with a high connectivity and tends to make the magnitude of the entries with a small connectivity large. This last entries are also the ones that an ordering tends to select first. This difference can provide a partial answer about the degradation of performance when using the MC30 scaling. We recommend that the MC30 scaling is not used on symmetric indefinite problems. Furthermore, the MC64SYM scaling seems to be

the most robust scaling on this set (the `MC77inf` approach is also good but only for 95% of the time, whereas the `MC64SYM` scaling is within a factor of 1.17 of the best on 100% of the problems). `MC64SYM` and `MC77one` have a similar behaviour on the augmented systems and are slightly better than `MC77inf` (see Figure 1.4.2.a).



(1.4.4.a) Number of delayed pivots profi le. $\log_{10}$ scale is use for the x-axis (number of delayed pivots).



(1.4.4.b) Ratio between memory used and memory predicted by the analysis on the y-axis. Ratio of 0.5 is used as a dummy value to indicate failure; problems sorted by Id on the x-axis, set 2 on the left, set 3 on the right.

Figure 1.4.4: Profi le showing influence of scaling on number of delayed pivots and memory (`AMD` ordering).

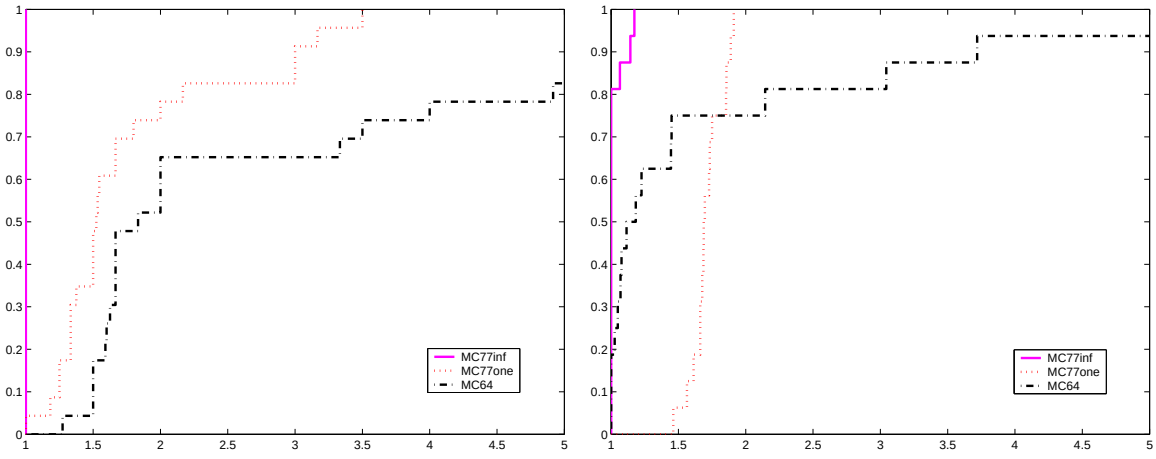
The CPU factorization times in Figure 1.4.2, the number of operations (Figure 1.4.3.b), the memory used (Figure 1.4.3.a) and the number of delayed pivots have similar characteristics. Figure 1.4.4.a shows that the number of delayed pivots is sometimes very large when using the `MC30` scaling. The average behaviour of `MA57` using the `MC30` scaling is similar to applying no scaling. We show in Figure 1.4.4.b the ratio between the memory used by the factorization and the memory predicted by the analysis. This indicates that the memory predictions of the analysis are sometimes not respected. It also indicates that most of the numerical problems appear on set 2 (to the left of the vertical line). Indeed there are no problems in set 3 over the 20% limit. If we allocate only 50% more memory than recommended by the analysis instead of 4 GBytes, we would have 7 additional failures with the `MC64SYM` and `MC77one` scalings (points above the highest horizontal dotted line in Figure 1.4.4.a).

Table 1.4.1 summarizes the `MC64SYM`, `MC77inf` or `MC77one` scaling impact on the factorization of symmetric indefinite problems. Using the `MC64SYM` scaling, we get a speedup greater than 5 on 7 matrices and a speedup of between 2 and 5 on two matrices. We are able to solve 5 problems using the symmetric scaling while `MA57` without scaling fails. This approach is faster than no scaling on 30 matrices and on 9 others it never exceeds the factorization time for no scaling by a factor of more than 1.25. We see also that on set 3, the `MC77one` scaling has problems whereas `MC64SYM` does not (see c-68 and c-71 matrices). As mentioned before, on set 3, the same order of speedup is obtained with the `MC77inf` scaling even if it is less robust in terms of the number of failures.

It may be interesting to use the `MC77inf` scaling on set 3 and the `MC77one` scaling on set 2 because they have a comparable influence on the factorization and are less

Matrix	NoScaling	MC77inf		MC77one		MC64SYM	
	F	F	S	F	S	F	S
Augmented systems (set 2)							
BLOWEYA	C	0.08	0.04	0.09	0.04	0.08	0.03
BRATU3D	P	P	–	77.6	0.06	P	–
NCVXQP1	108.	13.6	0.03	19.3	0.01	12.8	0.43
NCVXQP5	M	138.	0.19	140.	0.18	139.	2.94
NCVXQP7	M	M	–	C	–	1307	21.9
bloweybl	C	0.06	0.03	0.08	0.04	0.06	0.03
cvxqp3	263.	30.6	0.03	37.0	0.04	37.0	0.94
stokes128	2.03	0.81	0.21	0.79	0.16	0.81	0.29
stokes64	0.18	0.14	0.04	0.14	0.01	0.13	0.06
tuma2	0.08	0.08	0.01	0.06	0.01	0.04	0.01
TOTAL solved	18/23	21/23		22/23		22/23	
General symmetric indefi nite (set 3)							
BOYD1	C	C	–	39.9	0.44	33.9	14.8
c-68	24.5	23.1	0.29	28.5	0.21	22.2	0.13
c-71	76.4	76.2	0.48	108.	0.29	76.4	0.28
vibrobox	1.66	1.26	0.04	1.29	0.06	1.28	0.03
TOTAL solved	15/16	15/16		16/16		16/16	

Table 1.4.1: Factorization time (columns F) and scaling time (columns S). Times in seconds. AMD ordering used. Number of steps for MC77 set to 20. C: maximum CPU time exceeded. M: MA57 ran out of memory. P: problem with the precision of the solution.



(1.4.5.a) Scaling CPU profi le (set 2).

(1.4.5.b) Scaling CPU profi le (set 3).

Figure 1.4.5: Scaling time comparison. Number of MC77 iterations set to 20.

costly to compute (see Figure 1.4.5 and Table 1.4.1). Nevertheless, MC64SYM remains competitive. Note that the maximum number of iterations of MC77 has been set to 20 in our experiments and the norms of the rows and columns may still be far from 1 (for example with MC77one on set 3). A safe approach with MC77 could be to check the convergence every 20 steps and continue iterating until we are close to the limit. It would involve an additional cost that is not included in Table 1.4.1 and in the profile of Figure 1.4.5. On the contrary, when MC64SYM has finished, it guarantees that our scaled matrix satisfies Property 4.1. Secondly, because the scaling time has to be considered

relative to the factorization time, Table 1.4.1 shows that the MC64SYM cost is usually compensated for by the gain in factorization time.

In the rest of this chapter, we try to improve the CPU factorization time and to obtain a better behaviour with respect to other criteria (for example, the quality of the memory prediction). A first way of improving the MA57 factorization comes from using other orderings (METIS and AMF) to decrease the fill-in, the memory and the number of operations. A second way is to influence $a\ priori$ the ordering with numerical and structural information about the 2×2 pivots. The main goal of this approach is to preselect the pivots that will be effectively used during the factorization and thus decrease the number of delayed pivots. As MC64 is used in the approach that tries to answer the second point (ie the use of the MC64SYM scaling does not involve additional computation) and as the MC64SYM scaling seems to be a robust approach on both sets, we will systematically use this scaling in the remainder of this paper. From now on, A will refer to the matrix symmetrically scaled by MC64SYM (its entries are in $[-1,1]$).

1.5 Ordering and weighted matching

In this section, we present ordering approaches that aim at decreasing fill-in in the factors and at preselecting good pivots during the analysis. Thus we are interested in an algorithm that gives us a good approximation about what will happen during the factorization and that does not generate too much fill-in in the factors. We will first discuss our choices in Section 1.5.1. Then we present our different approaches. In Section 1.5.2 we present a way to preselect a set of 2×2 and 1×1 pivots. In Section 1.5.3, we discuss about three different orderings which are called ordering on the compressed graph, constrained ordering and relaxed constrained ordering.

The matrix

$$A33(x) = \begin{pmatrix} 0 & 1 & 1 & x & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ x & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

that has two 3×3 cycles with edges of weight 1 will be used in this section to illustrate our discussions.

1.5.1 Symmetric maximum weighted matching

In the unsymmetric case, a maximum weighted matching of the **bipartite graph** associated with the matrix is computed to put large entries onto the diagonal of the permuted matrix. A natural way of adapting this to the symmetric case is to search for a **symmetric maximum weighted matching**, $\mathcal{M}_s^{opt}$ (a maximum weighted matching such that if (i,j) is in $\mathcal{M}_s^{opt}$ then (j,i) is in $\mathcal{M}_s^{opt}$). Property 5.1 presents a method to get a symmetric maximum weighted matching. We will then explain why we decided not to use this method and to build only an approximation of it using the

MC64 maximum weighted matching. The fact that using MC64 may not return the optimum can be seen in the example $A33(10^{-1})$. MC64 could return the matching $\mathcal{M} = \{(1,2),(2,3),(3,1),(4,5),(5,6),(6,4)\}$ where we use the convention that, on a bipartite graph, the first index of an edge corresponds to a row vertex and the second index to a column vertex. However, the symmetric maximum weighted matching is $\mathcal{M}_s^{opt} = \{(1,4),(4,1),(2,3),(2,3),(6,5),(5,6)\}$ (if $x \neq 0$, it is the unique symmetric matching of size 6 and so it is the symmetric maximum weighted matching). It is impossible to build $\mathcal{M}_s^{opt}$ using only information in $\mathcal{M}$ because $(1,4)$ does not appear in it.

PROPERTY 5.1. *The problem of finding a symmetric maximum weighted matching in a bipartite graph is equivalent to the problem of finding a maximum weighted matching on an undirected graph.*

PROOF. Let $\mathcal{G}_A = (V_A, E_A)$ be a weighted graph associated with the symmetric matrix A. Note that it might not be a bipartite graph. We first define an undirected graph $\mathcal{G}$ with twice the number of vertices as $\mathcal{G}_A$. We associate to each node i a node i' that will be referred to as its companion node.

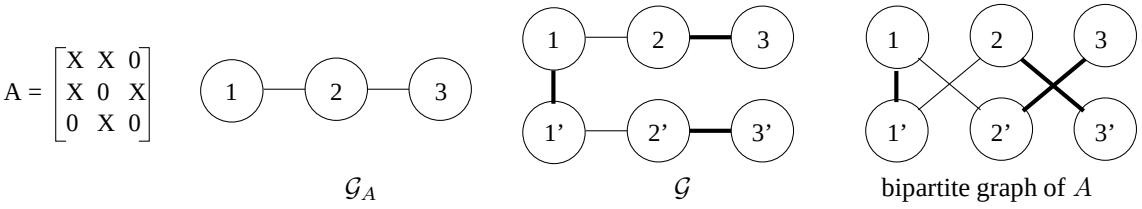


Figure 1.5.1: Computation of a symmetric maximum weighted matching using a maximum weighted matching on an undirected graph. Bold edges correspond to matching entries.

Let

$$\begin{aligned} V'_A &= \{i' \text{ such that } i \in V_A\}, \text{ be the set of companion nodes,} \\ E'_A &= \{(i',j') \text{ with weight } |a_{ij}| \text{ such that } (i,j) \in E_A\} \\ \text{and } E_{diag} &= \{(i,i') \text{ with weight } |a_{ii}| \text{ such that } i \in V_A \text{ and } a_{ii} \neq 0\}. \end{aligned}$$

Then $\mathcal{G} = (V_A \cup V'_A, E_A \cup E'_A \cup E_{diag})$ is the required weighted undirected graph. Figure 1.5.1 illustrates this construction on a small 3×3 example.

The bipartite graph of A is noted $G_b = (V_A, V'_A, E_b)$. V_A refers to the set of row vertices and V'_A refers to the set of column vertices. Edges of E_b only go from V_A to V'_A ($E_b \subset V_A \times V'_A$). An edge $(i,j') \in E$ exists if and only if $a_{ij} \neq 0$. Note that there is a simple correspondence between $\mathcal{G}$ and the bipartite graph of A. Let $\mathcal{M}_0$ be a maximum weighted matching on $\mathcal{G}$. $\mathcal{M}_0$ can be split into three parts:

$$\begin{aligned} \mathcal{S}_1 &= \mathcal{M}_0 \cap E_A, \\ \mathcal{S}_2 &= \mathcal{M}_0 \cap E'_A \\ \text{and } \mathcal{S}_d &= \mathcal{M}_0 \cap E_{diag}. \end{aligned}$$

We have $\omega(\mathcal{S}_1) = \omega(\mathcal{S}_2)$ (ω as defined in Section 1.4.2.1), otherwise $\mathcal{M}_0$ would not be a maximum weighted matching (because for example if $\omega(\mathcal{S}_1) > \omega(\mathcal{S}_2)$ then the matching $\mathcal{M}_1$ below has larger weight than $\mathcal{M}_0$). $\mathcal{M}_1 = \mathcal{S}_1 \cup \mathcal{S}'_1 \cup \mathcal{S}_d$ is a maximum weighted matching on $\mathcal{G}$ where

$$\mathcal{S}'_1 = \{(i', j') \text{ such that } (i, j) \in \mathcal{S}_1\}.$$

Let

$$\mathcal{M}_s = \{(i, j') \text{ such that } (i, j) \in \mathcal{S}_1 \text{ or } (j, i) \in \mathcal{S}_1\} \cup \mathcal{S}_d,$$

be a symmetric matching on the bipartite graph of A. Suppose for the purpose of deriving a contradiction that $\mathcal{M}_s$ is not a symmetric maximum weighted matching, and let $\mathcal{M}_{opt}$ be a symmetric maximum matching on the bipartite graph such that $\omega(\mathcal{M}_{opt}) > \omega(\mathcal{M}_s)$. We can build the matching

$$\begin{aligned} \mathcal{M}_2 \;=\; & \{(i, j) \text{ such that } (i, j') \in \mathcal{M}_{opt} \text{ and } i \neq j\} \cup \\ & \{(i', j') \text{ such that } (i, j') \in \mathcal{M}_{opt} \text{ and } i \neq j\} \cup \\ & \{(i, i') \text{ such that } (i, i') \in \mathcal{M}_{opt}\} \end{aligned}$$

so that $\omega(\mathcal{M}_2) > \omega(\mathcal{M}_0)$ giving us a contradiction. Thus, $\mathcal{M}_s$ is a symmetric maximum weighted matching on the bipartite graph of A and solving the problem of the maximum weighted matching on a non-bipartite graph, enables us to solve the problem of the symmetric maximum weighted matching.

Conversely, if we solve the problem of finding a symmetric maximum weighted matching on the bipartite graph of A, using the same kind of transformations as before, we have the solution of the problem of the maximum weighted matching on the non-bipartite graph of A. □

The equivalence of Property 5.1 has been established by [76] in the context of maximum cardinality matching. Property 5.1 gives us a way of computing a symmetric maximum weighted matching and says that the complexity of this problem is of the same order as the complexity of the computation of the maximum weighted matching on a graph with the same number of vertices and with the same number of edges as in the bipartite graph of A. Efficient algorithms for finding maximum matchings use the Hungarian method of Kuhn [74]. They are based on iterative process in which at each iteration a so called shortest augmenting path is computed. Solving the problem of the maximum weighted matching on a non-bipartite graph is much more complicated than on a bipartite one. A solution to this problem has been found by [49, 50] and its complexity was originally bounded by $\mathcal{O}(n^4)$. This complexity bound was later decreased to $\mathcal{O}(nnz(A)n + n^2 \log n)$ by [52].

Nevertheless, we will not use Property 5.1 to find good 1×1 and 2×2 pivots. We prefer to find an approximation to symmetric maximum matching using information returned by `MC64` for the following reasons:

- We want a preprocessing with reasonable complexity with respect to the rest of the analysis and the factorization.
- We do not know an efficient code available for using 5.1.

- In the unsymmetric case, MC64 has a complexity in the worst case of $\mathcal{O}(\tau n \log n)$ where τ is the number of nonzeros in A and in practice has a better average behaviour.
- Using MC64 enables us to compute both the MC64SYM scaling and an approximation of a symmetric maximum weighted matching in linear time. We get two complementary preprocessings for the cost of one.

That is why we decided to have a weaker formulation of the problem – we want to find a symmetric weighted matching that is not too far from the optimum – and to use the maximum matching technique on weighted bipartite graphs. We will see in Section 1.5.2.3 that on 37 matrices out of 43 in our test set a solution to the weak formulation leads to a solution of the symmetric maximum weighted matching. In Section 1.5.2.2, we also give theoretical bounds as to how far this formulation can be from the symmetric maximum matching.

1.5.2 Selection of 2 by 2 pivots and symmetric weighted matchings

1.5.2.1 MC64SYM algorithm

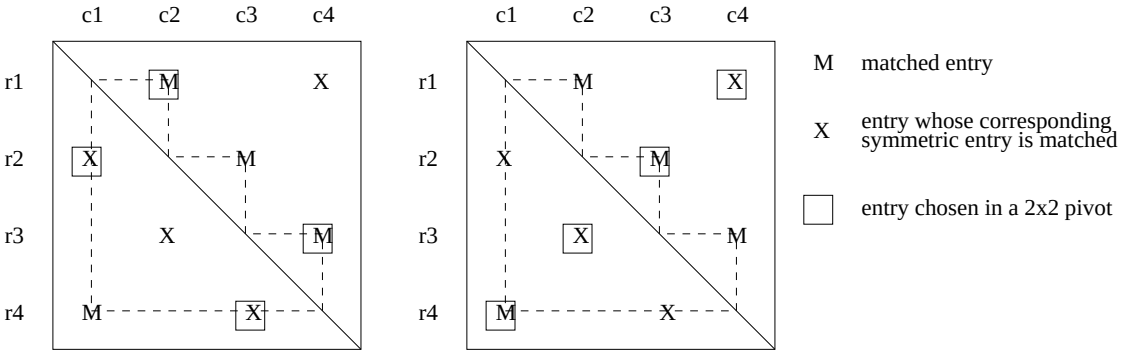


Figure 1.5.2: Example of selection of 2×2 pivots in an even cycle. Matched entries are $(r1, c2)$, $(r2, c3)$, $(r3, c4)$ and $(r4, c1)$. We show the two possibilities for selecting two 2×2 pivots.

MC64 is first used to compute a maximum weighted matching $\mathcal{M}$. This will normally have many entries that are not on the diagonal of A. Any diagonal entries that are in the matching are immediately considered as potential 1×1 pivots and are held in a set $\mathcal{M}_{1\times1}$. We then build a set $\mathcal{M}_{2\times2}$ of potential 2×2 pivots. We use the 2×2 pivot selection strategy suggested by [34], but with a structural metric instead of a numerical criterion. The basis for this strategy is to express the computed permutation, σ, in terms of its component cycles. Because of the scaling, all the entries in the cycles of σ are 1 in absolute value so we choose a structural criterion to select the potential 2×2 pivots. Cycles of length 1 correspond to a matching on the diagonal. We can extract k 2×2 pivots from even cycles of length $2k$ or from odd cycles of length $2k+1$. For even cycles there are only two possibilities for extraction. For example, in Figure 1.5.2 there are two alternatives to select two 2×2 pivots $p1$ and $p2$. In the first alternative (left figure), $p1$ is composed of an entry in position $(r1, c2)$ and its symmetric counterpart,

and $p2$ is composed of an entry in position $(r3, c4)$ and its symmetric counterpart. In the second alternative (right figure), $p1$ is composed of an entry in position $(r2, c3)$ and its symmetric counterpart, and $p2$ is composed of an entry in position $(r4, c1)$ and its symmetric counterpart.

For odd cycles of length $2k + 1$ there are $2k + 1$ possible combinations of 2×2 pivots. For example, in Figure 1.5.3 there are five possibilities for selecting two 2×2 pivots. In the first choice, $p1$ is composed of an entry in position $(r1, c2)$ and its symmetric counterpart, $p2$ is composed of an entry in position $(r3, c4)$ and its symmetric counterpart. In the second choice, $p1$ is composed of an entry in position $(r2, c3)$ and its symmetric counterpart, $p2$ is composed of an entry in position $(r4, c5)$ and its symmetric counterpart. In the third choice, $p1$ is composed of an entry in position $(r3, c4)$ and its symmetric counterpart, $p2$ is composed of an entry in position $(r5, c1)$ and its symmetric counterpart. In the fourth choice, $p1$ is composed of an entry in position $(r4, c5)$ and its symmetric counterpart, $p2$ is composed of an entry in position $(r1, c2)$ and its symmetric counterpart. In the fifth choice, $p1$ is composed of an entry in position $(r5, c1)$ and its symmetric counterpart, $p2$ is composed of an entry in position $(r2, c3)$ and its symmetric counterpart.

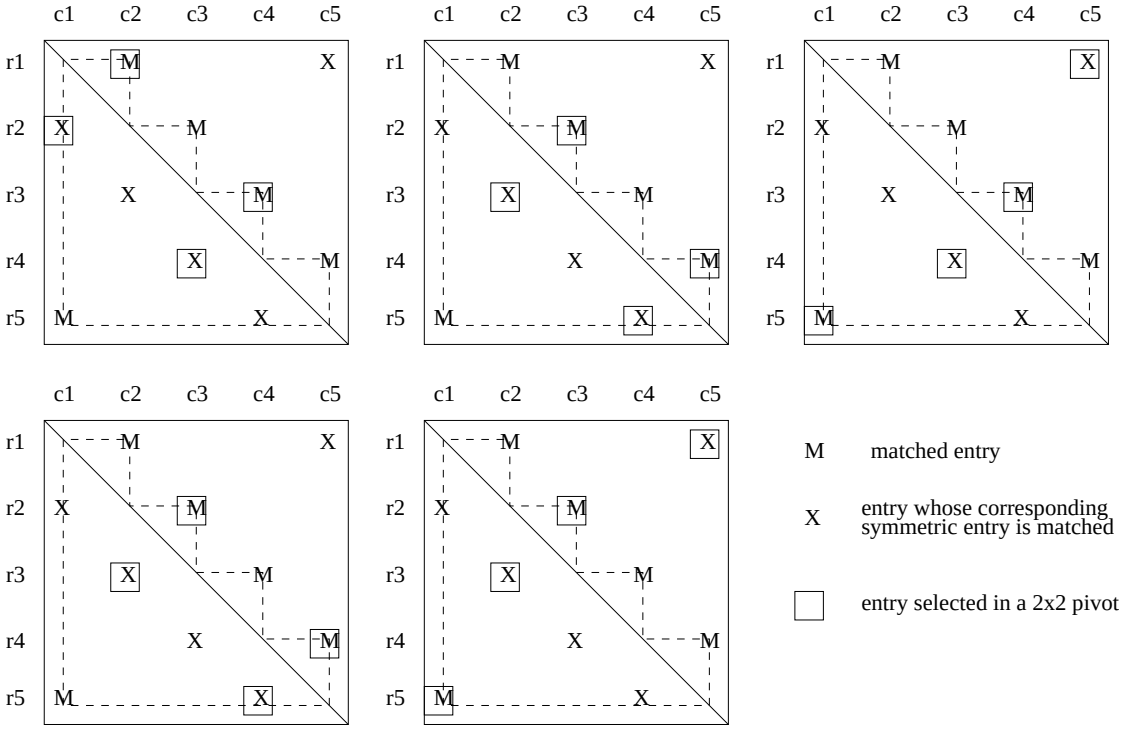


Figure 1.5.3: Example of selection of 2×2 pivots in an odd cycle. Matched entries are $(r1, c2)$, $(r2, c3)$, $(r3, c4)$, $(r4, c5)$ and $(r5, c1)$. We show the five possibilities for selecting two 2×2 pivots.

To make a choice from the alternative sets of pivots in Figures 1.5.2 and 1.5.3, we define a metric that is denoted by $metric(i, j)$ for a potential 2×2 pivot in rows i and j. In our approach, we use $metric(i, j) = |R_i \cap R_j|/|R_i \cup R_j|$ where R_i and R_j are the structure of row i and j respectively. The motivation for this metric is to associate rows in 2×2 pivots that have as similar a structure as possible. We will see in Section 1.5.3 that before

the ordering phase, for each 2×2 pivot (i,j), we assume that its nonzero structure is the merge of the structure of row i and j. So maximizing this metric will tend to decrease the fill-in due to the use of 2×2 pivots. For each cycle in σ, we then select 2×2 pivots such that the sequence of 2×2 pivots (i_k,j_k) maximizes $\prod metric(i_k,j_k)$.

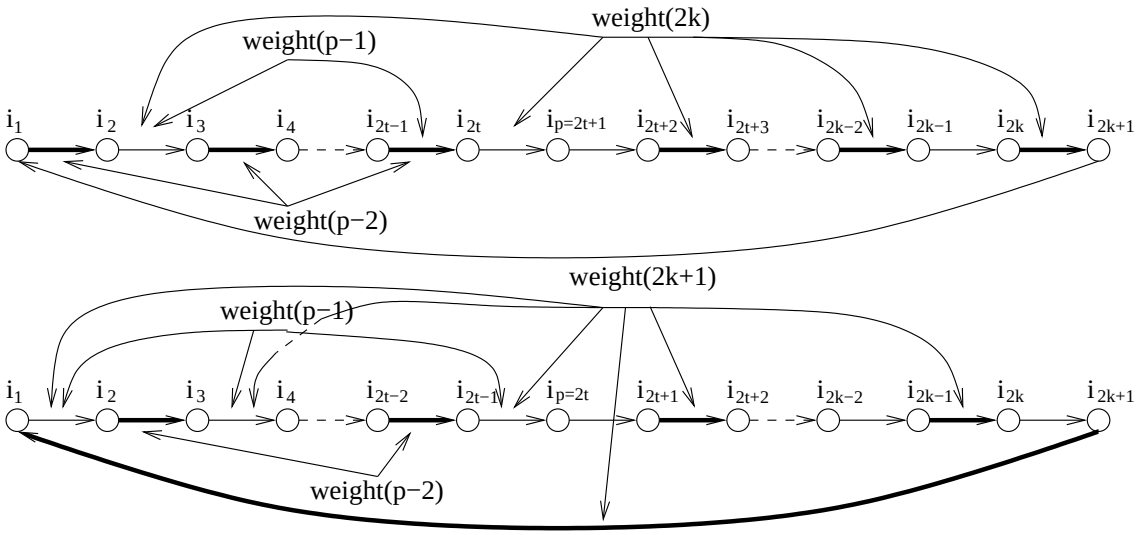


Figure 1.5.4: Computation of the weight of a set of 2×2 pivots when row/column p is not selected.

We will now see that this extraction can be done in two passes over the matching and thus in $\mathcal{O}(n\times c)$, where the cost of the computation of the structural score of a 2×2 pivot is $\mathcal{O}(c)$. Let $(i_p,i_{p+1})_{1\le p\le l}$ be the edges of a cycle of length l ($i_1=i_{l+1}$). In a first pass over the cycle, for each edge (i_p,i_{p+1}), $metric(i_p,i_{p+1})$ is computed in $\mathcal{O}(c)$ and a cumulative weight is computed:

$$weight(p)=weight(p-2)\times metric(i_p,i_{p+1})$$

with the convention $weight(0)=weight(-1)=1$. If $l=2k$, we only need to compare the values of $weight(2k)$ and $weight(2k-1)$ to extract the best 2×2 pivots. We now consider the case where $l=2k+1$. If the diagonal entry i_p remains unselected, then there is only one way of selecting the 2×2 chain (see Figure 1.5.4):

$$\text{if } p \text{ is odd, } \overbrace{(i_1,i_2),\ldots,(i_{p-2},i_{p-1})}^{\text{part 1}},\overbrace{(i_{p+1},i_{p+2}),\ldots,(i_{2k},i_{2k+1})}^{\text{part 2}} \text{ are selected,}$$

$$\text{if } p \text{ is even, } \overbrace{(i_2,i_3),\ldots,(i_{p-2},i_{p-1})}^{\text{part 1}},\overbrace{(i_{p+1},i_{p+2}),\ldots,(i_{2k+1},i_1)}^{\text{part 2}} \text{ are selected.}$$

Hence, when the entry in the p^{th} position is the unselected entry on the diagonal, the total metric of the selected 2×2 pivots is given directly by:

$$\frac{weight(p-2)\times weight(l-mod(p,2))}{weight(p-1)}$$

where the term $weight(p-2)$ corresponds to the metric of part 1 of the 2×2 chain and the term $weight(l-mod(p,2))/weight(p-1)$ corresponds to the metric of part 2 of the

2×2 chain. Thus, using the above relation, the best 2×2 pivots can be chosen during a second loop of length l.

For the above metric, $|Row_{i_k} \cap Row_{j_k}|$ and $|Row_{i_k} \cup Row_{j_k}|$ can be computed in a time bounded by $\mathcal{O}(\max_i |Row_i|)$ and thus the total complexity of the extraction is $\mathcal{O}(n \max_i |Row_i|)$. Note that this total cost can be reduced to $\mathcal{O}(nnz(A))$, if we use the flags set during the computation of $|Row_{i_p} \cap Row_{i_{p+1}}|$ for the computation of $|Row_{i_{p+1}} \cap Row_{i_{p+2}}|$, when i_p, i_{p+1}, i_{p+2} are three consecutive indices in a cycle.

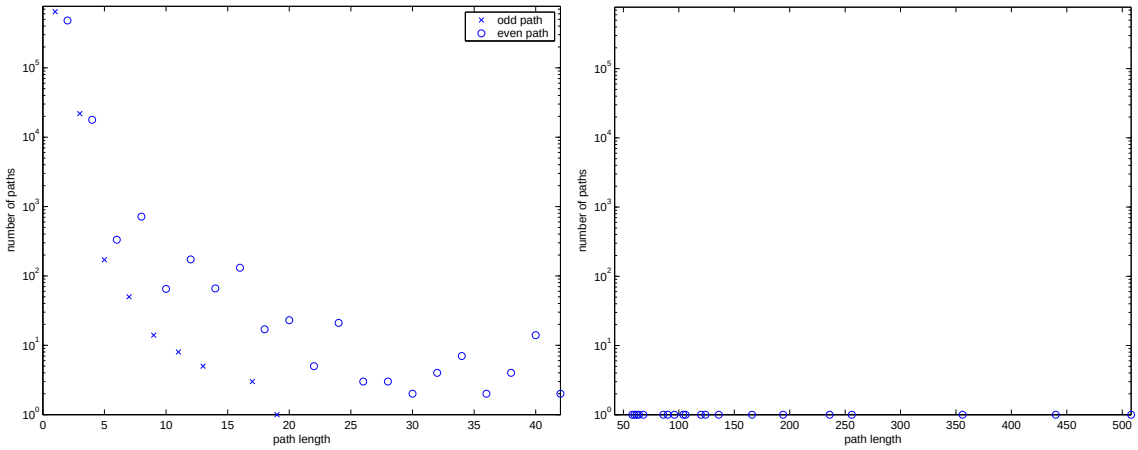


Figure 1.5.5: Length of MC64 paths.

At the end of this $\mathcal{M}_{2\times2}$ computation, we add to $\mathcal{M}_{1\times1}$ the nonzero diagonal entries that were not selected during the pass over odd cycles. We define $\mathcal{M}_s$ as $\mathcal{M}_{1\times1} \cup \mathcal{M}_{2\times2}$ corresponding to a symmetric matching obtained from $\mathcal{M}$. We note that $\mathcal{M}_{1\times1} \cap \mathcal{M}_{2\times2} = \emptyset$.

Let $\mathcal{M}_s^c$ be the set $\{(i,i)$ such that i does not belong to any pivots in $\mathcal{M}_s\}$, that is the complementary set to $\mathcal{M}_s$. It may be non empty because of odd cycles with zero diagonals. In practice, we note that this set is small ($\mathcal{M}_s^c = \emptyset$ on all the matrices of sets 2 and 3 except the structurally singular ones). Thus we will not investigate approaches to increase the size of $\mathcal{M}_s$ further.

In practice, most of the cycles from the MC64 permutation are of length 1 or 2. On our test matrices, 55% of the cycles are of length 1, 41% are of length 2 and less than 4% have a length greater than 2. We illustrate this in Figure 1.5.5 where we show the total number of paths of varying lengths for runs over all of our test matrices. That is why we will not try different structural criteria to select the 2×2 pivots.

1.5.2.2 Theoretical bounds

We will now give properties which are quite obvious but which have the merit of fixing the ideas about some theoretical bounds. After estimating these bounds, we will conclude that using MC64 is a safe approach. We use the notations $\mathcal{M}^{opt}$ for the maximum weighted matching returned by MC64, p_{opt} for its associated permutation, $\mathcal{M}_s^{opt}$ for the symmetric

maximum weighted matching, N_{odd} for the number of odd cycles in p_{opt} and N_1 for the number of cycles of length 1 in p_{opt} (cycles of length 1 are also counted as odd cycles). $\mathcal{M}_s$ is still the symmetric matching extracted by our `MC64` based approach.

PROPERTY 5.2. *If all the cycles of the matching are even or of length 1 then $\mathcal{M}_s$ is a symmetric maximum weighted matching.*

PROPERTY 5.3.

$$|\mathcal{M}^{opt}| \geq |\mathcal{M}_s^{opt}| \geq |\mathcal{M}_s|$$

and

$$|\mathcal{M}_s| \geq |\mathcal{M}^{opt}| - N_{odd} + N_1.$$

Let us consider the example $A33(10^{-1})$ to illustrate the bounds of the above property. `MC64` returns the column permutation $p = [2, 3, 1, 5, 6, 4]$, $N_{odd} = 2$, $N_1 = 0$, $\mathcal{M}_s = \{(1,2),(2,1),(4,5),(5,4)\}$ and $\mathcal{M}_s^{opt} = \{(1,4),(4,1),(2,3),(2,3),(6,5),(5,6)\}$. Thus the difference between $|\mathcal{M}_s^{opt}|$ and $|\mathcal{M}_s|$ is exactly the number of odd cycles of length greater than 1.

PROPERTY 5.4. *If $|\mathcal{M}_s| = |\mathcal{M}_s^{opt}|$ and the pivot extraction is done with $metric(i,j) = |a_{ij}|$ then*

$$K_1 \leq \frac{\omega(\mathcal{M}_s)}{\omega(\mathcal{M}_s^{opt})} \leq 1$$

with

$$K_1 = \prod_{(j,j) \in \mathcal{M}_{1\times1} \setminus \mathcal{M}^{opt}} \frac{|a_{jj}|}{\min\{|a_{jp_{opt}(j)}|, |a_{p_{opt}^{-1}(j)j}|\}}.$$

PROOF. Let $\mathcal{C}$ be an odd cycle of p_{opt} of length greater than 1. Let j be the row/column index that appears in $\mathcal{C}$ and has not been selected during the 2×2 pivot extraction. (j,j) is not in $\mathcal{M}^{opt}$ because $(j, p_{opt}(j)) \in \mathcal{M}^{opt}$ and $p_{opt}(j) \neq j$. $|a_{jj}| \neq 0$ because of the $|\mathcal{M}_s| = |\mathcal{M}_s^{opt}|$ assumption. So (j,j) has necessarily been added in $\mathcal{M}_s$ and more precisely to $\mathcal{M}_{1\times1}$ after the selection of the 2×2 pivots (see Section 1.5.2.1). That is why (j,j) belongs to $\mathcal{M}_{1\times1} \setminus \mathcal{M}^{opt}$. The $(\mathcal{C} \setminus \mathcal{M}_s) \setminus \{(j, p_{opt}(j))\}$ entries define a set of 2×2 pivots which has not been selected during the extraction. For example, let us suppose that the selected 2×2 pivots of Figure 1.5.3 correspond to the first choice. Then $(j, p_{opt}(j))$ is the diamond in the left bottom corner and $(\mathcal{C} \setminus \mathcal{M}_s) \setminus (j, p_{opt}(j))$ corresponds to the second choice of pivots. Thus, we have

$$\omega((\mathcal{C} \setminus \mathcal{M}_s) \setminus \{(j, p_{opt}(j))\}) = \frac{\omega(\mathcal{C} \setminus \mathcal{M}_s)}{|a_{jp_{opt}(j)}|} \leq \omega(\mathcal{C} \cap \mathcal{M}_s).$$

because the 2×2 pivots of $\mathcal{M}_s$ has been chosen for each cycle using a maximum weight criterion. We have

$$\begin{aligned} \omega(\mathcal{C}) &= \omega(\mathcal{C} \setminus \mathcal{M}_s) \times \omega(\mathcal{C} \cap \mathcal{M}_s) \\ &\leq \omega(\mathcal{C} \cap \mathcal{M}_s)^2 \times |a_{jp_{opt}(j)}| \end{aligned}$$

On even cycles and cycles of length 1, the weight of $\mathcal{M}^{opt}$ and $\mathcal{M}_s$ are the same. Let $\mathcal{C}_e$ be the set of even cycles and cycles of length 1 and $\mathcal{C}_o$ be the set of odd cycles of length greater than 1. After extending the above inequalities to all the cycles, we have

$$\begin{aligned}
\omega(\mathcal{M}_s) &= \prod_{\mathcal{C}\in\mathcal{C}_e} \omega(\mathcal{C}) \prod_{\mathcal{C}\in\mathcal{C}_o} \omega(\mathcal{C}\cap\mathcal{M}_s)^2 \prod_{(j,j)\in\mathcal{M}_{1\times1}\setminus\mathcal{M}^{opt}} |a_{jj}| \\
&\geq \prod_{\mathcal{C}\in\mathcal{C}_e} \omega(\mathcal{C}) \prod_{\mathcal{C}\in\mathcal{C}_o} \omega(\mathcal{C}) \prod_{(j,j)\in\mathcal{M}_{1\times1}\setminus\mathcal{M}^{opt}} \frac{|a_{jj}|}{|a_{jp_{opt}(j)}|} \\
&\geq \omega(\mathcal{M}_s^{opt}) \prod_{(j,j)\in\mathcal{M}_{1\times1}\setminus\mathcal{M}^{opt}} \frac{|a_{jj}|}{|a_{jp_{opt}(j)}|}.
\end{aligned}$$

The same kind of inequality can be obtained with p_{opt}^{-1}, which proves the first part of the inequality. The second part is true by definition. □

Property 5.4 gives us an *a posteriori* (after the 2×2 detection) cheap bound for estimating how far we are from the optimum. If the matrix satisfies the `MC64` constraints and under the assumption that $|\mathcal{M}_s| = |\mathcal{M}_s^{opt}|$, the bound of Property 5.4 becomes obvious because $\omega(\mathcal{M}_{opt}) = 1$ and $\omega(\mathcal{M}_s) = \omega(\mathcal{M}_{1\times1}) = K_1$. It seems difficult to do better than this bound. In an extreme case, on the matrix $A33(1) + diag(\epsilon)$, with ϵ small, we have $\omega(\mathcal{M}_s) = \epsilon^2$ and $\omega(\mathcal{M}_s^{opt}) = 1$.

1.5.2.3 Recursive improvement of $\mathcal{M}_s$

We have just seen that in the case of a matrix which satisfies the `MC64` constraints $K_1 = \omega(\mathcal{M}_{1\times1})$. In this section, we propose a recursive algorithm to improve the weight of $\mathcal{M}_s$ (*i.e.*, to increase K_1) without changing the already selected 2×2 pivots. Unfortunately we will show by the conclusion of this section that recursive improvement has a limited effect on our test set and so it will not be used. However, the good news is that our weak non-recursive approach leads us to the optimum in most of our test cases.

Algorithm 1.5.1 Recursive detection of 2×2 pivots.

Compute $\mathcal{M}_{2\times2}$, $\mathcal{M}_{1\times1}$, $\mathcal{M}_s$ and $\mathcal{M}_s^c$ using an `MC64` matching on the bipartite graph of A (see Section 1.5.2.1).
repeat
  Build $\mathcal{G}'$, the bipartite graph of A restricted to $\mathcal{M}_{1\times1}\cup\mathcal{M}_s^c$.
  Compute $\mathcal{M}'_{2\times2}$, $\mathcal{M}'_{1\times1}$, $\mathcal{M}'_s$ and $\mathcal{M}_s^{c\prime}$ using an MC64 matching on $\mathcal{G}'$ (see Section 1.5.2.1).
  $\mathcal{M}_s \leftarrow (\mathcal{M}_s \setminus \mathcal{M}_{1\times1}) \cup \mathcal{M}'_{2\times2} \cup \mathcal{M}'_{1\times1}$
  $\mathcal{M}_s^c \leftarrow \mathcal{M}_s^{c\prime}$.
until ($\omega(\mathcal{M}'_s) \leq \omega(\mathcal{M}_{1\times1})$)

At each step of the recursion, we apply the same kind of detection to the submatrix defined by $\mathcal{M}_{1\times1}$; we add the detected 2×2 pivots to $\mathcal{M}_{2\times2}$ and remove the corresponding indices from $\mathcal{M}_{1\times1}$. Algorithm 1.5.1 gives the details of this recursive approach. The termination criteria $\omega(\mathcal{M}'_s) \leq \omega(\mathcal{M}_{1\times1})$ says that if we again call the detection on the subgraph $\mathcal{G}'$ the weight of the symmetric matching will be unchanged. If the recursive algorithm is applied to a matrix which satisfies the `MC64` constraints then the bound K_1

is still $K_1 = \omega(\mathcal{M}_s)$. On the matrix $A33(10^{-1})+diag(\epsilon)$ the application of the recursive algorithm does not improve $\mathcal{M}_s$ if it is initialized to

$$\{(1,2),(2,1),(4,5),(5,4),(3,3),(6,6)\},$$

but, if $\mathcal{M}_s$ is initialized to,

$$\{(3,2),(2,3),(6,5),(5,6),(1,1),(4,4)\},$$

the recursive algorithm finds the symmetric maximum matching

$$\mathcal{M}_s^{opt} = \{(3,2),(2,3),(6,5),(5,6),(1,4),(4,1)\}$$

at step 2 and $\omega(\mathcal{M}_s^{opt}) = K_1 = 10^{-2}$.

Matrix	$N_{odd} - N_1$	Step 0	Step 1	Step 2
BLOCKQP1	20000	-13979	–	–
copter2	416	-194	-124	–
dawson5	465	-205	-141	-140
HELM3D01	1	-0.06	-0.007	–
NCVXQP5	555	-47	–	–
SPMSRTLS	638	-432	-310	–
other 37	–	0	–	–

Table 1.5.1: Symmetric matching weight ($\log_{10} K_1$) when recursive algorithm is applied. The second column lists the number of odd cycle in the matching with more than 1 vertex.

Table 1.5.1 gives the different values of $\log_{10} K_1$ when a symmetric matching is computed recursively. We see that on most of the matrices (on 37 out of 43 to be precise), we have computed the symmetric maximum matching (line other 37) without any use of recursion. On the six matrices where the computed symmetric matching is not necessarily a symmetric maximum weighted matching. we see that the recursive algorithm does not improve K_1 significantly and stops after at most 2 steps. That is why we have decided to avoid the recursive approach. Note that, even if these K_1 values seem to be large, they must be viewed relative to the number of odd cycles greater than 1. Furthermore, K_1 is viewed relative to $\mathcal{M}^{opt}$ and not to $\mathcal{M}_s^{opt}$ the weight of which can be very small compared to $\mathcal{M}^{opt}$ (see for example $A33(\epsilon)$).

1.5.3 Coupling detected pivots and orderings

1.5.3.1 Ordering on the compressed graph

Let $\mathcal{M}$ be a maximum matching on A from which we have obtained a set of 1×1 and 2×2 candidate pivots, $\mathcal{M}_s$. The undirected graph $\mathcal{G}$ associated with the matrix A has a set of vertices V_A, corresponding to the rows (and columns) of A and a set of unordered edges E_A, where the unordered pair $(i,j) \in E_A$ if and only if $a_{ij} \neq 0$.

We define R, the **reduced matrix of** A **relative to** $\mathcal{M}_s$ as the square matrix of order the number of pivots in $\mathcal{M}_s$ whose associated undirected graph will be referred to as the **compressed graph** (it is a generalization of the compressed graph of [15] which only compresses indistinguishable vertices). Each of its vertices $\mathcal{I}_i$ is weighted and either

associated with a row/column of A corresponding to a candidate 1×1 pivot (weight of 1) or a pair of rows/columns $(i_k,\ i_l)$ corresponding to a candidate 2×2 pivot (weight of 2). The edge set E_R consists of the unordered pairs $(\mathcal{I}_i,\ \mathcal{I}_j)$ where there exists an edge in E_A between one of the constituent vertices of $\mathcal{I}_i$ and one of the constituent vertices of $\mathcal{I}_j$. In other words, the candidate 2×2 pivots of $\mathcal{M}_s$ are compressed into one vertex and the union of their adjacency defines the adjacency of the new vertex. The weight of the vertices will be used to initialize the size of the supervariables and to compute the appropriate metric. For example, the external degree of a supervariable $\mathbf{i}$ will be initialized to $ext\_deg(\mathbf{i}) = \sum_{\mathbf{j}\in Adj(\mathbf{i})} |\mathbf{j}|$, where $Adj(\mathbf{i})$ is the set of vertices adjacent to $\mathbf{i}$ in the compressed graph and $|\mathbf{j}|$ is the weight of the vertex $\mathbf{j}$. Obviously, when two supervariables $\mathbf{i}$ and $\mathbf{j}$ are merged – either in the phase of supervariable detection for a greedy ordering or in the coarsening phase for a partitioning, they form a new supervariable of size $|\mathbf{i}| + |\mathbf{j}|$. Note that the rows/columns of A that are not represented in $\mathcal{M}_s$ will not be represented in the reduced matrix.

Algorithm 1.5.2 Main steps of the ordering on the compressed graph.

1 Apply a symmetric `MC64` scaling and get $\mathcal{M}_s$.
2 Compute R, a reduced matrix of A relative to $\mathcal{M}_s$.
3 Set P_{red}, the symmetric permutation returned by an ordering on R.
4 Compute P_{aug}, an extension of P_{red} relative to $\mathcal{M}_s$.
5 Put the components of $\mathcal{M}_s^c$ in the last positions.

Let P_{red} be a symmetric permutation on R. Then it is easy to extend P_{red} to a permutation P_{aug} on A by expanding each component corresponding to a 2×2 composite node to the two rows/columns of A associated with that composite node and by putting the rows/columns of A that were not represented in R ($\mathcal{M}_s^c$ entries) at the end of this permutation. This ensures that zero pivots in $\mathcal{M}_s^c$ will be filled by the previous eliminations if we assume that no numerical cancellation occurs and the matrix is structurally nonsingular. Clearly this expansion can be done in a single pass through P_{red}. Note that P_{aug} is not unique: firstly, when a node corresponding to a 2×2 pivot (i,j) is expanded, we have the choice of taking i or j first in P_{aug}; secondly, when we visit the entries in $\mathcal{M}_s^c$, there is no $a\ priori$ order for placing them in P_{aug}. This approach on the compressed graph generates an ordering that is expected to have preselected good numerical pivots. Algorithm 1.5.2 summarizes the main steps of this preprocessing.

To illustrate the compression and the expansion, let us take the following matrices:

$$A = \begin{pmatrix} 2 & -1 & 1 & 0 & 0 \\ -1 & 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 1 \\ 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \text{ and } R = \begin{pmatrix} 1 & X & X \\ X & 1 & 0 \\ X & 0 & 2 \end{pmatrix}.$$

In A, we detect a 2×2 pivot in rows/columns 3,4 and two 1x1 pivots on the diagonal. We have $\mathcal{M}_s = \{(1,1),(2,2),(3,4)/(4,3)\}$ and $\mathcal{M}_s^c = \{(5,5)\}$. R is a reduced matrix of A relative to $\mathcal{M}_s$ where row/column 3 corresponds to the 2×2 pivot. If $P_{red} = [3,2,1]$ then $P_{aug} = [3,4,2,1,5]$ or $[4,3,2,1,5]$ is an expansion of P_{red}.

1.5.3.2 Constrained ordering

The main principle of the orderings presented in this section and Section 1.5.3.3 is to relax the above preselection of the 2×2 pivots by splitting some of them into two 1×1 pivots before the compression. Moreover, when there is a danger of making a bad numerical decision, we will add some dependency about the precedence between some of the 1×1 pivots coming from split 2×2 pivots. We will propose an algorithm that can be implemented in the context of orderings based on local heuristics like AMD or AMF.

For each 2×2 pivot, $P_{ij} = \begin{pmatrix} a_{ii} & a_{ij} \\ a_{ij} & a_{jj} \end{pmatrix}$, with (i,j) ordered such that $|a_{ii}| \geq |a_{jj}|$ and with $|a_{ij}| = 1$, i is said to be the **leading variable** and j, the **trailing variable**. Let $\theta \in [0,1]$ be a real constant. P_{ij} is said to be:

a **locked pivot** according to θ if and only if $|a_{ii}| \leq \theta$ and $|a_{jj}| \leq \theta$ (if the diagonal entries are too small, it is dangerous to allow elimination of 1×1 pivots),

a **constrained pivot** if and only if $|a_{ii}| > \theta$ and $|a_{jj}| \leq \theta$ (if the value of the leading variable is large enough, it can be eliminated as a 1×1 pivot, but as the value of the trailing variable is too small, it is more safe to add the constraint that the leading variable has to be eliminated before),

a **splittable pivot** if and only if $|a_{ii}| > \theta$ and $|a_{jj}| > \theta$ (if the diagonal entries are large, the leading and trailing variables can be eliminated as 1×1 pivots independently).

In our experiments we set $\theta = 10^{-2}$.

The set of MC64SYM detected 2×2 pivots is separated into three sets, LP_θ, CP_θ and SP_θ, the set of locked pivots, constrained pivots and splittable pivots respectively. Note that when $\theta = 0$, locked pivots correspond to *oxo* pivots (two zeros on the diagonal), constrained pivots to *tile* pivots (one zero on the diagonal) and splittable pivots to full 2×2 pivots. When $\theta = 1$, none of the 2×2 pivots is split and the ordering will behave as the ordering on the compressed graph. The more θ decreases, the more pivots are split and the more the risks of making a bad decision increase. In the rest of our discussion, we will omit the θ from our notation.

During the ordering, we manipulate two kinds of (super)variables : **free (super)variables** and **constrained (super)variables**. At the beginning of the ordering, a supervariable i is said to be a free supervariable if and only if one of the following conditions hold:

(1) i belongs to a splittable pivot,

(2) i is the leading variable of a constrained pivot,

(3) i belongs to $\mathcal{M}_{1\times1}$,

(4) i is a locked pivot.

Otherwise, a supervariable i is said to be a constrained if it appears as the trailing variable of a constrained pivot.

During the pivot selection, there are two main rules:

(R1) a free supervariable can be eliminated whenever we want (it does not depend on the elimination of another one),

(R2) a constrained supervariable can be eliminated if and only if a free pivot with which it is associated has already been eliminated.

The second rule is equivalent to marking constrained supervariables as free as soon as (R2) is satisfied. Thus, during the ordering, constrained supervariables can become free. If i is the leading variable of a constrained 2×2 pivot, we will say that i **releases** j where j is the variable associated with i in this pivot (it is a sufficient condition to make the supervariable j free). The free supervariables that correspond to entries in $\mathcal{M}_{1\times 1}$ or to locked pivots can be eliminated but do not release any constrained supervariables.

At each step of the ordering, FV and CV will be used to denote respectively the set of supervariables that can be eliminated and the set of pivots that cannot be eliminated. For each free supervariable i belonging to a constrained pivot, we define $assoc(i)$, its associated constrained supervariable. For other supervariables $assoc(i) = \emptyset$.

Algorithm 1.5.3 Constrained ordering scheme.

Determine free and constrained supervariables according to $\mathcal{M}_s$.
$CV \leftarrow \{$ constrained supervariables $\}$ and $FV \leftarrow \{$ free supervariables $\}$.
while there are uneliminated supervariables **do**
  $i \leftarrow arg\min_{p\in FV} metric(p)$
  Do symbolic elimination of i.
  $FV \leftarrow (FV \cup \{assoc(i)\}) \setminus \{i\}$
  $CV \leftarrow CV \setminus \{assoc(i)\}$
end while

Algorithm 1.5.3 describes our constrained ordering. At each step of the symbolic elimination, we select the best pivot in the set FV. Moreover, when the leading supervariable of a constrained pivot is selected, its associated constrained supervariable is released (it is inserted in the FV set and removed from the CV set). Intuitively, if the leading part of a constrained pivot is eliminated then it is possible that the modified value of the trailing supervariable becomes large because our scaling ensures that $a_{pq} = 1$ and $a_{pp} \leq 1$, and thus that the corresponding entry becomes numerically acceptable. This constrained ordering has been implemented with the `AMD` and `AMF` orderings. As in the ordering on the compressed graph, we put the variables in $\mathcal{M}_s^c$ at the end of the permutation.

1.5.3.3 Relaxation of constrained ordering

This approach uses the same terminology as the previous one. Here we relax the selection of the pivots, that is, we enlarge the set of free pivots and the possibilities for releasing the constrained variables. We fix a dropping threshold $\theta_{drop} \in [0, 1]$ and build a matrix $C = (c_{ij})$ that is called the **constraint matrix**. This matrix contains a subset of the entries in A and so its size is bounded by the size of A. This matrix together with $\mathcal{M}_s$ will be used to define the constrained and free supervariables and the relations between them (who releases whom). Firstly the rows and columns of C that correspond to indices that appear in locked pivots are set to 0 (it corresponds to rows/columns 6 and 7 of Figure 1.5.6). Then

$$A = \begin{pmatrix} 1 & 1 & 0 & \epsilon & 0 & 1 & 0 \\ 1 & \epsilon & 0 & 1 & 1 & \epsilon & 0 \\ 0 & 0 & 1 & 1 & 0 & \epsilon & 1 \\ \epsilon & 1 & 1 & \epsilon & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & \epsilon & \epsilon & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}, C = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 1.5.6: A matrix A and its constraint matrix C ($\epsilon < \theta_{drop}$). The symmetric matching of A is composed of two constrained pivots in positions $(1,2)$ and $(3,4)$, of a 1×1 pivot in position $(5,5)$ and of a locked pivot in position $(6,7)$.

the rest of the entries c_{ij} of the constrained matrix are set to 1 if and only if $a_{ij} \geq \theta_{drop}$, otherwise they are set to 0 (the ϵ entries are removed in Figure 1.5.6). In our experiment we take $\theta_{drop} = 0.9$ (see Section 1.6). C describes the dependency among the free and constrained supervariables: i can release j if and only if $c_{ij} \neq 0$. The above construction prevents a locked pivot from releasing a constrained supervariable.

Each nonzero off-diagonal entry of C defines a potential 2×2 pivot. A supervariable i is free if and only if one of the following conditions hold:

(1) i appears in the indices of a potential splittable pivot in C,

(2) i is the leading variable of a potential constrained pivot in C,

(3) i corresponds to an entry in $\mathcal{M}_{1\times1}$,

(4) i is a locked pivot.

Here again the supervariables of $\mathcal{M}_{1\times1}$ and the locked pivots are systematically included in the set of free variables. For example, in Figure 1.5.6, the variables 1, 3, 5 and the locked pivots $(6,7)$ are free and the variables 2 and 4 are constrained. During the ordering, a constrained pivot j can be selected if and only if it is reachable from eliminated supervariables in C. In other words it is possible that it has been updated by large enough entries. For example, in Figure 1.5.6, let us suppose that the ordering has selected the variable 5 as pivot. Since $c_{5,2} \neq 0$, the variable 2 becomes free and can be selected at the next step of the ordering.

Algorithm 1.5.4 describes this approach. At each step, the entries in CV that satisfy the above condition are added to FV. We have implemented both the constrained and the relaxed constrained approaches with an AMD/AMF ordering, where small changes are made to allow supervariable detection and locked pivot supervariables are initialized with a weight of 2. The total overhead of the manipulation of these sets is $\mathcal{O}(nnz(C))$ which is negligible with respect to the total complexity of the ordering, shown in [68] to be $\mathcal{O}(n \times nnz(A))$.

We have presented two alternatives to the ordering based on the compressed graph: a constrained ordering and a relaxed constrained ordering. We expect to have less fill-in using the (relaxed) constrained ordering than using the ordering on the compressed graph. However, we expect that the prediction of the ordering on the compressed graph

Algorithm 1.5.4 Relaxed constrained ordering scheme.

Define the free supervariables and the constrained supervariables according to the constraint matrix C and $\mathcal{M}_s$.
$CV \leftarrow \{$ constrained supervariables $\}$ and $FV \leftarrow \{$ free supervariables $\}$.
while there are uneliminated supervariables **do**
  $i \leftarrow arg\min_{p \in FV} Metric(p)$
  Do symbolic elimination of i
  $FV \leftarrow (FV \cup C_{i.}) \setminus \{i\}$
  $CV \leftarrow CV \setminus C_{i.}$
end while

will be more exact than the prediction of the (relaxed) constrained ordering where there is a greater risk of making a bad numerical decision. We will see in Section 1.6 that these intuitions are verified and that the constrained and the relaxed constrained ordering have a similar behaviour.

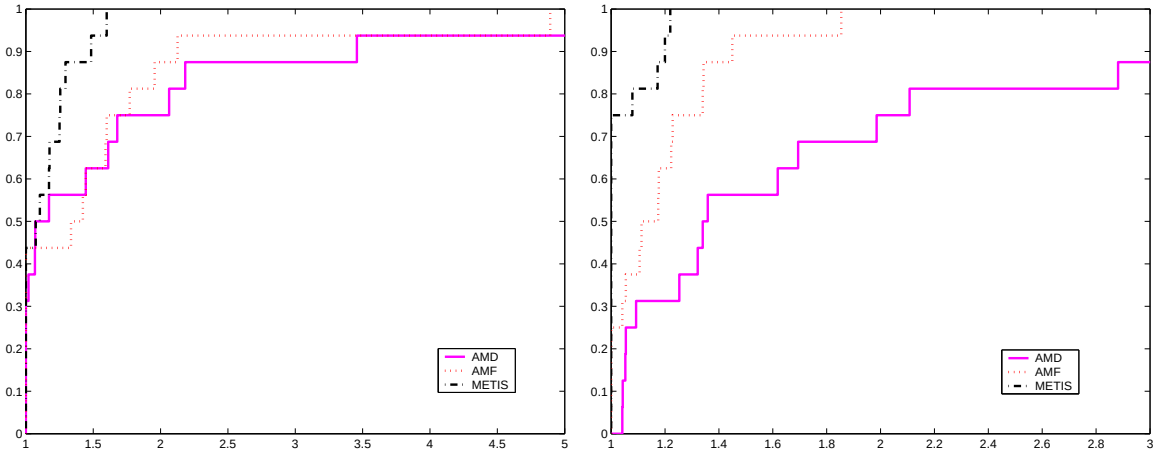
1.6 Experimental results

We now consider the effect of our pivoting strategies on MA57. We have four approaches from the above strategies.

- MA57_1 refers to the approach using the MC64SYM scaling coupled with an ordering which can be AMD, AMF or METIS.
- MA57_2 refers to the approach using the MC64SYM scaling, the preselection of the 2×2 pivots and the ordering (AMD, AMF or METIS) based on the compressed graph (Section 1.5.3.1).
- MA57_3 refers to the approach using the MC64SYM scaling, the preselection of the 2×2 pivots and the constrained ordering scheme (Section 1.5.3.2). This strategy is compatible with AMD or AMF.
- MA57_4 refers to the approach using the MC64SYM scaling, the preselection of the 2×2 pivots and the relaxed constrained ordering scheme (Section 1.5.3.3). This strategy is compatible with AMD or AMF.

METIS is called using the routine METIS_NodeWND in the MA57_2 approach in order to take into account the size of the supervariables in the metrics while trying to find a balanced partition. We use the routine METIS_NodeND in the MA57_1 approach since in this case all the supervariable have a size of 1. In MA57_3 and MA57_4, a threshold of $\theta = 10^{-2}$ is used to determine if the diagonal entries correspond to free variables (see Section 1.5.3.2). We tested different values and remark that increasing it from 10^{-2} degrades the fill-in returned by the ordering (there are too many constraints and not enough free variables), that decreasing it too much from 10^{-2} degrades the memory prediction (on some matrices it did not guarantee good numerical pivots) and did not significantly decrease the fill-in and the number of operations. In MA57_4, a threshold of $\theta_{drop} = 0.9$ is used to determine the constraint matrix (see Section 1.5.3.3). Here also, we tested different values with the same conclusions as above.

1.6.1 General symmetric indefinite matrices (set 3)



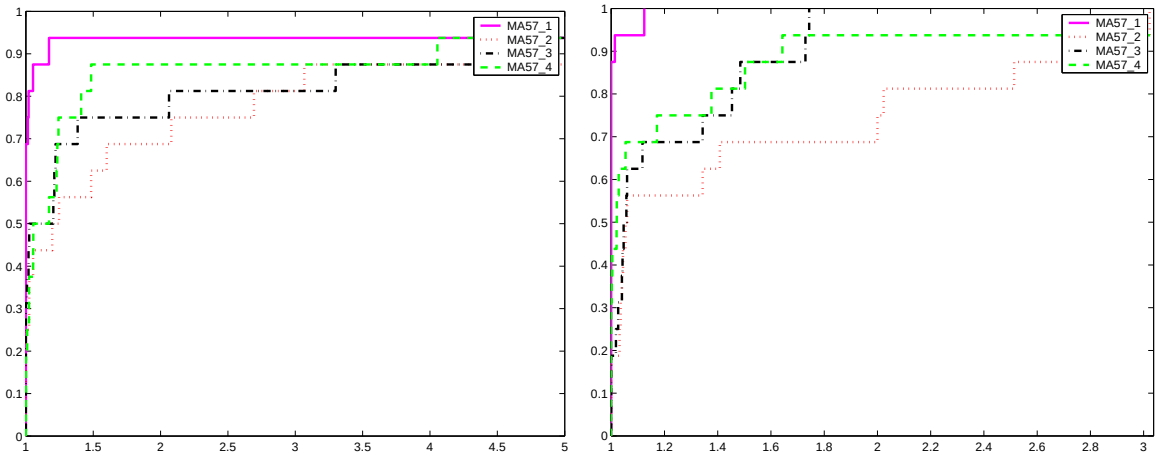
(1.6.1.a) CPU factorization time profile. (1.6.1.b) Memory profile.

Figure 1.6.1: AMD, AMF, METIS comparison (set 3). The MC64SYM scaling is used.

We first determine the best approach on set 3. We will only give details about results for the METIS based ordering for three reasons.

Firstly, METIS is clearly better than AMD and AMF in terms of CPU factorization time (see Figure 1.6.1.a) and memory (see Figure 1.6.1.b). On 75% of the problems, the approach based on METIS is the best in term of memory and its factorization time is within a factor 1.3 of the best.

Secondly, the relative behaviour between AMD+MA57_1 and AMD+MA57_2 and between AMF+MA57_1 and AMF+MA57_2 are similar to the relative behaviour between METIS+MA57_1 and METIS+MA57_2 (see Figure 1.6.2).



(1.6.2.a) AMD ordering. (1.6.2.b) AMF ordering.

Figure 1.6.2: CPU factorization time profile, AMD and AMF ordering (set 3). The MC64SYM scaling is used.

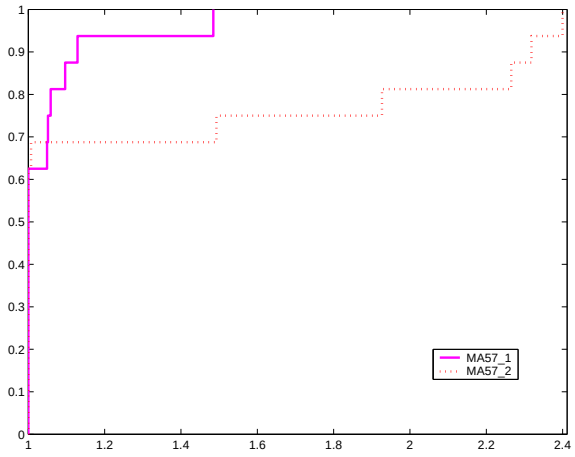
Finally, on this set, Figure 1.6.2 shows that the constrained ordering of Section 1.5.3.2 and the relaxed constrained ordering of Section 1.5.3.3 do not significantly improve the CPU factorization time and do not decrease the size of the factors compared with the

approach on the compressed graph (Section 1.5.3.1). On the contrary, we will see in the next section that they clearly improve the factorization on set 2.

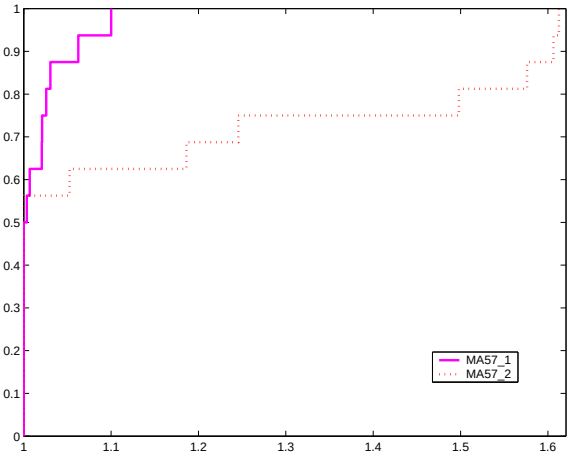
AMD	AMF	METIS
178 sec	130 sec	78 sec

Table 1.6.1: Cumulative factorization time on set 3 except c-71.

Figure 1.6.3.a shows that MA57_1 is faster than MA57_2 on set 2. This execution time difference is due to the fill-in and the number of operations. Indeed, the approach on the compressed graph merges some rows of the initial matrix and the METIS ordering is called on a coarsened graph. Thus, the MA57_2 pretreatment implies non-negligible constraints on the ordering because of the $a\ priori$ selection of the 2×2 pivots. These constraints severely degrade the quality of the ordering. In particular, they increase the memory requirement (see Figure 1.6.3.b).

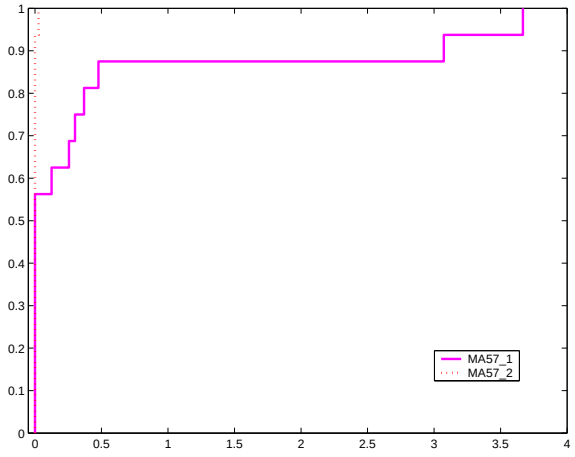


(1.6.3.a) CPU factorization time profile.

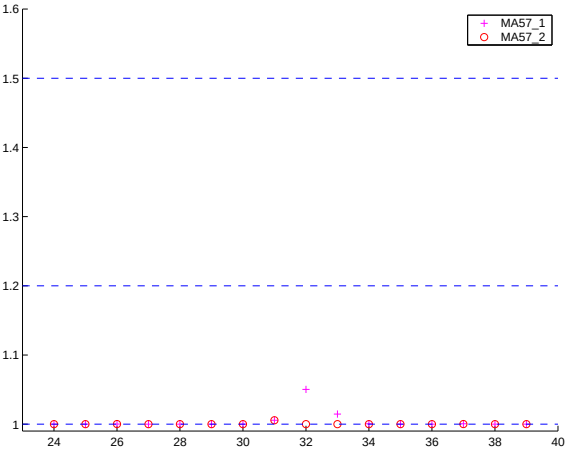


(1.6.3.b) Memory profile.

Figure 1.6.3: CPU factorization time and memory (METIS, set 3).



(1.6.4.a) Number of delayed pivots profile (log_{10} scale for x-axis).



(1.6.4.b) Ratio between used memory and memory prediction of analysis. Problems sorted by Id on the x-axis.

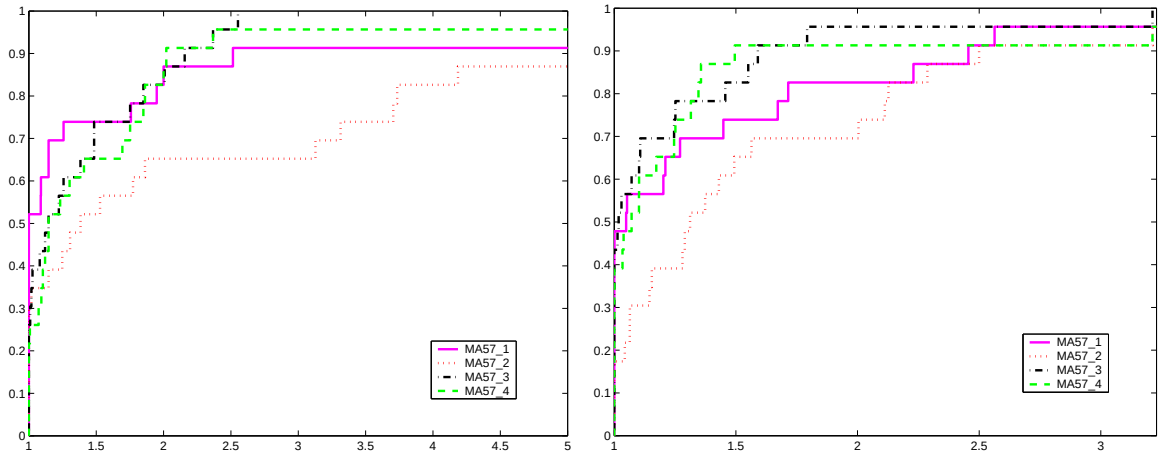
Figure 1.6.4: Quality of analysis prediction (METIS, set 3).

The MA57_2 analysis succeeds in selecting good pivots for the factorization and delays less pivots than MA57_1 (see Figure 1.6.4.a). Furthermore, the memory estimation is accurate for both approaches: the ratio between the predicted and the used memory remains close to 1 (see Figure 1.6.4.b).

1.6.2 Augmented systems (set 2)

We will now study the behaviour of the MA57 factorization when it is coupled with the three orderings AMD, AMF and METIS on set 2 of our test matrices. We will see that the influence of our preprocessing depends on the ordering with which it is associated. We first do relative comparisons between the the four AMD based codes, between the two METIS based codes and between the four AMF based codes. Then we select the best approaches and discuss them further.

1.6.2.1 AMD based approaches

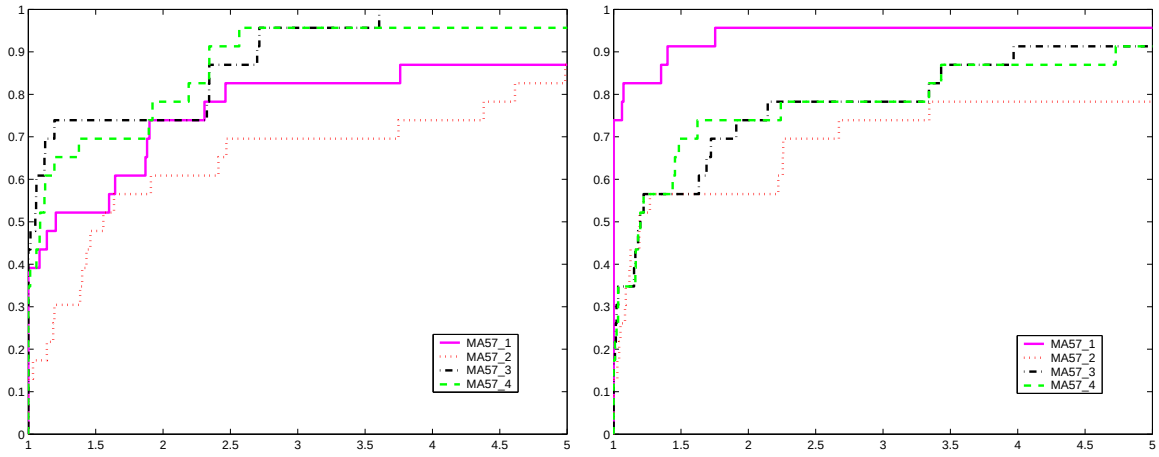


(1.6.5.a) CPU factorization time profile. (1.6.5.b) Memory profile.

Figure 1.6.5: CPU factorization time and memory (AMD, set 2).

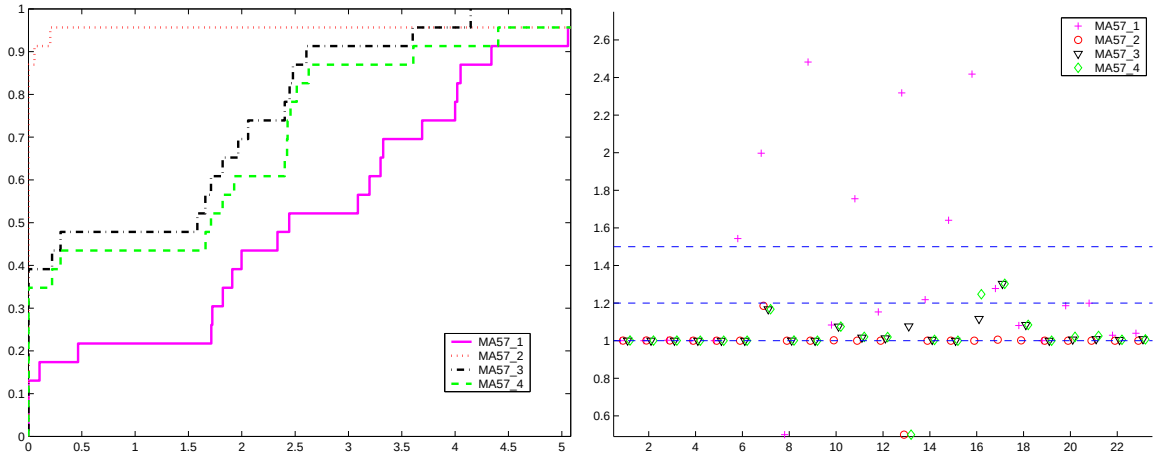
MA57_1, MA57_2 and MA57_4 fail on one matrix (they exceeded the CPU time limit on NCVXQP7) but MA57_3 does not fail. Figure 1.6.5.a shows that, in terms of CPU factorization time, MA57_1 is within a factor of 1.25 of the best 75% of the time, MA57_3 and MA57_4 are comparable.

MA57_2 can be far from the best ordering because of the constraints that we impose in the analysis phase. That is why it performs more computation and needs more memory than the other codes (see Figures 1.6.6.a and 1.6.5.b) and thus is also slower. MA57_3 and MA57_4 do less operations than MA57_1. Thus the total number of operations cannot explain why MA57_1 is faster on 75% of the problems. Further examination shows that MA57_3 and MA57_4 perform more assembly operations than MA57_1 (see Figure 1.6.6.b). Assembly operations involve indirect addressing whereas elimination operations call level 3 BLAS dense kernels, so that assembly operations are slower than eliminations, which explains why MA57_3 and MA57_4 are slower than MA57_1.



(1.6.6.a) Operations profile. (1.6.6.b) Assembly operations profile.

Figure 1.6.6: Number of floating-points operations (AMD, set 2).

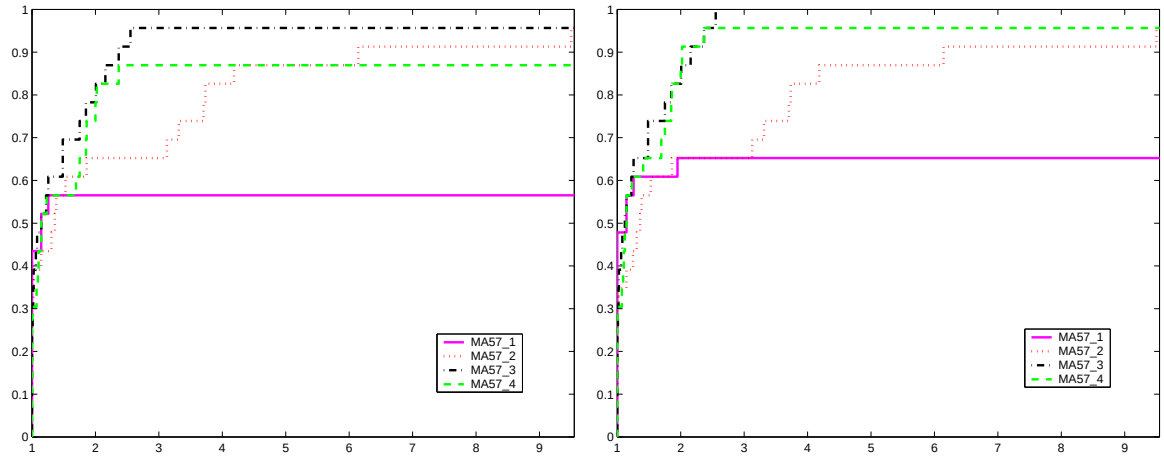


(1.6.7.a) Number of delayed pivots profile (log_{10} scale for x-axis). (1.6.7.b) Ratio between used memory and memory prediction of analysis. Problems sorted by Id on the x-axis; ratio of 0.5 is used as a dummy value to indicate failure.

Figure 1.6.7: Quality of analysis predictions (AMD, set 2).

The three new approaches clearly improve the reliability of the analysis predictions. MA57_2 decreases the number of delayed pivots as shown in Figure 1.6.7.a. Moreover MA57_2, MA57_3 and MA57_4 improve the memory estimations. If a relaxation parameter of 50% is used between the analysis and the factorization the number of failures decreases from 8 to 1, 0 and 1 respectively. (see Figure 1.6.7.b).

With respect to the above aspects (CPU factorization time, memory, number of delayed pivots and quality of the analysis prediction), MA57_3 seems to be the best if compromises have to be done between CPU time and memory for an AMD based ordering (see Figure 1.6.8). MA57_1 remains the fastest on most of the problems if a large fixed amount of memory can be allocated for the factorization. That is why, concerning the AMD based ordering, we keep the approach with only the MC64SYM scaling (MA57_1) and the approach with a constrained ordering (MA57_3) for our final comparison.

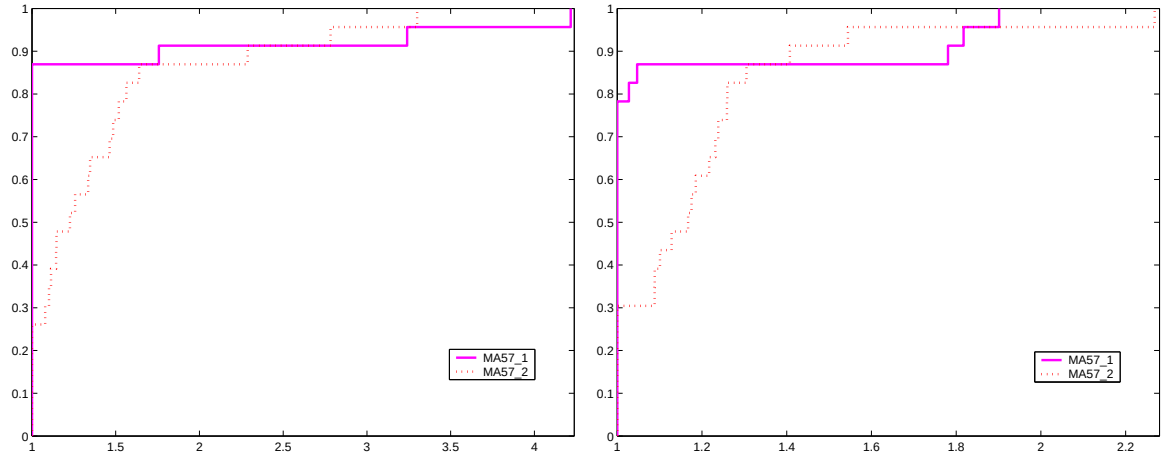


(1.6.8.a) CPU factorization time profile with 20% memory relaxation.

(1.6.8.b) CPU factorization time profile with 50% memory relaxation.

Figure 1.6.8: CPU factorization time profile with relaxed memory (AMD, set 2).

1.6.2.2 MeTiS based approaches



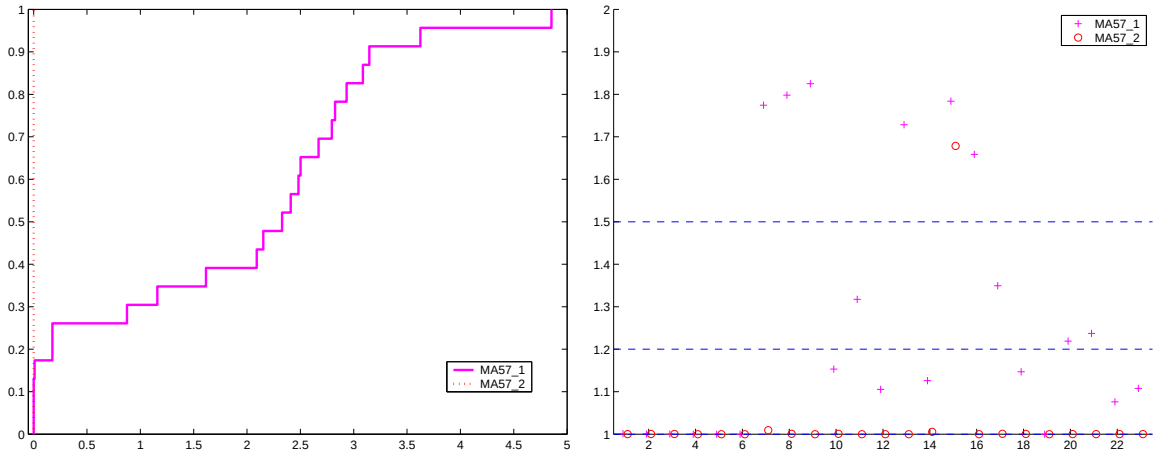
(1.6.9.a) CPU factorization time profile.

(1.6.9.b) Memory profile.

Figure 1.6.9: Influence on CPU factorization time and memory (METIS, set 2).

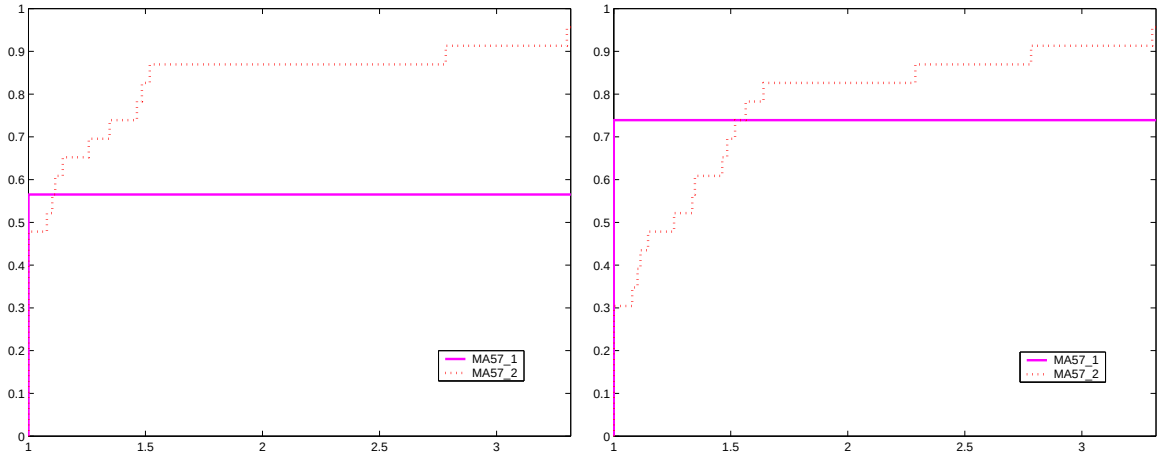
When our approaches are coupled with METIS, we did not get any failures. Moreover, MA57_1 is faster 85% of the time, but MA57_2 is not too far behind in terms of factorization time and memory usage (see Figure 1.6.9.a and 1.6.9.b).

Figure 1.6.10 shows that MA57_1 does not give an accurate prediction for the subsequent factorization. There are 10 and 6 matrices over the 20% and 50% limits respectively. This can be explained by the huge number of delayed pivots (MA57_1 delays 10000 times more pivots than MA57_2 in the worst cases, see Figure 1.6.10.a). On the contrary, MA57_2 exceeds the 20% limit only once and otherwise the ratio between the memory predicted and the memory needed is always near to one. This better memory estimation is clearly shown by the profile of Figure 1.6.11 where MA57_2 is faster than MA57_1 with 20% memory relaxation. With respect to the above comments, we keep the two METIS approaches for our final comparison.



(1.6.10.a) Number of delayed pivots profile (log_{10} scale for x-axis). (1.6.10.b) Ratio between used memory and memory prediction of analysis. Problems sorted by Id on the x-axis.

Figure 1.6.10: Quality of analysis predictions (METIS, set 2).

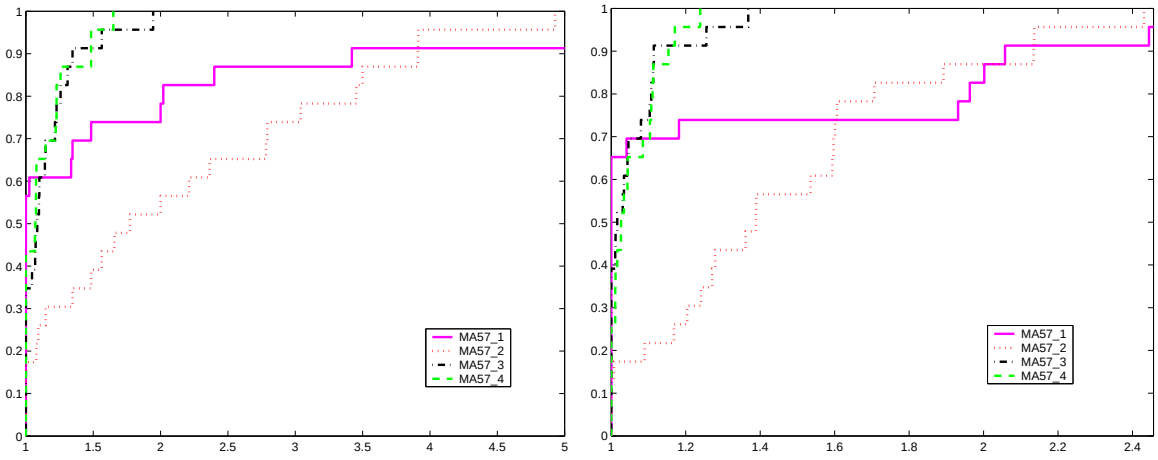


(1.6.11.a) CPU factorization time profile with 20% memory relaxation. (1.6.11.b) CPU factorization time profile with 50% memory relaxation.

Figure 1.6.11: CPU factorization time profile with relaxed memory (METIS, set 2).

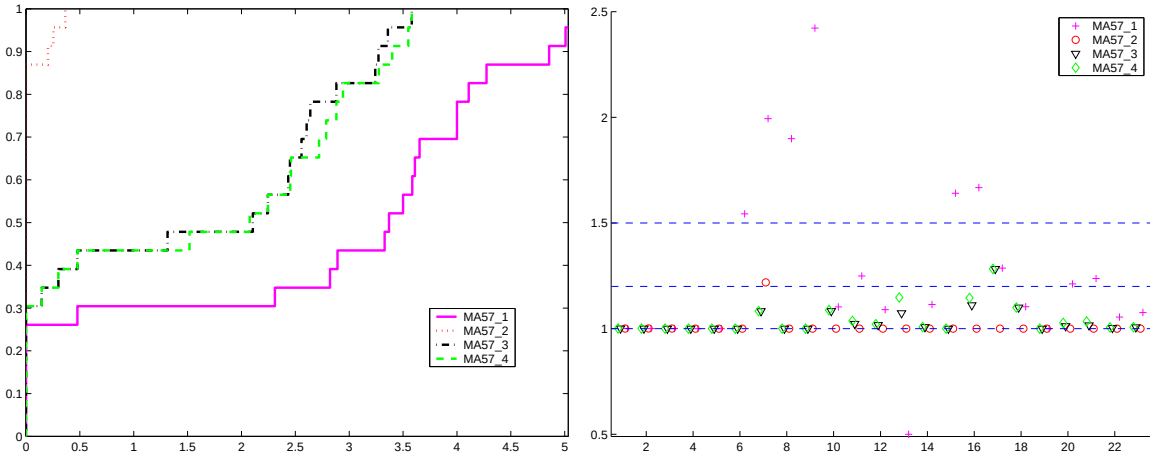
1.6.2.3 AMF based approaches

We observe a similar relative behaviour between MA57_1 and MA57_2 with an AMF based ordering instead of an AMD based one. We did the experimental observation that the main advantage of an AMF based ordering is that MA57_3 and MA57_4 are not penalized by a large number of assembly operations. Thus MA57_3 and MA57_4 are the fastest approaches (Figure 1.6.12.a) and need less memory than the other approaches (Figures 1.6.12.b). Moreover they decrease the number of delayed pivots (Figure 1.6.13.a) and improve the memory estimation (Figure 1.6.13.b). They decrease the number of failures from seven with MA57_1 to none if 50% relaxation is used between analysis and factorization. That is why they are also the best approaches in terms of CPU factorization time with a fixed memory relaxation as shown in Figure 1.6.14. For AMF, we only keep MA57_4 for our final comparison.



(1.6.12.a) CPU factorization time profile. (1.6.12.b) Memory profile.

Figure 1.6.12: Influence on CPU factorization time and memory (AMF, set 2).



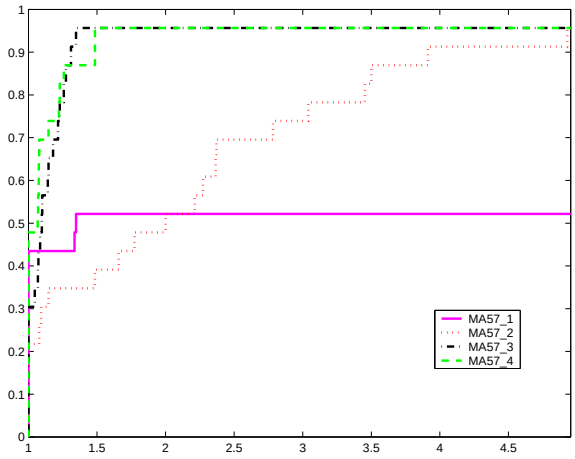
(1.6.13.a) Number of delayed pivots profile (log_{10} scale for x-axis). (1.6.13.b) Ratio between used memory and memory prediction of analysis. Ratio of 0.5 is used as a dummy value to indicate failure; problems sorted by Id on the x-axis.

Figure 1.6.13: Quality of analysis predictions (AMF, set 2).

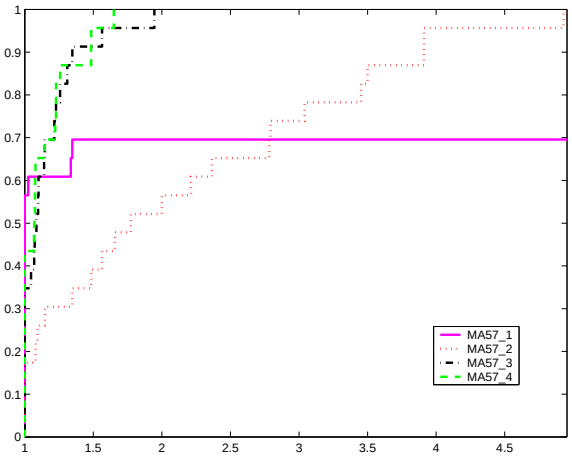
1.6.2.4 Best approaches discussion

In this section, we discuss the best approach for set 2. We have kept five codes for the comparison because of the earlier discussion: MA57_1/3 with AMD, MA57_1/2 with METIS and MA57_4 with AMF.

Figure 1.6.15 shows that the AMD based approaches are the slowest and the most memory consuming. The pure METIS based approach is the best on most of the problems and the two other approaches (MA57_2(METIS) and MA57_4(AMF)) are comparable. Figure 1.6.16 presents the factorization time if a relaxation parameter is used between analysis and factorization. With 20% relaxation MA57_2(METIS) and MA57_4(AMF) are clearly the best approaches (see Figure 1.6.16.a). With 50% relaxation MA57_1(METIS) is the fastest on many problems and MA57_4(AMF) is the most robust (see Figure 1.6.16.b).

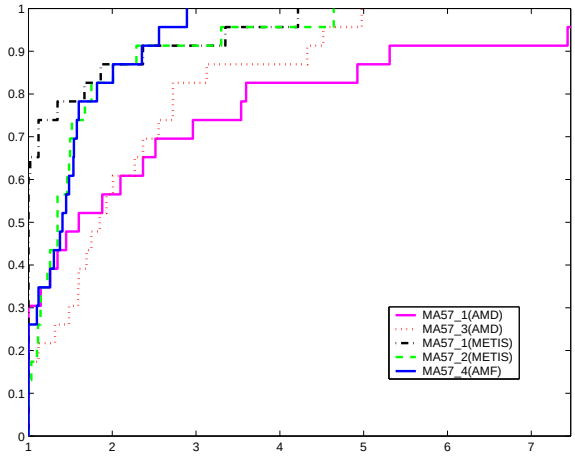


(1.6.14.a) CPU factorization time profile with 20% memory relaxation.

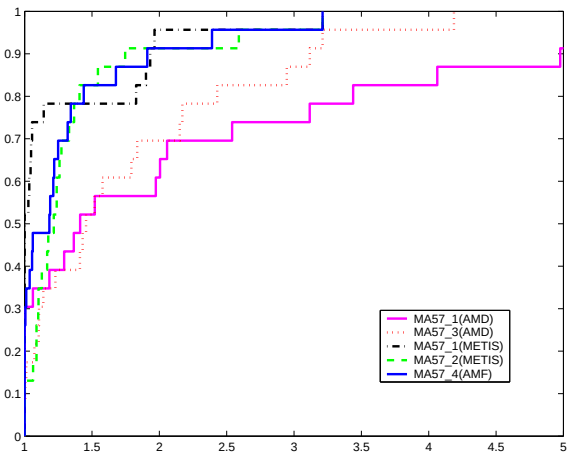


(1.6.14.b) CPU factorization time profile with 50% memory relaxation.

Figure 1.6.14: CPU profile with relaxed memory (AMF, set 2).



(1.6.15.a) CPU factorization time profile.



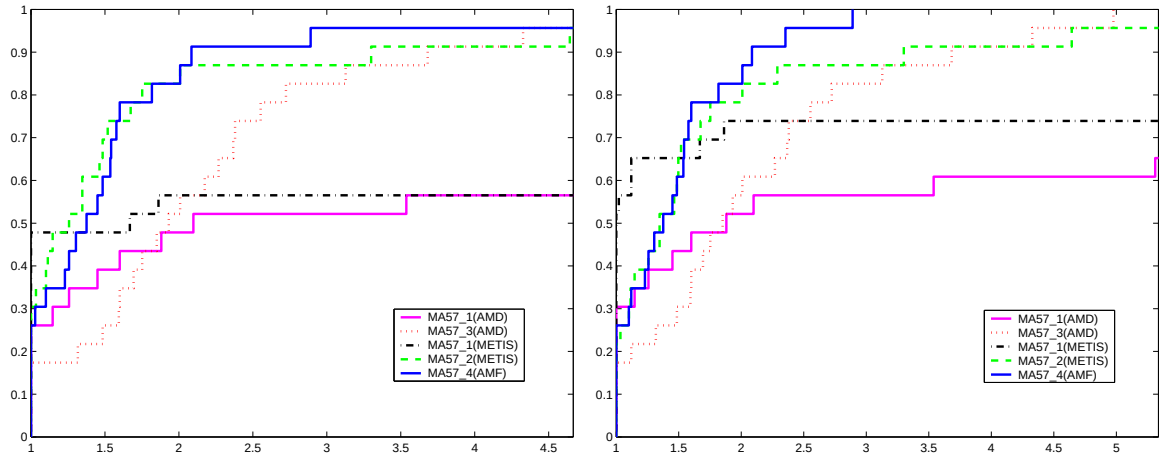
(1.6.15.b) Memory profile.

Figure 1.6.15: AMD/METIS/AMF comparison on set 2.

Thus, if a large amount of memory is available, the approach with only MC64SYM scaling and METIS is sufficient. But, if the memory of the factorization needs to be estimated, we recommend the use of the MA57_4+AMF approach.

1.7 Conclusions

We have shown how MC64 can be used when the matrix is symmetric to effect a symmetric scaling and identify potential 2×2 pivots. We have observed that the use of an appropriate scaling (the MC64 symmetrized scaling, MC64SYM) solves many computational difficulties. We also noticed that MC77 can sometimes be a good alternative and can benefit from its cheap cost. Perhaps this scaling has to be better understood before becoming a default approach. In particular, a fixed convergence rate is difficult to obtain since it is strongly dependent from the structure of the problem.



(1.6.16.a) CPU factorization time profile with 20% memory relaxation.

(1.6.16.b) CPU factorization time profile with 50% memory relaxation.

Figure 1.6.16: AMD/METIS/AMF comparison (set 2): CPU factorization time profile with relaxed memory.

The performance of MA57 depends very much on the nature of the matrix, but we have shown that it can benefit from this preprocessing, sometimes with no change, or only minor changes to the analysis. Another benefit of our work is that the analysis phase gives a better indication of the work and storage required by the subsequent factorization.

In Chapter 2, we will give partial answers to the following question. How can we adapt our approaches to a solver like MA47? Indeed, MA47 is designed for augmented systems and will $a\ priori$ do less computation, because it can manage the *oxo* and *tile* pivots better.

Chapter 3 tries to design a code with static 2×2 and 1×1 pivoting. Static pivoting can address a large variety of problems in the unsymmetric case when it is coupled with MC64 preprocessing and has the advantage of giving exact memory estimation. It is also more friendly for a parallel distributed implementation. Can we adapt the unsymmetric static pivoting to the symmetric indefinite case and does it address a large range of symmetric indefinite matrices?

Constrained ordering and partition?

In our future work, we will try to decrease the fill-in in the MA57_2(METIS) analysis while keeping a stable factorization. We have succeeded in decreasing the number of operations in AMD and AMF with a (relaxed) constrained ordering. Fill-in may be lower if we extend this idea to have a tighter coupling with METIS.

The main principles of constrained and relaxed constrained ordering are that a constrained variable must be released before being eliminated and two variables of a locked pivot are chosen at the same time. Algorithms to generate valid permutations have been proposed for approaches based on local heuristics (AMD/AMF).

In the context of an ordering produced by a partitioning scheme, it is more difficult to generate permutations which are compatible with the (relaxed) constraints. The design of this kind of algorithm is not obvious and is not in the scope of this present study.

Nevertheless, we give a more graph oriented formulation of the problem in order to understand the difficulties better.

At each step of the top-down approach, we have to find a valid partition $\Phi, \Omega_1, \ldots, \Omega_K$ of the vertices where each of the Ω_i are disconnected domains and Φ is the interface. We want to respect the conditions that a constraint variable can be eliminated only after it has been freed and that the variables of locked pivots are eliminated at the same time. More formally the partition is valid if the following two conditions are satisfied:

$$\begin{aligned} &\text{for all constrained variables, } j \in \Omega_p, \\ &\qquad \text{there exists a free variable } i \in \Omega_p \text{ which releases } j, \end{aligned} \tag{1.7.1}$$

$$\text{for each locked pivot } (i,j), i \in \Omega_p \Leftrightarrow j \in \Omega_p. \tag{1.7.2}$$

To satisfy condition 1.7.2, a straightforward solution is to compress the edges and the vertices associated with the locked pivots before doing the ordering. A constrained variable can be either in Φ or in Ω_p if it satisfies condition 1.7.1. This condition is more complicated to implement.

In the context of a constrained ordering, the $\mathcal{M}_{2\times 2}$ set is used to define the initial type of variables and the rules to release the constrained variables. One approach could be to maintain these constraints during the coarsening and the refinement phases. Another approach could be to compute a partition and to move some constrained variables from the domains to the interface. Refinement could be also optionally applied with respect to Conditions (1.7.1) and (1.7.2).

In the context of a relaxed constrained ordering in which the way of releasing a variable is not unique and where a constrained variable can become free and later release another constrained variable and so on, Condition (1.7.1) can be more difficult to verify (contrary to the bottom-up approach, in the top-down approach we do not have the information about who will free whom). In fact, if we want to benefit totally from the freedom offered by the relaxed constraints, we have to look for the paths which start with a free variable. Let F_0 be the initial set of free variables. For any sets of vertices S and L, and a symmetric matrix B, we denote by $Reach_{B(S)}(L)$ the set of vertices which can be reached from vertices of L through paths included in the undirected graph of B restricted to vertices of S. Perhaps it is easier to think of maintaining Condition (1.7.1) using the following equivalent condition in the context of a relaxed constrained ordering:

$$\forall j \in \Omega_p, \text{ constrained variable, } j \in Reach_{C(\Omega_p)}(F_0 \cap \Omega_p). \tag{1.7.3}$$

Chapter 2

Preprocessings of augmented systems for sparse direct solvers.

What should work, does not and could ?

2.1 Introduction

We study preprocessing techniques for the LDL^T factorization of symmetric augmented systems where L is a lower triangular matrix and D is a block diagonal matrix with 1×1 and 2×2 blocks. In Chapter 1 we presented techniques based on maximum weighted matching to preprocess symmetric indefinite matrices in the context of a general symmetric indefinite solver, `MA57`. We examine ways to adapt these techniques in a solver that detects zeros blocks which appear during the factorization of augmented systems.

As in the previous chapter, we make extensive use of routines from HSL [72] (see Table 1.3.4).

In Section 2.2, we first describe the main features of `MA47` and the main differences with `MA57`. Section 2.3 introduces algorithms to preselect good numerical pivots and to preserve the sparsity as much as possible in `MA47`. In Section 2.4, we first identify the best `MA47` approaches and then compare them with `MA57`.

2.2 Multifrontal codes and augmented systems

In this study A is a sparse symmetric matrix which has the structure of an augmented system and which has been symmetrically scaled by the `MC64SYM` scaling. Thus its entries are in $[-1, 1]$, and it has the form

$$PAP^T = \mathcal{K}_{H,A} = \begin{pmatrix} H & B \\ B^T & 0 \end{pmatrix}.$$

During the factorization of an augmented system three kinds of 2×2 pivots can appear. *oxo* pivots have a pattern of the form

$$O_{ij} = \begin{array}{c} \begin{array}{cc} i & j \end{array} \\ \begin{pmatrix} 0 & X \\ X & 0 \end{pmatrix} \begin{array}{c} i \\ j \end{array} \end{array} \tag{2.2.1}$$

tile pivots have a pattern of the form

$$T_{ij} = \begin{array}{c} \begin{array}{cc} i & j \end{array} \\ \begin{pmatrix} X & X \\ X & 0 \end{pmatrix} \begin{array}{c} i \\ j \end{array} \end{array} \tag{2.2.2}$$

and *full* pivots, F_{ij} have a full pattern. The use of these kinds of pivots is detailed in [47, 48]. In the following, when the indices i and j are used to represent the variables of a 2×2 pivot, i is the first index and j the second one if the pattern of the pivot matches the forms (2.2.1) or (2.2.2). At each step of Gaussian elimination, R_i and R_j will refer to the structure of row i and row j of the reduced submatrix.

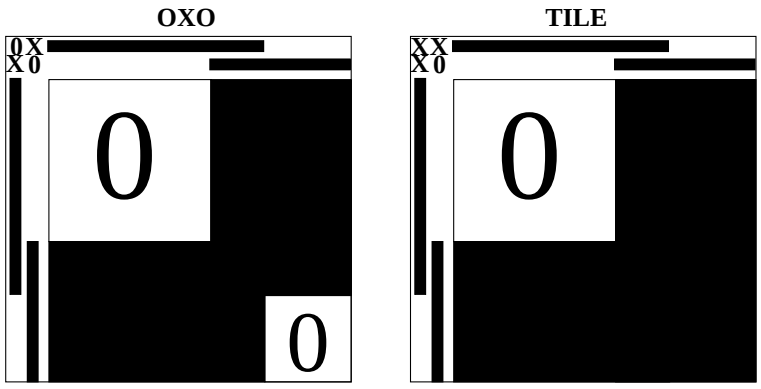


Figure 2.2.1: Zero blocks of *oxo* and *tile* pivots.

During Gaussian elimination, *oxo* and *tile* pivots generate contribution blocks with 2 and 1 blocks of zeros respectively (see Figure 2.2.1). Thanks to a generalization of the elimination tree and of the quotient graph, MA47 can manage this kind of pivot. Each contribution block is divided into smaller full subblocks and dense level 3 BLAS kernels are used on these subblocks. To respect these structures, assemblies of contributions from the children are only done partially. That is why an element may be partially absorbed and remain active during more than one generation.

MA57 is not designed to manage these kinds of blocks. Let us consider a node of the elimination tree. The zero blocks of *oxo* or *tile* pivots of its children are explicitly assembled. If we are lucky, there is numerical cancellation and some entries of L can be dropped. Indeed the fully summed rows of a node can contain zero columns which can be removed from the factors after having computed the contribution block.

For example, in Figure 2.2.2, the zero entries in column 4 and rows 3 and 10 can be removed. This dropping can be done to have a faster solve phase and a cheaper storage of the factors. But it is not trivial to remove them before the elimination operations. In an extreme case (DTOC matrix), these zeros correspond to 67% of the factors when the METIS ordering is used. It is important to note that this dropping does not change the number of flops for the factorization. Because, in a multifrontal approach, information can only be passed from child to parent, we have to compute the contribution block before dropping any entries. We could remove them before the Schur computation but it would involve a lot of indirect addressing and, compared to MA47, MA57 is clearly not designed for this. These zeros columns are automatically removed in our MA57 experiments.

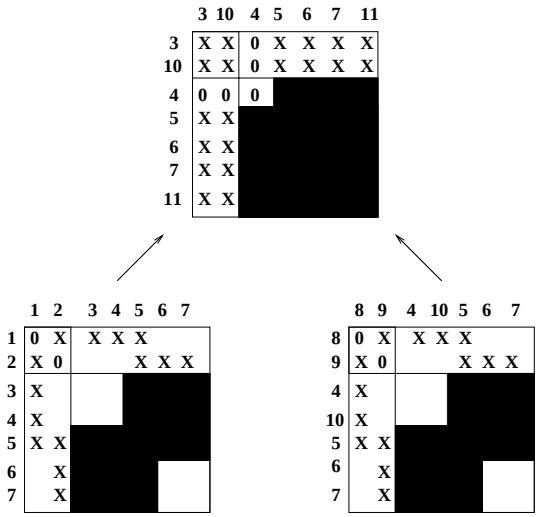


Figure 2.2.2: An example of removable zeros in the MA57 factors when we are lucky. **X** corresponds to a nonzero and shaded areas to nonzeros in the contribution blocks.

2.3 Ordering and 2 by 2 pivot selection

In this section, we present ordering approaches which have the same flavour as our MA57 preprocessing. We will first briefly introduce the notation that will be used. Then we will describe three kinds of (relaxed) constrained orderings in the framework of a solver which can manage the zeros blocks coming from *oxo* and *tile* pivots.

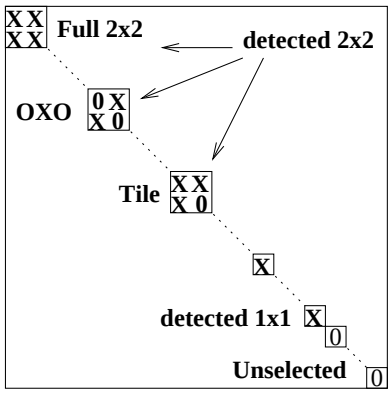


Figure 2.3.1: Detection of 1×1 and 2×2 pivots.

We use the algorithm of Section 1.5.2.1 to compute a symmetric weighted matching $\mathcal{M}_s$. The associated symmetric permutation can be rearranged so that our matrix A has the form of Figure 2.3.1 (we group *full*, *oxo*, *tile* and 1×1 pivots for having a convenient representation and this matrix will be later reordered in order to decrease the fill-in in the factors). We recall that we defined the set $\mathcal{M}_{2\times2}$ of detected 2×2 pivots, the set $\mathcal{M}_{1\times1}$ of detected 1×1 pivots, and that we have $\mathcal{M}_s = \mathcal{M}_{1\times1} \cup \mathcal{M}_{2\times2}$ and that the set $\mathcal{M}_s^c$ is the set of unselected 1×1 pivots (it is the complementary set to $\mathcal{M}_s$).

2.3.1 Acceptable and unselectable pivots

In our orderings, we use the set $\mathcal{P}_s$, the candidate pivot set. At each step k of the symbolic factorization the set $\mathcal{P}_s$ of candidate pivots is updated and divided into two parts $\mathcal{A}_s^k$ and $\mathcal{U}_s^k$. $\mathcal{A}_s^k$ is the set of **selectable** (or **acceptable**) pivots and $\mathcal{U}_s^k$ is the set of **unselectable** (or **unacceptable**) pivots. At each step of Gaussian elimination, the ordering tries to select the pivot of minimal metric among the set of acceptable pivots.

$$\begin{pmatrix} 0 & m & \times \\ \times & 0 & m \\ m & \times & 0 \end{pmatrix}$$

Figure 2.3.2: Matching and potential pivots on 3×3 example.

We illustrate these sets by the small 3×3 example in Figure 2.3.2, where the matching is shown as m and the candidate 2×2 pivot is defined by the off-diagonal entry (1,2). Thus, the set $\mathcal{M}_{2\times2}$ is $\{(r_1,c_2),(r_2,c_1)\}$, the set $\mathcal{M}_{1\times1}$ is empty, and the set $\mathcal{M}_s^c$ is $\{(r_3,c_3)\}$. For example, $\mathcal{P}_s=\{(r_1,c_2),(r_2,c1),(r_3,c_3)\}$ and we can set $\mathcal{A}_s^1=\mathcal{M}_s$ and $\mathcal{U}_s^1=\mathcal{M}_s^c$. At the first step of the ordering, 2×2 pivot corresponding to $\{(r_1,c_2),(r_2,c_1)\}$ is selected and removed from $\mathcal{P}_s$. We then build $\mathcal{A}_s^2=\{(r_3,c_3)\}$ and $\mathcal{U}_s^2=\emptyset$. Note that the entry in position $(3,3)$ has been filled by the previous elimination.

2.3.2 Ordering with fixed 2 by 2 pivots

Algorithm 2.3.1 Main steps of an ordering with fixed 2×2 pivots

$\mathcal{A}_s^1\leftarrow\mathcal{M}_s$, $\mathcal{U}_s^1\leftarrow\mathcal{M}_s^c$ and $k\leftarrow1$.
while $\mathcal{A}_s^k\neq\emptyset$ **do**
  $p\leftarrow arg\min_{i\in\mathcal{A}_s^k} metric(i)$
  Do symbolic elimination of p.
  Set $\mathcal{F}$, the indices of filled diagonal entries in $\mathcal{U}_s^k$.
  $\mathcal{A}_s^{k+1}\leftarrow(\mathcal{A}_s^k\setminus\{p\})\cup\mathcal{F}$ and $\mathcal{U}_s^{k+1}\leftarrow\mathcal{U}_s^k\setminus\mathcal{F}$
  $k\leftarrow k+1$
end while
if $\mathcal{U}_s^k\neq\emptyset$ **then** /* the matrix is structurally singular */
  Put $\mathcal{U}_s^k$ variables at the end of the permutation.
end if

In this approach we want the ordering to follow the pivots chosen by our symmetric `MC64` preprocessing. Algorithm 2.3.1 shows the main steps of this modified `MA47` analysis. We take $\mathcal{P}_s=\mathcal{M}_s^c\cup\mathcal{M}_s$, the candidate pivot set. At each step of the ordering, a pivot can be selected during the analysis if and only if it is a 2×2 detected pivot or a 1×1 pivot in $\mathcal{P}_s$ that corresponds to a nonzero diagonal entry of the reduced submatrix. For example, entry (3,3) of Figure 2.3.2 cannot be selected at the first step, but becomes selectable because of the fill-in coming from the pivot corresponding to $\{(r_1,c_2)/(r_2,c_1)\}$. At each step k of the ordering, the selected pivot is removed from the set of acceptable pivots and the new nonzero diagonal entries may be added. After the elimination of all the 2×2 pivots which belong to an odd cycle of Section 1.5.2.1, we are sure (assuming that there is no numerical cancellation) that the unselected 1×1 entry has been filled. After having

eliminated all acceptable pivots, if the set of unselectable pivots is non empty then the matrix is structurally singular. In the structurally singular case, the remaining variables are put at the end of the permutation. We will now omit the k in our notation when there will be no ambiguity.

This approach is the analogue to an ordering on the compressed graph in the MA57 context even if the compression is not done explicitly ($\mathcal{A}_s$ corresponds to free variables and $\mathcal{U}_s$ to constrained variables). Avoiding the compression of the graph enables MA47 to know the type of the 2×2 pivots (*oxo*, *tile* or *full*). The metric used is a variant of the Markowitz cost [47]. For a 1×1 pivot i,

$$metric(i) = (|R_i| - 1)^2,$$

for an *oxo* pivot

$$metric(O_{ij}) = (|R_i| - 1)(|R_j| - 1),$$

and for a *tile* pivot

$$metric(T_{ij}) = (|R_j| - 1)(|R_i| + |R_j| - 3).$$

Concerning the *full* 2×2 pivots, the real cost should be related to the quantity $(|R_i \cup R_j| - 2)^2$. Unfortunately this quantity is expensive to compute for all the candidate *full* 2×2 pivots in the framework of MA47. We tested different metrics and choose

$$metric(F_{ij}) = (|R_i| + |R_j| - 2)^2/2$$

to have a cheap estimate of the cost of the elimination of a *full* 2×2 pivot.

2.3.3 Ordering with pattern of selectable pivots

Algorithm 2.3.2 Main steps of an ordering with pattern of selectable pivots

Let $\mathcal{S} = \{(i,j)$ such that $|\tilde{a}_{ij}| >$ threshold $\}$.
$\mathcal{A}_s \leftarrow \mathcal{S} \cup \mathcal{M}_{1\times1}$.
while $\mathcal{A}_s \neq \emptyset$ **do**
  $p \leftarrow arg\min_{i\in\mathcal{A}_s} metric(i)$
  Do symbolic elimination of p.
  Update $\mathcal{A}_s$ and $\mathcal{M}_s$ with algorithm 2.3.3 to keep a maximum matching property.
end while
Put $\mathcal{M}_s^c$ variables at the end of the permutation.

The previous selection can be relaxed by enlarging the set of acceptable pivots but working on A restricted to $\mathcal{M}_s$. We will denote this matrix by $\tilde{A}$. Let $\mathcal{S} = \{(i,j)$ such that $|\tilde{a}_{ij}| >$ threshold $\}$. The set of acceptable pivots is initialized to $\mathcal{S} \cup \mathcal{M}_s = \mathcal{S} \cup \mathcal{M}_{1\times1}$ (we are sure that $\mathcal{M}_{2\times2}$ entries are in $\mathcal{S}$ whereas some entries in $\mathcal{M}_{1\times1}$ may not be in $\mathcal{S}$). Algorithm 2.3.2 shows the main steps of this ordering. To keep as much information as possible about potentially good pivots and to guarantee the good completion of the ordering, we decided to add entries in $\mathcal{A}_s$. For example, if a 2×2 pivot $(i,j) \notin \mathcal{M}_s$, (not detected by the MC64SYM preprocessing) is selected, then the pattern defined by $\mathcal{A}_s \setminus \{2\times2$ pivots which intersect $(i,j)\}$ may be structurally

singular and we will not be able to order some pivots. We force the ordering to keep a symmetric maximum matching on the remaining submatrix (see Algorithm 2.3.3). We have the following cases during the update of the set of numerically acceptable pivots ($\times$ corresponds to the current pivots, m corresponds to the old matching entries, f corresponds to new matching entries):

(case 1) $\begin{array}{c} \\ p \\ i \end{array} \begin{array}{c} \begin{array}{cc} p & i \end{array} \\ \begin{pmatrix} \times & m \\ m & f \end{pmatrix} \end{array}$, (case 2) $\begin{array}{c} \\ i \\ j \\ j_1 \end{array} \begin{array}{c} \begin{array}{ccc} i & j & j_1 \end{array} \\ \begin{pmatrix} m & \times & \\ \times & & m \\ & m & f \end{pmatrix} \end{array}$,

(case 3) $\begin{array}{c} \\ j \\ i \\ i_1 \end{array} \begin{array}{c} \begin{array}{ccc} j & i & i_1 \end{array} \\ \begin{pmatrix} m & \times & \\ \times & & m \\ & m & f \end{pmatrix} \end{array}$, (case 4) $\begin{array}{c} \\ i \\ j \\ i_1 \\ j_1 \end{array} \begin{array}{c} \begin{array}{cccc} i & j & i_1 & j_1 \end{array} \\ \begin{pmatrix} & \times & m & \\ \times & & & m \\ m & & & f \\ & m & f & \end{pmatrix} \end{array}$.

Algorithm 2.3.3 Update of the set of numerically acceptable pivots.

Remove from $\mathcal{A}_s$ the entries which intersect the current pivot.
if the selected pivot is in $\mathcal{M}_s$ **then**
  Remove the current pivot from $\mathcal{M}_s$.
else
  if the selected pivot is a 1×1 pivot p **then**
    Let i be such that $(i,p) \in \mathcal{M}_s$.
    Add (i,i) to $\mathcal{M}_s$ and $\mathcal{A}_s$, remove (i,p) from $\mathcal{M}_s$ (case 1).
  else
    Let (i,j) , $i \neq j$ be the 2×2 pivot, i_1 be such that $(i,i_1) \in \mathcal{M}_s$ and j_1 be such that $(j,j_1) \in \mathcal{M}_s$.
    if $i_1 = i$ and $j_1 \neq j$ **then**
      Add (j_1,j_1) to $\mathcal{M}_s$ and $\mathcal{A}_s$ (case 2).
    else if $j_1 = j$ and $i_1 \neq i$ **then**
      Add (i_1,i_1) to $\mathcal{M}_s$ and $\mathcal{A}_s$ (case 3).
    else if $i \neq i_1$ and $j \neq j_1$ **then**
      Add (i_1,j_1) to $\mathcal{M}_s$ and $\mathcal{A}_s$ (case 4).
    end if
    Remove (i,i_1) and (j,j_1) from $\mathcal{M}_s$.
  end if
end if

2.3.4 Use of MA57 orderings

An *a priori* drawback of the above approaches is that they are based on a Markowitz cost metric whereas our MA57 study has shown that METIS or AMF improves the quality of the analysis significantly. This motivates us to try to exploit an MA57 permutation in MA47. To do that we need to say to the MA47 analysis where the 2×2 pivots are in order to exploit as much as possible the *tile* and *oxo* pivots. That is why we try to feed MA47

with the permutation coming from a compressed graph approach (see Section 1.5.3.1) and using AMD, AMF or METIS.

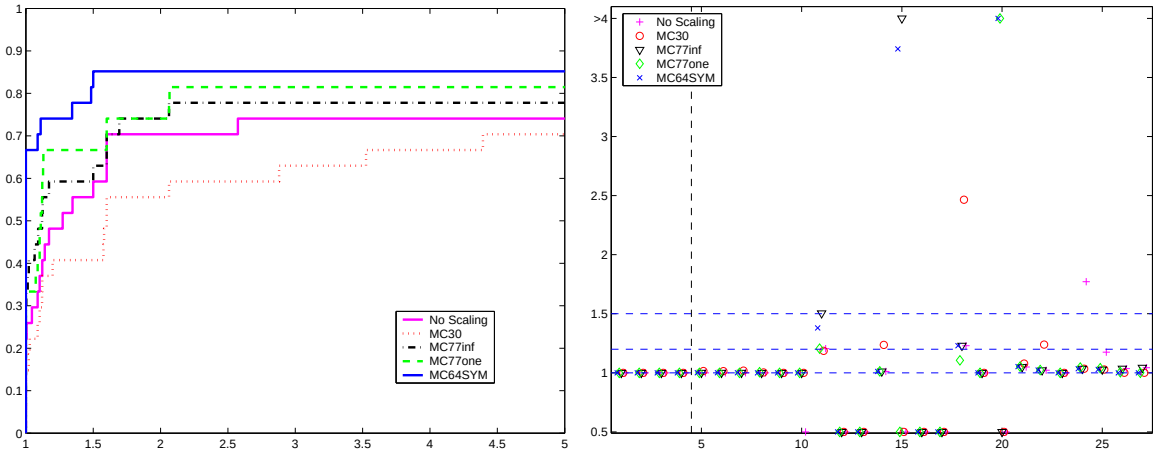
2.4 MA47 experimental results

From the above presentation of implemented strategies in MA47 we have the following approaches:

- MA47_1 refers to the default MA47. This strategy can be coupled with different scalings (MC30, MC77inf, MC77one MC64SYM ...). When it is not explicitly mentioned, the MC64SYM scaling is used.
- MA47_2 refers to the approach with fixed pivots ordering of Section 2.3.2.
- MA47_3 refers to the approach with a pattern of acceptable pivots of Section 2.3.3. A threshold of 0.99 is used to define $\mathcal{S}$.
- MA47_4 refers to the approach using the ordering on the compressed graph coming from MA57 of Section 1.5.3.1. This strategy can be combined with different orderings (AMD, AMF or METIS).

A relative threshold of 10^{-2} is used in both MA47 and MA57 to perform numerical pivoting. We conduct our experiments on the sets of matrices presented in Section 1.3.2.

2.4.1 Scaling influence



(2.4.1.a) CPU factorization time profile (set 2). (2.4.1.b) Memory prediction of the analysis (set 2).

Figure 2.4.1: Influence of the scaling.

As with MA57, we observed that the MC30 scaling degrades the MA47 factorization. For MA47 the MC64SYM scaling is slightly better than the MC77 scaling (see Figure 2.4.1). That is why it is retained as the default scaling in all of our approaches.

2.4.2 Ordering influence

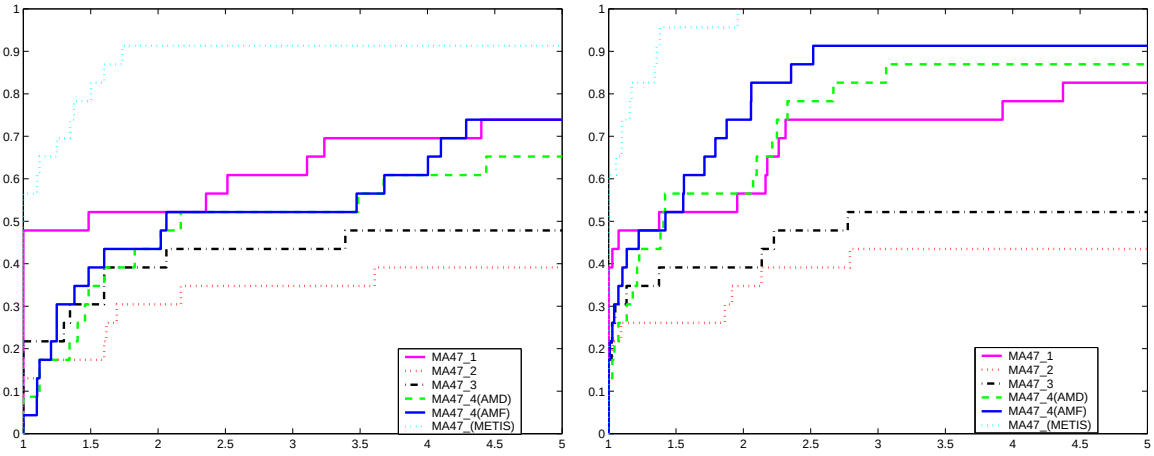
2.4.2.1 Number of failures

No error is reported for the set 1 (see Section 1.3.2) in Table 2.4.1, which means that MA47 never fails on the set 1. MA47_1 without scaling is the least robust approach, it fails for 7 matrices. MA47_4 with METIS is clearly the most robust approach on set 2. It seems that the approaches using a variant of the Markowitz cost (MA47_2 and MA47_3) suffer from the approximations in the computation of the Markowitz cost of the full 2×2 pivots and perhaps of the *oxo* and *tile* pivots metric also.

Matrix	MA47_1		MA47_2	MA47_3	MA47_4	MA47_4	MA47_4
	No Scaling	MC64SYM			AMD	AMF	METIS
A0NSDSIL	0	0	C	0	0	0	0
A2NNSNSL	0	0	C	0	0	0	0
BLOWEYA	C	0	0	0	0	0	0
BRATU3D	C	C	0	0	0	0	0
CONT-201	C	C	0	0	0	0	0
NCVXQP1	C	0	0	0	0	0	0
NCVXQP5	A	A	F	C	C	C	0
NCVXQP7	C	C	C	C	C	C	0
cvxqp3	C	0	0	0	A	0	0
olesnik0	0	0	C	0	0	0	0
stokes128	0	0	C	C	0	0	0

Table 2.4.1: MA47 error output. 0: success. C: CPU time exceeded. A: not enough memory after analysis. F: not enough memory during factorization. Only matrices on which at least one version of MA47 did not succeed are shown.

2.4.2.2 Factorization time and memory requirement



(2.4.2.a) CPU factorization time profile (set 2). (2.4.2.b) Memory used profile (set 2)

Figure 2.4.2: MA47: factorization time and memory on set 2.

On set 2, MA47_4 with the METIS ordering on the compressed graph seems to be the best of the MA47 strategies in terms of memory used and CPU factorization time

(see Figures 2.4.2.a and 2.4.2.b). It seems that the approaches based on the minimum degree/fill metric generates orderings which are poorer at preserving the *oxo* and *tile* structures. Again we remark that the MA47_2/3 orderings are too coarse and thus do more operations and have more entries in the factors.

2.4.2.3 Analysis estimations and memory used

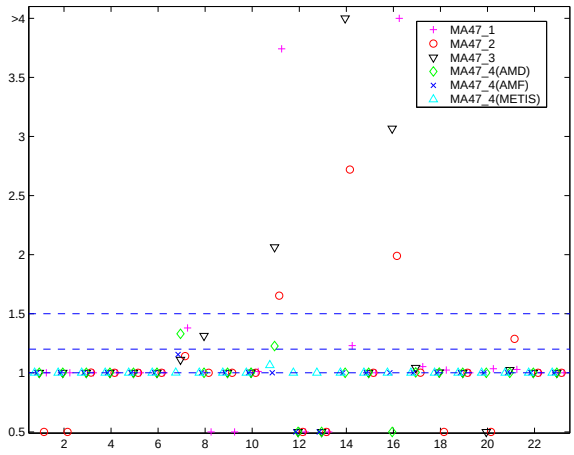


Figure 2.4.3: MA47: reliability of analysis memory prediction (set 2). Ratio between used memory and predicted memory. A ratio of 0.5 indicates a failure.

Figure 2.4.3 shows the ratio between the memory used and the memory estimated by the analysis. We see that using only the MC64SYM scaling can be very dangerous. Clearly the best memory estimation is given by MA47_4+METIS: the increase in memory over the analysis forecast during factorization is always less than 20% and it never fails.

That is why we retain only the MA47_4+METIS approach with a 20% memory relaxation for the factorization. We think that the approaches MA47_2/_3 can be potentially improved but with significantly more work. This point will be discussed in our conclusion.

2.5 MA47/MA57 comparison

2.5.1 Augmented systems with 2 zero blocks (set 1)

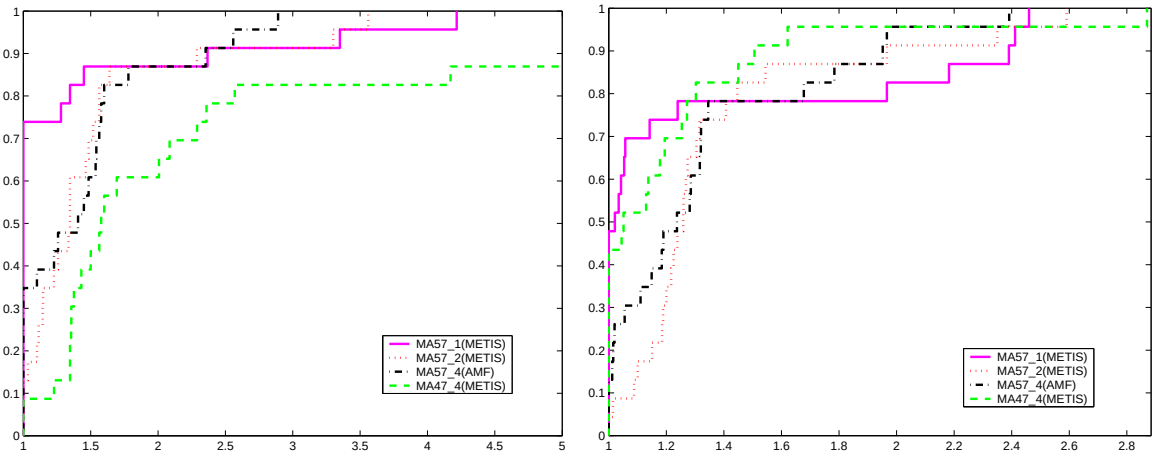
On set 1, the analysis phase of MA47 will identify a structurally nonsingular block and will permute a block of zeros to the end. We call this the structural kernel. On these systems, MA47 approaches (except the approach with METIS) have the least factorization time. Moreover we observed that preprocessings does not have a much influence on MA47 factorization time. Table 2.5.1 shows that MA47 is far better than MA57 on the set 1. Moreover, because of the structural singularity there is much numerical cancellation that MA57 cannot predict and it uses more memory. We decided to keep MA47_2 because it seems to us that it could be the more robust approach on even more tricky problems even

if there are not much differences between MA47_2 and MA47_1 in Tables 2.5.1. In the rest of this section we will focus on the set 2.

Matrix	MA57_1	MA57_4	MA47_2	MA47_1
AUG2D	316.	58.7	0.02	0.02
AUG2DC	322.	1540	0.03	0.02
AUG3D	–	–	0.01	0.02
DTOC	11.3	7.36	0.02	0.02

Table 2.5.1: MA47/MA57 CPU factorization time on set 1. Time in seconds. METIS used for MA57.

2.5.2 Memory, size of the factors, number of operations and factorization time



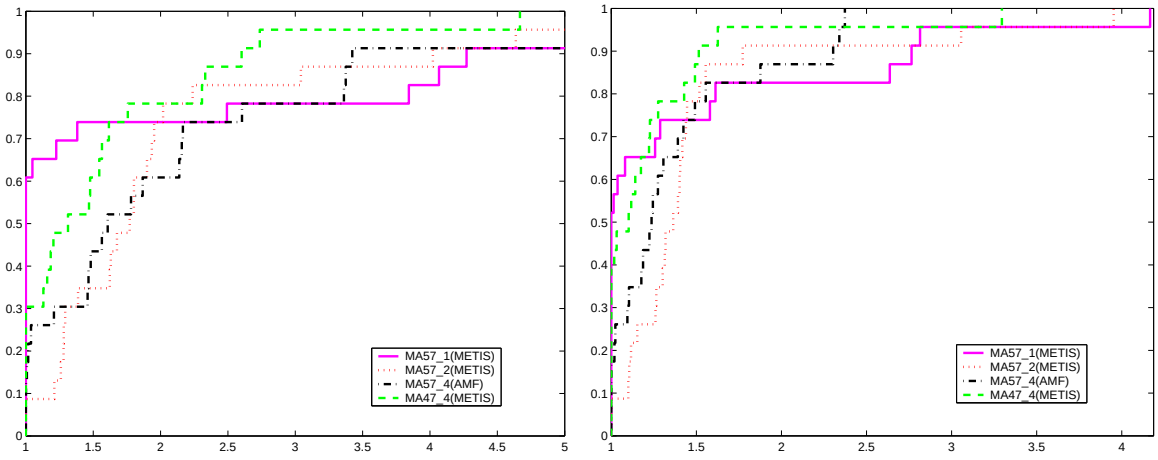
(2.5.1.a) CPU factorization time profile (set 2). (2.5.1.b) Memory profile (set 2).

Figure 2.5.1: MA47/MA57 comparison on set 2.

On set 2, MA57 approaches are faster than MA47 approaches (see Figure 2.5.1.a), but MA47 tends to use less or the same memory as MA57 (see Figure 2.5.1.b). This tendency is also true when looking at the number of operations (see Figures 2.5.2.b). Figure 2.5.2.a shows that MA47 tends to have sparser factors than MA57 approaches. Hence it will do less operations during the solve phase. Sometimes MA47 takes advantage of the zero blocks and avoids much of the computation and storage done by MA57. When numerical pivoting does not change the analysis predictions, MA47 can manage the zero blocks of *tile* and *oxo* pivots better than MA57. The drawback of the MA47 approach is that the structures are more complicated with smaller granularity when calling level 3 BLAS dense kernels. Hence, for an equal number of floating-point operations, MA57 will be faster than MA47.

2.6 Conclusions

We have observed the same scaling effect with MA47 as with MA57. MA47 can benefit from preprocessings designed for MA57. We have shown that MA47 is far better than



(2.5.2.a) Total number of operations profile (set 2).

(2.5.2.b) Number of nonzeros in the factor profile (set 2).

Figure 2.5.2: MA47/MA57 comparison on set 2.

MA57 on set 1. MA47, however, is not competitive with MA57 on set 2, but we have stressed some of its good points (number of operations, memory used and size of the factors). We hope to improve its factorization time on set 2 by finding answers or partial answers to the following open questions.

In the next chapter, we will try to find a robust way to perturb 2×2 pivots. In particular, we may want to have perturbations which respect the structure of the matrix and thus preserve what the MA47 analysis has predicted.

An answer to the following question might provide a significant improvement for MA47. How do we improve the accuracy of MA47 metric or have an AMF like metric which takes into account the zero blocks? A drawback of the original MA47 analysis was that it was not able to restrict $a\ priori$ the search set of good structural pivots (it has to look at some combinations of rows/columns to form 1×1 or 2×2 pivots). Mechanisms with candidate pivots significantly decrease the cost of searching for pivots. It would also enable us to compute a more complicated and better approximate metric and thus have a kind of AMF metric without increasing the complexity too much.

Chapter 3

Static pivoting for sparse symmetric indefi nite problems

3.1 Introduction

We are interested in solving $Ax = b$ where A is symmetric indefinite in the framework of a static multifrontal solver. By a static code we mean a code in which the factorization respects the ordering of the analysis. The factorization does not necessarily follow the analysis and some slight variations are allowed. For example in a multifrontal context, it is sufficient that the factorization decisions are compatible with the assembly tree (numerical pivoting can be performed within a front). Hence the analysis predicts exactly the memory needed and the number of operations of the factorization because it selects pivots from the fully summed variables for each node of the elimination tree and it does not postpone pivots. This approach was first proposed by [77] in the context of LU factorization. During Gaussian elimination *"small perturbations"* are added to limit the growth of the factors in order to enhance the backward stability of the algorithm (see for example [71] Theorem 11.4). By *"small perturbations"* we mean that the factorization of the perturbed matrix has to be as close as possible to the factorization of the original matrix so that it is possible to have a cheap and sufficiently accurate solution using the computed factors. In the framework of our study, we want a static approach with the following features:

(F1) The decision of adding a perturbation or not is easy to take. In particular, in a parallel symmetric indefinite solver, this decision must not involve any extra communication cost.

(F2) The perturbations are restricted to the block of fully summed rows/columns in each front.

(F3) The factorization of the perturbed matrix must lead to a stable solution in most cases.

Given a symmetric matrix A, not necessarily positive definite, approaches to compute a modified Cholesky factorization of $A + E$ with E as small as possible have been developed in [22, 51, 59], but it seems difficult to adapt them to our case; firstly because we want to have E small even if A has a very negative eigenvalue (in a modified

Cholesky $||E||_2 \geq -\min \lambda_i(A)$); secondly because the perturbations have to be localized which is not the case in [22].

In Section 3.2, we formalize our approach and give theoretical solutions. In Section 3.3, we examine experimental results on real test problems using two multifrontal solvers `MA47` [48] and `MA57` [37].

3.2 Static pivoting and perturbations

3.2.1 Notation

In the following, s will be the function to denote the sign of a real number:

$$s : x \rightarrow \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases} .$$

$||\ ||_1$, $||\ ||_\infty$ will denote sub-multiplicative matrix norms and $||A||_M$ will denote the norm $\max_{ij} |a_{ij}|$. For each matrix or submatrix $A = (a_{ij})$, $|A| = (|a_{ij}|)$. u and $\mu \in]0, 1]$ will denote real numbers which will be used as thresholds in our pivoting strategies. Let A be the matrix that we want to factorize, we define $\tau = u||A||_1$.

3.2.2 Numerical pivoting

Here we briefly recall which criteria are commonly used in the context of numerical pivoting to ensure stability. In the context of sparse LU factorization with numerical pivoting, a 1×1 pivot is stable if and only if

$$|a_{ii}| \geq u \max_k |a_{ik}| \tag{3.2.1}$$

where u is a threshold between 0 and 1. In the context of sparse symmetric matrices, the criterion of [44] uses a threshold $u \in [0, \frac{1}{2})$ and can be used for the 2×2 pivots to ensure a growth factor lower than $1/u$:

$$P = \begin{pmatrix} a_{ii} & a_{ij} \\ a_{ij} & a_{jj} \end{pmatrix} \text{ is stable } \Leftrightarrow |P^{-1}| \begin{pmatrix} \max_{k\neq i,j} |a_{ik}| \\ \max_{k\neq i,j} |a_{jk}| \end{pmatrix} \leq \begin{pmatrix} 1/u \\ 1/u \end{pmatrix}. \tag{3.2.2}$$

3.2.3 Static pivoting: problem formulation

In the context of LU factorization with static pivoting, `SuperLU_DIST` [79] adds small perturbations δ to the diagonal entries when the pivot a_{ii} is too small and the inequality (3.2.1) is transformed into the constraint

$$(\mathcal{C}_{1\times1}) \ : \ |a_{ii} + \delta| \geq u\ ||A||_1$$

At each step of Gaussian elimination, we have to solve the problem

$$(\mathcal{P}_{1\times1}) \begin{cases} \min |\delta| \\ w.r.t.\ (\mathcal{C}_{1\times1}) \end{cases}$$

PROPERTY 2.1. *A solution of* $(\mathcal{P}_{1\times1})$ *is given by:*

$$\delta = s(a_{ii}) \max(\tau - |a_{ii}|, 0)$$

where s *is the sign function and* $\tau = u||A||_1$.

In the context of 2×2 pivoting, we want to compute the factorization of $A+B = LDL^T$ where B is the block diagonal matrix which contains the 1×1 and 2×2 perturbations. If after computing the solution $x = (A+B)^{-1}b$, the backward error is too large, iterative refinement is applied. Intuitively, we hope that the smaller $||B||$ is, the fewer iterative refinement steps will be needed. This intuition is confirmed by theoretical bounds given in [71] (pages 231–243). By analogy to the 1×1 case, the inequality (3.2.2) is transformed into the constraint

$$(\mathcal{C}1) \ : \ ||(P+\Delta)^{-1}||_\infty \leq \frac{1}{u\ ||A||_1}.$$

Moreover we want to keep the numerical symmetry and the structure of the problem. This involves the following two constraints:

$$(\mathcal{C}2) \ : \ \Delta \text{ symmetric,}$$

$$(\mathcal{C}3) \ : \ \mathcal{P}attern(\Delta) \subset \mathcal{P}attern(P).$$

We will minimize the norm of a 2×2 or 1×1 perturbation matrix at each step of Gaussian elimination. Thus the choice of the norm which has to be minimized is not very important (the norm equivalence constants are small). To be homogeneous with constraint $(\mathcal{C}1)$ we decide to use an infinity norm. Finally, we have to solve the following optimization problem:

$$(\mathcal{P}_{||\ ||_\infty}) \ \begin{cases} \min ||\Delta||_\infty \\ w.r.t.\ (\mathcal{C}1),\ (\mathcal{C}2),\ (\mathcal{C}3). \end{cases}$$

3.2.4 *full* 2×2 pivot

In this section we present a way of adding small perturbations to a *full* 2×2 pivot to satisfy constraints $(\mathcal{C}1)$, $(\mathcal{C}2)$ and $(\mathcal{C}3)$. We propose a different point of view to the previous 1×1 perturbations, which makes the adaptation of the perturbation to the case of a 2×2 pivot easier.

A 1×1 pivot can be seen as a 1×1 matrix whose eigenvalue is $\lambda_1 = a_{ii}$. If $|\lambda_1|$ is too small then the previous 1×1 perturbation shifts it to $s(\lambda_1)\tau$.

By analogy, we will shift the eigenvalues of a *full* 2×2 pivot which are too small. Let λ_1 and λ_2 be the eigenvalues of the pivot P such that $|\lambda_1| \leq |\lambda_2|$, u_1 and u_2 be two associated normalized eigenvectors. $P = \lambda_1 u_1 u_1^T + \lambda_2 u_2 u_2^T$ and it is easy to move each of the eigenvalues independently using the rank one linear operators $u_1 u_1^T$ and $u_2 u_2^T$.

PROPERTY 2.2.

$$\Delta = s(\lambda_1)\ \max(\tau - |\lambda_1|, 0)\ u_1 u_1^T + s(\lambda_2)\ \max(\tau - |\lambda_2|, 0)\ u_2 u_2^T$$

is within a factor of $\sqrt{2}$ *of the solution of* $(\mathcal{P}_{||\ ||_\infty})$.

PROOF. We will first see that Δ is the solution to the following problem:

$$(\mathcal{P}_{||\ ||_2}) \left\{ \begin{array}{l} \min ||\Delta||_2 \\ w.r.t.\ (\mathcal{C}1),\ (\mathcal{C}2) \end{array} \right.$$

Indeed, for all Δ' which satisfy $(\mathcal{C}1)$ we have:

$$\begin{aligned} \min_x |x^T(P+\Delta')x| &= 1/||(P+\Delta')^{-1}||_2 \\ &\geq 1/||(P+\Delta')^{-1}||_\infty \\ &\geq \tau. \end{aligned}$$

For all x, we have:

$$\begin{aligned} |x^T\Delta' x| &\geq |x^T(P+\Delta')x| - |x^TPx| \\ &\geq \tau - |x^TPx| \end{aligned}$$

That is why $||\Delta'||_2 \geq |x^T\Delta' x| \geq \tau - |x^TPx|$. It implies that $||\Delta'||_2 \geq \tau - |\lambda_1| \geq ||\Delta||_2$. Thus Δ is a solution of $(\mathcal{P}_{||\ ||_2})$.

Let Δ_{opt} be a solution of $(\mathcal{P}_{||\ ||_\infty})$. We have

$$||\Delta||_\infty \leq \sqrt{2}||\Delta||_2 \leq \sqrt{2}||\Delta_{opt}||_2 \leq \sqrt{2}||\Delta_{opt}||_\infty$$

□

3.2.5 *oxo* pivot

Let $\Delta = \begin{pmatrix} 0 & \delta \\ \delta & 0 \end{pmatrix}$ be the perturbation. We have to solve:

$$(\mathcal{P}_{oxo}) \left\{ \begin{array}{l} \min |\delta| \\ w.r.t.\ |p_{12}+\delta| \geq \tau. \end{array} \right.$$

PROPERTY 2.3. *A solution is given by:*

$$\Delta = s(p_{12}) \max(\tau - |p_{12}|, 0) \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

The solution for an *oxo* pivot can be interpreted as a generalization of the 1×1 pivot solution. Indeed, applying an *oxo* pivot is numerically equivalent to applying two off-diagonal 1×1 pivots. Note that it corresponds also to a solution if the *oxo* pivot is considered as full (it is the case in most of the solvers because they cannot detect the zeros on the diagonal). Indeed we have $\lambda_1 = -\lambda_2 = p_{12}$, $u_1^T = (1,1)/\sqrt{2}$ and $u_2^T = (-1,1)/\sqrt{2}$.

3.2.6 *tile* pivot

For a tile pivot, the computation of an optimal perturbation is more complicated. We decided to simplify it and to add only off-diagonal perturbations for the following reason. The kinds of ordering that we want to apply in the context of static pivoting try to preselect potentially good 2×2 pivots during the analysis and thus have large off-diagonal entries (see Section 1.5.2.1). As we are going to measure the component-wise backward error it may preferable to perturb large original entries and thus off-diagonal entries.

Let $P = \begin{pmatrix} a & b \\ b & 0 \end{pmatrix}$ be a *tile* pivot and let $\Delta = \begin{pmatrix} 0 & \delta \\ \delta & 0 \end{pmatrix}$ be the perturbation.

After arranging problem $(\mathcal{P}_{||\,||_\infty})$, we have to solve:

$$(\mathcal{P}_{tile}) \begin{cases} \min |\delta| \\ w.r.t.\ |b+\delta| + |a| \leq (1/\tau)(b+\delta)^2 \end{cases}$$

PROPERTY 2.4. *A solution is given by:*

$$\Delta = \tau s(b) \max(\frac{1+\sqrt{1+4|a|/\tau}}{2} - \frac{|b|}{\tau}, 0) \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

PROOF. If $|b| + |a| \leq b^2/\tau$, obviously $\delta = 0$ is a solution of $(\mathcal{P}_{tile})$. Moreover it implies that $|b| \geq \tau(1+\sqrt{1+4|a|/\tau})/2$. If $|b| + |a| > b^2/\tau$, solving $(\mathcal{P}_{tile})$ leads to the equation $(b+\delta)^2 - \tau|b+\delta| - \tau|a| = 0$ which has a positive solution

$$|b+\delta| = \tau\frac{1+\sqrt{1+4|a|/\tau}}{2}.$$

□

Finding a perturbation that respects the pattern may not be well adapted to static pivoting. For example, it is possible to have $\delta = \mathcal{O}(\sqrt{\tau|a|})$. In other words, when a is large enough and b small enough we cannot succeed in finding a small enough perturbation which satisfies the *tile* constraints and which has the same order of magnitude as in the *full* case. We will see in the experiments that this drawback can be observed with MA47.

3.2.7 Mixing numerical pivoting and static pivoting

In this section, we present an approach which combines numerical checking for stability and 1×1 static pivoting. We decided to use only 1×1 perturbations because they are easier to implement and we want to clearly identify the impact of this mixed approach. In other words, in order to have a clear experimental analysis we will not try to have a mixed approach with 2×2 perturbations. We will return to this point in our conclusions.

Let us consider a frontal matrix from the elimination tree. It contains two kinds of variables, the fully summed variables (FSV) which correspond to the pivot block that we want to eliminate and the partially summed variables (PSV) on which the Schur complement will be computed.

Our mixed approach is based on two phases. In the first phase, we perform numerical pivoting in the block of fully summed variables until no remaining variables satisfy

the numerical criterion. In a second phase, we eliminate the remaining fully summed variables adding 1×1 perturbations if necessary.

Moreover, we can relax the conditions of stability of phase 1 in order to have less perturbation to the diagonal in phase 2. We will see also that this relaxation allows us to design a parallel approach. Instead of using the inequalities (3.2.1) and (3.2.2) for our stability criteria, we only consider entries in a subset K, $K \subset FSV \cup PSV$ and $FSV \subset K$. More precisely, during the first phase, a 1×1 pivot a_{ii} is considered to be stable if and only if

$$|a_{ii}| \geq u \max_{k\in K\setminus\{i\}} |a_{ik}| \tag{3.2.3}$$

and a 2×2 pivot P is considered to be stable if and only if

$$|P^{-1}| \begin{pmatrix} \max_{k\in K\setminus\{i,j\}} |a_{ik}| \\ \max_{k\in K\setminus\{i,j\}} |a_{jk}| \end{pmatrix} \leq \begin{pmatrix} 1/u \\ 1/u \end{pmatrix}. \tag{3.2.4}$$

Algorithm 3.2.1 Relaxed pivot selection in a frontal matrix

Phase 1: Eliminate as many 1×1 and 2×2 pivots as possible which satisfy inequalities (3.2.3) and (3.2.4) respectively using Duff-Reid algorithm with threshold u.
while there are uneliminated variables p in the fully summed block **do** /* Phase 2 */
  if $|a_{pp}| < \mu \max\{||A||_M, \max_{k\in K} |a_{pk}|\}$ **then**
    $a_{pp} \leftarrow s(a_{pp})\mu \max\{||A||_M, \max_{k\in K} |a_{pk}|\}$
  end if
  Perform eliminations using p as a 1×1 pivot.
end while

During the second phase we use a threshold μ. The static phase perturbs the diagonal of the matrix if the pivot is too small with respect to the initial values in A (smaller than $\mu||A||_M$) and the values of the current front which were allowed to be examined (smaller than $\mu \max_{k\in K} |a_{pk}|$). Algorithm 3.2.1 summarizes the two phases.

We decided to have two different versions of this algorithm: an approach with $K = FSV \cup PSV$ and an approach with $K = FSV$.

The first approach ($K = FSV \cup PSV$) will be referred to as the `mixedSEQ` algorithm because checking the stability within partially summed rows is not well designed for existing parallel implementations (it involves communications). Hence this approach does not fully agree with the feature (F1) presented in the introduction of this chapter.

The second approach ($K = FSV$) will be referred to as `mixedPAR` because it is more friendly for a parallel distributed implementation, as for example in `MUMPS` [7], where the fully summed part of a frontal matrix is stored on a single processor.

3.3 Experimental results

In this section, we consider the different combinations between the static approaches and the ordering. Firstly, we examine the influence of the thresholds for each approach. Secondly, we examine the different approaches in terms of precision of the solution, number of steps of iterative refinement and factorization time.

We saw in Section 1.6.2.4 for MA57 and in Section 2.4.2 for MA47 that METIS combined with the MC64SYM scaling was the best ordering on symmetric indefinite matrices in terms of CPU factorization time but that it sometimes caused many pivots to be delayed. On the contrary, the METIS approach based on the compressed graph combined with the MC64SYM scaling does more operations but predicts well what will happen during the factorization (see also Section 1.6.2.4). That is why we retain these two orderings for the present experiments.

Six static approaches emerge from our previous discussion. The first two correspond to mixed sequential approaches: the mixedSEQ approach can be combined with METIS (this approach will be referred to as MA57 mixedSEQ+METIS) and with the METIS approach based on the compressed graph (it will be referred to as MA57 mixedSEQ+COMPRESS).

The mixedPAR strategy can replace the mixedSEQ one in the above to give us MA57 mixedPAR+METIS and MA57 mixedPAR+COMPRESS.

The last two approaches correspond to static pivoting as in a SuperLU_DIST sense which strictly follow the pivot order predicted by the analysis. These approaches are only applied when the METIS ordering has been computed on the compressed graph (a pure static approach with only METIS would clearly not be safe). The approach referred to as MA57 static+COMPRESS corresponds to a static code based on MA57 which strictly follows the pivots decided by the analysis and which does 1×1 and *full* 2×2 perturbations. The use of *tile* and *oxo* perturbations can be exploited in a code which can recognize the real structure of the pivots. With MA57 some numerical cancellations may not be seen. That is why we decided to apply the *tile* and *oxo* perturbations only to the MA47 factorization. These kinds of perturbations have also the advantage of keeping the predicted MA47 structure during the factorization. This approach will be referred to as MA47 static+COMPRESS.

We compute the sparse component-wise backward error using the theory and measure developed by [14]. The scaled residual of the i^{th} equation is

$$\frac{|r_i|}{(|A||x| + |b|)_i} \text{ where } r = b - Ax \text{ and } x \text{ is the computed solution}$$

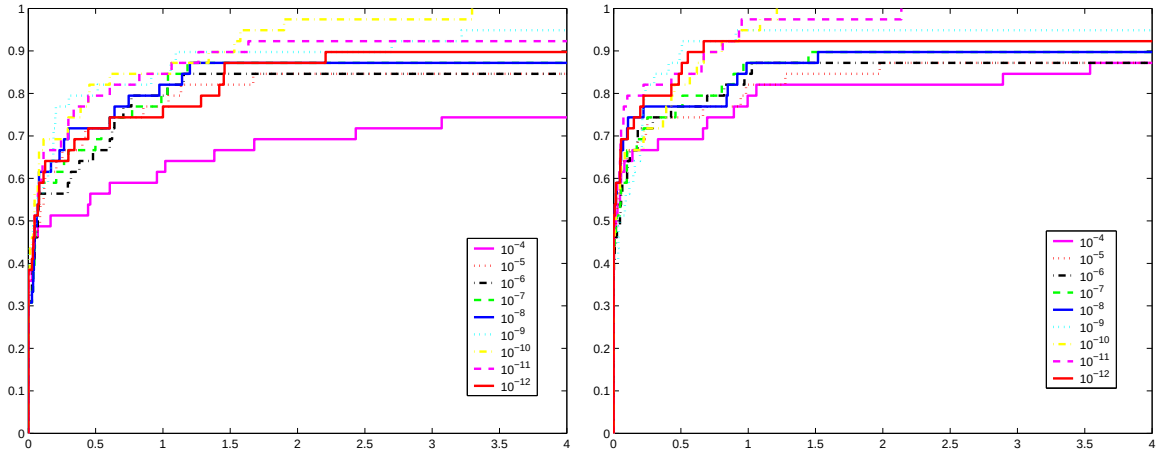
except if the denominator is too small. In this case, we measure

$$\frac{|r_i|}{((|A||x|)_i + ||A_i||_\infty ||x||_\infty)_i} \text{ where } A_i \text{ represents the } i^{th} \text{ row of } A.$$

We apply iterative refinement in all our approaches. At each step k of the iterative refinement, we compute the current backward error $berr^{(k)}$. We stop if $berr^{(k)} < 10^{-15}$ or if $berr^{(k)} > 0.9 \times berr^{(k-1)}$ (the convergence rate is too slow) or $k = 20$ (the maximum number of iterations has been reached).

3.3.1 Threshold influence on the precision

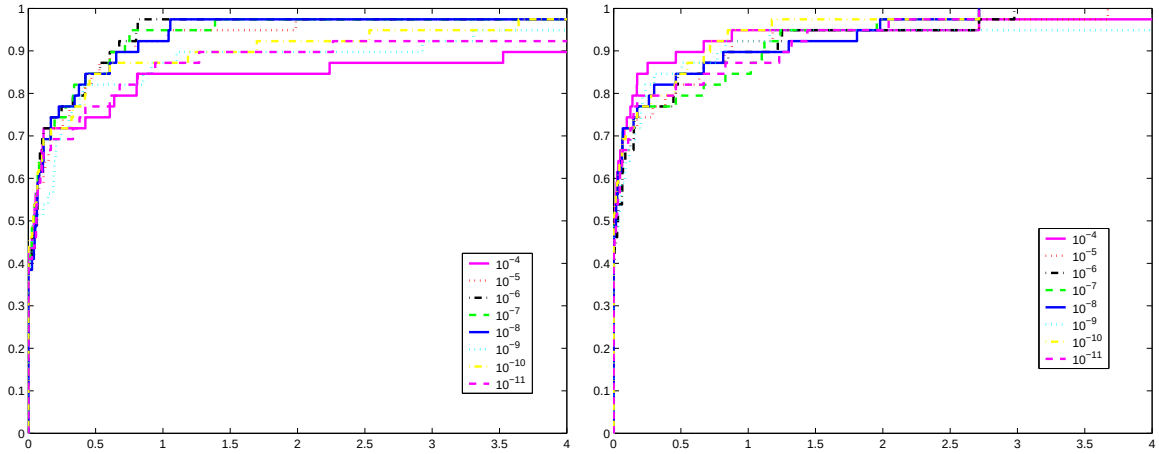
Concerning the mixedPAR approach, we decided to keep the threshold of numerical pivoting u at 10^{-2}. It is not numerically safe to make it smaller because the stability is



(3.3.1.a) MA57 mixedPAR+METIS. (3.3.1.b) MA57 mixedPAR+COMPRESS.

Figure 3.3.1: Component-wise backward error profi le with MA57 and mixedPAR pivoting while varying μ. $\log_{10}$ scale used for the precision.

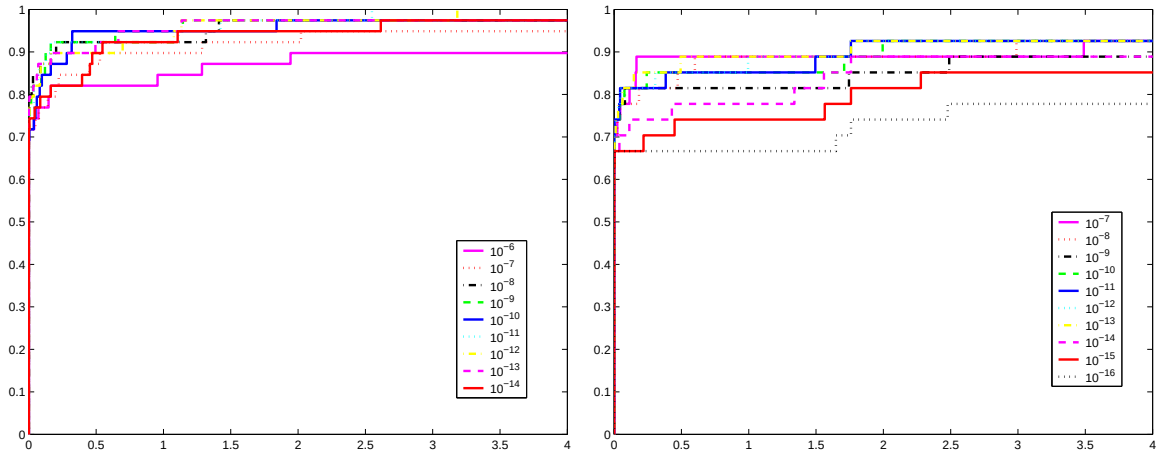
only checked in the fully summed block. Figure 3.3.1 shows the influence of the static threshold μ when the METIS ordering is used (left plot) and when the METIS ordering is applied on the compressed graph (right plot). For both orderings, values of μ between 10^{-9} and 10^{-11} seem to have comparable effects and to be the more adapted for the precision of the solution. We decide to keep $\mu = 10^{-10}$ for both approaches.



(3.3.2.a) MA57 mixedSEQ+METIS. (3.3.2.b) MA57 mixedSEQ+COMPRESS.

Figure 3.3.2: Component-wise backward error profi le with MA57 and mixedSEQ pivoting while varying μ. $\log_{10}$ scale used for the precision.

There is no reason to keep $u = 10^{-2}$ for the mixedSEQ approaches. Indeed there is no reason to refuse a 2×2 pivot because u is large and then in the second phase to accept 1×1 pivots which are less stable. Moreover we experimented to keep $u = 10^{-2}$ with different values of μ. We remarked that the quality of the solution was far from the results that we are going to present here. We decided to set u and μ to the same value for the mixedSEQ approaches. Figure 3.3.2 shows that the static threshold does not influence the precision of the solution too much. We keep $\mu = 10^{-8}$ for the mixedSEQ approaches on both orderings.



(3.3.3.a) MA57 static+COMPRESS. (3.3.3.b) MA47 static+COMPRESS.

Figure 3.3.3: Component-wise backward error with 1×1 and 2×2 perturbations. $\log_{10}$ scale used for the precision while varying τ.

We set $u = 10^{-2}$ for the static approaches. Figure 3.3.3.a shows that the static threshold does not have much influence on MA57 static+COMPRESS, and we decided to set μ to 10^{-10}. Concerning the approach MA47 static+COMPRESS we do not retain a threshold of 10^{-7} (which seems good in Figure 3.3.3.b) because it causes too many iterative refinement steps. As with MA57, we also keep $\mu = 10^{-10}$ for the MA47 static+COMPRESS approach.

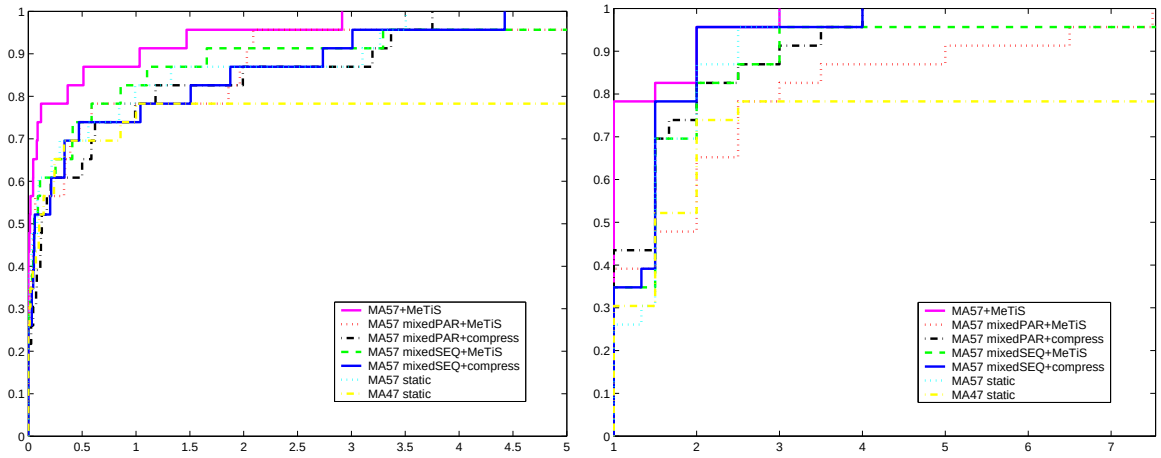
3.3.2 Comparison of the different approaches

MA57 and MA47 can only be compared on set 2 (on set 3 MA57 is clearly better and on set 1 classic MA57 approaches are slower or fail more than MA47). Moreover we remarked that most of the numerical difficulties are in the set 2 (there are few delayed pivots in set 3). That is why the profiles of Sections 3.3.2.1 and 3.3.2.2 are only done for set 2.

3.3.2.1 Precision of the solution and number of iterative refinement steps

Naturally the best approach in terms of precision of the solution is the approach using numerical pivoting. The precisions of the mixedPAR and the mixedSEQ approaches are comparable and do not seem to be very sensitive to the ordering (see Figure 3.3.4.a). Nevertheless, Figure 3.3.4.b shows that applying a METIS ordering to the compressed graph tends to reduce the number of iteration steps. MA57 static validates our *full* 2×2 perturbations presented in Section 3.2.4 and confirms what happens in the LU case: in general when good pivots are preselected during the analysis, it is not necessary to check stability and adding perturbations is sufficient. We notice also that the precision of MA47 static+COMPRESS is degraded by the *tile* perturbations and is far less robust than the MA57 static approaches (see Figure 3.3.4.a).

Table 3.3.1 indicates how many times the component-wise backward error is in the ranges $[0, 10^{-15}[$, $[10^{-15}, 10^{-8}[$, $[10^{-8}, 10^{-4}[$ and $[10^{-4}, +\infty[$. It gives us an idea of



(3.3.4.a) Profi le of component-wise backward error profi le. log_{10} scale used for the x-axis.

(3.3.4.b) Profi le of number of iterative refi nement steps.

Figure 3.3.4: Solution phase comparison on set 2.

Code	$berr \in$ $[0, 10^{-12}]$	$]10^{-12}, 10^{-8}]$	$]10^{-8}, 10^{-4}]$	$]10^{-4}, 1]$
MA57				
MeTiS	_/23/16	_/0/0	_/0/0	_/0/0
mixedSEQ+MeTiS	4/22/16	0/0/0	0/0/0	0/1/0
mixedSEQ+COMPRESS	_/21/16	_/2/0	_/0/0	_/0/0
mixedPAR+MeTiS	4/21/16	0/2/0	0/0/0	0/0/0
mixedPAR+COMPRESS	_/20/16	_/3/0	_/0/0	_/0/0
static+COMPRESS	_/21/15	_/2/1	_/0/0	_/0/0
MA47				
static+COMPRESS	4/18/_	0/0/_	0/1/_	0/4/_

Table 3.3.1: Range of component-wise backward error after iterative refi nement on all our test sets. Results given in the form set 1 / set 2 / set 3.

the absolute values of the backward error and confirms the behaviour of the codes. We can see that the static MA57 approaches are quite robust in terms of precision. We also ran the static approach that uses the METIS ordering on the set 1 (we did not do that with the ordering on the compressed graph because the factorization time increases dramatically with MA57 in this case as we remarked in Section 2.5.1) and noticed that MA57 was in this case competitive with MA47. Table 3.3.1 shows that the solution is accurate in this case.

In Table 3.3.2, we count the number of times where an approach calls the solution phase (#itstep + 1) to obtain a component-wise backward error in a fixed range. It gives us an idea of the cost of the solution phase when the objective is to get a backward error smaller than 10^{-15} and is a complement to what we observed in Figure 3.3.4.b. We make the following remarks :

- The global results are quite encouraging, in most of the approaches and on most cases, the number of iterative refinement steps is not too large.

- The approach with numerical pivoting does less than one iterative refinement step in most of the cases.

- The use of an ordering based on the compressed graph decreases the number of refinement steps. Then, the supplementary cost of the factorization can be compensated by a low solution cost. This effect will be stressed when there are many right-hand sides.

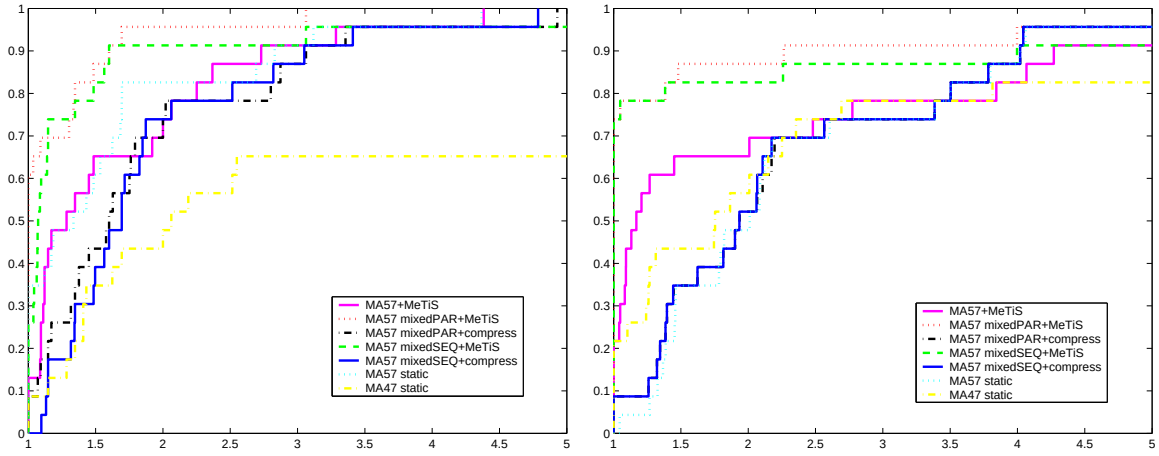
- The two `mixedSEQ` approaches do less iterative refinement steps than their corresponding `mixedPAR` approaches.

#itstep + 1	$berr \in$ $[0, 10^{-12}]$	$]10^{-12}, 10^{-8}]$	$]10^{-8}, 10^{-4}]$	$]10^{-4}, 1]$
`COMPRESS`	/ /x/ /x/x/x	/ /x/ /x/x/x	/ /x/ /x/x/x	/ /x/ /x/x/x
`mixedSEQ`	/ x/x/ / / /	/x/x/ / / /	/x/x/ / / /	/x/x/ / / /
`mixedPAR`	/ / /x/x/ /	/ / /x/x/ /	/ / /x/x/ /	/ / /x/x/ /
`static`	/ / / / /x/x	/ / / / /x/x	/ / / / /x/x	/ / / / /x/x
1	1/ 2/1/2/1/1/2	/ / / / / /	/ / / / / /	/ / / / / /
2	16/ 4/7/5/8/6/5	/ / /1/1/ /	/ / / / / /1	/1/ / / / /4
3	4/11/9/3/7/9/6	/ /1/ /1/1/	/ / / / / /	/ / / / / /
4	2/ 3/4/3/1/3/4	/ /1/ / / /	/ / / / / /	/ / / / / /
5	/ 1/ /4/2/1/1	/ / / / /1/	/ / / / / /	/ / / / / /
6	/ 1/ /2/ /1/	/ / / / / /	/ / / / / /	/ / / / / /
7	/ / / /1/ /	/ / /1/ / /	/ / / / / /	/ / / / / /
8	/ / / / / /	/ / / /1/ /	/ / / / / /	/ / / / / /
9	/ / / / / /	/ / / / / /	/ / / / / /	/ / / / / /
10	/ / / / / /	/ / / / / /	/ / / / / /	/ / / / / /
11	/ / / / / /	/ / / / / /	/ / / / / /	/ / / / / /
12	/ / / / / /	/ / / / / /	/ / / / / /	/ / / / / /
13	/ / /1/ / /	/ / / / / /	/ / / / / /	/ / / / / /
14	/ / / / / /	/ / / / / /	/ / / / / /	/ / / / / /
15	/ / /1/ / /	/ / / / / /	/ / / / / /	/ / / / / /

Table 3.3.2: Relation between range of component-wise backward error and number of iterative refi nement steps on set 2. Results given in the form `MA57`: MeTiS / `MA57`: `mixedSEQ`+MeTiS / `MA57`: `mixedSEQ`+`COMPRESS` / `MA57`: `mixedPAR`+MeTiS / `MA57`: `mixedPAR`+`COMPRESS` / `MA57`: `static`+`COMPRESS` / `MA47`: `static`+`COMPRESS`. #itstep is the number of iterative refi nement steps. The header lines are here to facilitate the reading of the table. Each header line corresponds to a criterion `COMPRESS` ordering, `mixedSEQ`, `mixedPAR` and `static` approaches. For each criterion, an x indicates the column that refers to a code that uses the corresponding criterion.

3.3.2.2 CPU factorization time

Figure 3.3.5.a shows the CPU factorization profile when a component-wise backward error greater than 10^{-4} is considered as a failure. The mixed approaches combined with MeTiS are clearly faster than the approach with numerical pivoting. Figure 3.3.5.b illustrates the fact that `MA47` is competitive in terms of number of operations and we think that it suffers again from too small sizes of blocks on which BLAS calls are made.



(3.3.5.a) CPU factorization time profile. (3.3.5.b) Number of operation profile.

Figure 3.3.5: Factorization time and number of operations. $berr$ failure $= 10^{-4}$

3.4 Conclusions

We have presented different choices for static pivoting for `MA57` and have identified their main characteristics. They seem to address a large class of problems and to be significantly faster than approaches with numerical pivoting.

Even if the precision of the solution is good, further improvements can be obtained. Firstly, there are still problems on which we get a backward error larger than 10^{-12}. Secondly, the number of iterative refinement steps could be decreased. This problem becomes all the more critical when we have many right-hand sides. In our future work, we would like to decrease the number of iterative refinement steps, trying to include 2×2 perturbations of the `static` approaches in the `mixedSEQ` and `mixedPAR` approaches . Hence, it might be useful to discover dynamically patterns of 2×2 pivots during the factorization and to have a criterion to decide between 1×1 or 2×2 perturbations.

We do not recommend our `MA47` based static version since it is not robust due to *tile* perturbations. Perhaps *tile* perturbations have to be revisited.

II

Chapter 4

Unsymmetric Ordering Using A Constrained Markowitz Scheme [1]

We consider the LU factorization of unsymmetric sparse matrices based on a three-phase approach (analysis, factorization and solution). In such an approach to reduce the amount of numerical problems during factorization, analysis may take into account the numerical values in order to provide a tentative pivot order that will also be numerically correct. To our knowledge, in most of the cases, such preprocessing phases are based on two separate steps. In the first step, the numerical data is used to scale the matrix and compute a permutation to put large entries on the diagonal. In the second step, to reduce the fill-in in the factors, a symmetric permutation preserving the diagonal entries is then computed. In this chapter, we present a preprocessing phase that simultaneously satisfies the objectives of selecting numerically good pivots while preserving the sparsity. By mixing the two objectives we show that we can reduce the amount of fill-in in the factors and reduce the amount of numerical problems during factorization. Note that the improvement in the numerical factorization resulting from this preprocessing is even more critical on distributed memory versions of the sparse solvers since numerical issues involve extra computation and communications.

4.1 Introduction

We consider the LU factorization of a sparse unsymmetric matrix A based on a three-phase approach (analysis, factorization, solve). The analysis phase transforms $\mathbf{A}$ into $\bar{\mathbf{A}}$ with better properties for sparse factorization. It exploits structural information to reduce the fill-ins in the LU factors and exploits numerical information to reduce the amount of numerical pivoting needed during factorization. Two consecutive treatments are commonly used for these two objectives. Firstly, scaling and maximum transversal algorithms are used to transform $\mathbf{A}$ into $\mathbf{A}_1$ with large entries in magnitude on the diagonal. Secondly, a symmetric fill-reducing ordering, which preserves the large diagonal, is used to permute $\mathbf{A}_1$ into $\mathbf{A}_2$ so that the factors of $\mathbf{A}_2$ are sparser than those of $\mathbf{A}_1$. Note that during factorization, numerical instabilities can still occur and will be handled either by partial pivoting resulting potentially in extra fill-in in the factor matrices or by static pivoting resulting in a potentially less accurate factorization.

During analysis, since numerical and structural considerations are separated, the numerical treatment requires the fill-reducing ordering to be symmetric, and the structural

[1] Part of this research was supported by a grant NSF-INRIA number NSF-INT-0003274.

phase does not have numerical information to correct any wrong numerical decisions. To avoid these two drawbacks, we present an approach which mixes the numerical and structural phases. Based on a numerical pretreatment of the matrix we build at each step k of the elimination a set of numerically acceptable pivots, referred to as matrix $\mathbf{C}^k$ that may contain off-diagonal entries. We then compute an unsymmetric ordering taking into account both the structure of $\mathbf{A}$ and the numerical information in $\mathbf{C}^k$. The $\mathbf{C}$ matrix serves as a **constraint matrix** in pivot selection. The new algorithm is referred to as *constrained Markowitz with local symmetrization* (or `CMLS`), in which local symmetrization is an idea developed in [12]. We present and illustrate this concept in Section 4.5.1.2.

The rest of the chapter is organized as follows. Section 4.2 presents the context of our study (related work and field of application). Section 4.3 introduces the main components of our algorithm. In Section 4.4, we describe a Matlab implementation including the main features of the algorithm, and demonstrate both structural and numerical benefits from this new approach. Section 4.5 defines the graph-theoretic notations and the structural and numerical metrics that will be used in the algorithm. In particular, our notations are summarized in Table 4.5.1. Section 4.6 gives the algorithmic details of the proposed `CMLS` method. Section 4.7 analyses the results of the newly implemented `CMLS` algorithm when applied to real-life unsymmetric test cases. Appendix A gives the data structures and implementation details. Appendix B presents exhaustive results.

4.2 Context of our study

We consider the LU factorization of unsymmetric sparse matrices based on a *three-phase approach*. It includes an *analysis* phase, a *numerical factorization* phase, and a *triangular solution* phase. The triangular solution may include iterative refinement to improve the accuracy of the solution. In this class of methods, without loss of generality, we will consider the multifrontal and the supernodal algorithms (see for example [6, 16, 28, 44, 45]). The analysis phase involves preprocessing of the matrix that may consider numerical values and a symbolic step that builds the computational graph for the numerical factorization phase. Both algorithms (supernodal and multifrontal) can be described by a *directed acyclic graph* [57] whose nodes represent computations and whose edges represent transfer of data. This graph reduces to a tree in the case of the multifrontal method. In a multifrontal algorithm, some steps of Gaussian elimination are performed on a dense frontal matrix at each node and the Schur complement (or contribution block) that remains is passed for assembly at the parent node. To continue our description of the three-phase approach we will focus on two state-of-the-art direct solvers representative of this class: the multifrontal code `MA41_UNS` [6, 13] and the supernodal code `SuperLU_DIST` [78, 79]. Both solvers can benefit from a numerical preordering (row or column permutations) to maximize entries on the diagonal and from a numerical scaling of the matrix, and can then generate an ordering (symmetric permutation of $\mathbf{A}$) based on an analysis of the pattern of $\mathbf{A}+\mathbf{A}^T$, where the summation is performed symbolically. The computation graph of the factorization phase is then computed assuming that diagonal pivots are numerically stable. As mentioned before,

this assumption is not reasonable. Therefore, during numerical factorization, both solvers will have, to some extent, to accommodate numerical stability issues.

For the MA41_UNS multifrontal code, standard partial pivoting with a threshold value is applied to select numerically stable pivots. It is thus possible that some variables cannot be eliminated from a frontal matrix. The rows and columns containing the non-eliminated variables of a frontal matrix are then added to the contribution block and passed to the father node. Those *delayed* eliminations will result in an increase in the size of the $\mathbf{LU}$ factors estimated during analysis and in an increase in the number of operations. In the case of the supernodal code, SuperLU_DIST, the distributed memory version uses a right-looking formulation which, having computed the factorization of a block of columns, then immediately sends the data to update the block columns in the trailing submatrix. A static pivoting strategy is used and we keep the pivotal sequence chosen in the analysis. The magnitude of the potential pivot is tested against a threshold of $\epsilon^{1/2}||\mathbf{A}||_1$, where ϵ is the machine precision and $||\mathbf{A}||_1$ is the one-norm of $\mathbf{A}$. If it is less than this value it is immediately set to this value (with the same sign) and the modified entry is used as pivot. This corresponds to a half-precision perturbation to the original matrix entry. The result is that the factorization is not exact and iterative refinement may be needed.

The importance of the numerical preprocessing of the matrix (scaling and permuting large entries onto the diagonal) is different for each solver. For MA41_UNS, it should limit the number of delayed pivots. The estimations performed during analysis will better reflect the reality of the numerical phase. For SuperLU_DIST, this numerical preprocessing is more crucial since it will reduce the number of small pivots that are modified and set to $\varepsilon^{\frac{1}{2}}||\mathbf{A}||_1$. In practice, it has been observed in [9] that using the MC64 code [41] from HSL [72] one can very significantly reduce the amount of numerical problems during factorization. Nevertheless this static approach still fails on some problems. One should add that taking into account numerical pivoting during factorization adds a significant level of complexity (dynamic irregular distributed data structures) to sparse distributed memory codes. In this case, limiting the amount of pivoting and/or having a better estimation during the analysis of the structures that will be involved during factorization is even more critical.

In this context, the problem of finding a good ordering for preserving the diagonal entries (symmetric ordering) but taking into account the asymmetry of the matrix was studied in [12]. To reduce the fill-in in the factors and improve the numerical quality of the preselected pivots we describe in this chapter, a family of orderings that selects off-diagonal pivots based on a combination of structural and numerical criteria. The work done in [12], which will be referred to as DMLS in the remainder of this document, has been generalized and extended in many directions. Firstly, we do not limit our choice of pivots to a transversal of the original matrix. Secondly, the pivots are chosen within a matrix of eligible pivots that needs to be updated. Thirdly, we also consider numerical values and numerical updates during this reordering phase so that we perform an incomplete factorization on the matrix of eligible pivots. Finally, because of all these modifications, we have to introduce new algorithms and new data structures to implement them as well as revisiting most of the algorithms described in [12].

4.3 Components of our unsymmetric ordering

Given a matrix $\mathbf{A}$, our unsymmetric ordering consists of two main steps:

- **Step 1:** Numerical preprocessing of $\mathbf{A}$ and construction of the constraint matrix, $\mathbf{C}$ such that:
  1. $\mathcal{P}attern(\mathbf{C}) \subset \mathcal{P}attern(\mathbf{A})$
  2. if $c_{ij} \neq 0$ then $c_{ij} = a_{ij}$
- **Step 2 :** Constrained unsymmetric ordering

Step 1 is a numerical preprocessing phase that controls the freedom given to the ordering phase. A matrix $\mathbf{C}$ is built in this first step and will be used to control the set of eligible pivots (possibly with respect to both numerical and structural criteria) at each step of Gaussian elimination in **Step 2**.

Compared with existing approaches, the expected improvement comes from the fact that firstly we do not limit our choice of pivots to a transversal of $\mathbf{A}$, and secondly numerical values can also be considered during pivot selection resulting in an ordering that gives a more stable numerical factorization. We now describe these two steps in detail.

4.3.1 Step 1: Numerical preprocessing

The objective of this preprocessing step is to extract the most significant (structurally and numerically) entries of matrix $\mathbf{A}$ and to use them to build the matrix $\mathbf{C}$.

As described in Algorithm 4.3.1, we first scale the matrix $\mathbf{A}$ resulting in $\mathbf{A} \leftarrow \mathbf{D}_r\mathbf{A}\mathbf{D}_c$. Secondly, a *constraint matrix* $\mathbf{C}$ can be constructed from $\mathbf{A}$ such that $\mathcal{P}attern(\mathbf{C}) \subset \mathcal{P}attern(\mathbf{A})$ and $\mathbf{C}$ satisfies certain numerical and/or structural properties. Since the entries in $\mathbf{C}$ correspond to the potential pivots for the subsequent step (Step 2), we keep the largest entries of the scaled matrix (greater in absolute value than a numerical threshold, $NumThresh$ which is chosen in practice between 0.01 and 1). Furthermore, we want $\mathbf{C}$ to be structurally nonsingular and thus we add entries from $\mathbf{A}$ to guarantee that $\mathbf{C}$ includes a maximum transversal $\mathcal{M}$. Note that $\mathcal{M}$ is also a maximum transversal of $\mathbf{A}$. To limit the size of $\mathbf{C}$ and the complexity (cost and memory) of the ordering phase, entries in $\mathbf{C}$ are further dropped until a reasonable size for $\mathbf{C}$ is reached (a threshold $StructThresh$ is used for this purpose and a value between $2n$ and $3n$ is often used in our experiments of Section 4.7). We remove from $\mathbf{C}$ in priority entries of small magnitude or entries that belong to the densest rows or columns of $\mathbf{C}$. Random dropping is applied if all the entries in $\mathbf{C}$ have the same magnitude and if all the row and columns of $\mathbf{C}$ have approximately the same number of nonzeros. Note that while dropping, we still maintain the property $\mathcal{M} \subset \mathbf{C}$.

Introducing this constraint matrix and exploiting it in the context of a quotient bipartite graph are the main contributions of this work. Full details on the construction and implementation of a matrix $\mathbf{C}$ are given in Section 4.6 and in Appendix A.

Algorithm 4.3.1 Generic preprocessing ($NumThresh$, $StructThresh$).

Compute row and column scalings of $\mathbf{A}$, $\mathbf{A} \leftarrow \mathbf{D}_r\mathbf{A}\mathbf{D}_c$.
Build matrix $\mathbf{C}$:
$\mathcal{P}attern(\mathbf{C}) = \{(i,j) \text{ s.t. } |a_{ij}| > NumThresh\}$,
Initialize the numerical values of entries in $\mathbf{C}$ if they will be needed,
Add entries from $\mathbf{A}$ to $\mathbf{C}$ so that $\mathbf{C}$ includes a maximum matching $\mathcal{M}$,
Remove entries from $\mathbf{C}$ (not in $\mathcal{M}$) so that $|\mathbf{C}| \leq StructThresh$.

4.3.2 Step 2: Constrained unsymmetric ordering

Based on the numerical pretreatment of the matrix, we build at each step k of the elimination a set of numerically acceptable pivots in matrix $\mathbf{C}^k$ that may contain off-diagonal entries. We select a pivot taking into account both the sparsity structure of $\mathbf{A}^k$ and the numerical and/or structural information in $\mathbf{C}^k$. Let $\mathbf{A}^1$ be the original matrix of order n and $\mathbf{A}^k$ be the reduced matrix after eliminating the first $k-1$ pivots (not necessarily on the diagonal). Let $\mathbf{C}^1$ be such that $\mathcal{P}attern(\mathbf{C}^1) \subset \mathcal{P}attern(\mathbf{A}^1)$. At each step k we select the best pivot p for a given metric such that $p \in \mathcal{P}attern(\mathbf{C}^k)$. Matrix $\mathbf{A}^k$ is updated (remove row and column of the pivot and add fill-ins in the Schur complement). Matrix $\mathbf{C}^k$ is updated so that $\mathbf{C}^{k+1}$ is structurally nonsingular and $\mathcal{P}attern(\mathbf{C}^{k+1})$ is included in the reduced matrix of $\mathbf{C}^k$ after p is eliminated. This implies that $\mathcal{P}attern(\mathbf{C}^{k+1}) \subset \mathcal{P}attern(\mathbf{A}^{k+1})$.

During the unsymmetric ordering, the k^{th} pivot $p \in \mathbf{C}^k$ can be selected using various structural local heuristics (e.g., approximate Markowitz count, approximate minimum fill, etc.). To avoid having a structurally singular $\mathbf{C}^k$, a maximum matching in $\mathbf{C}^k$ is maintained at each step.

4.3.3 Generic updating strategies for $\mathbf{C}$

Two considerations influence the update that will be performed on $\mathbf{C}$:

- which metric do we use to select a pivot?
- which entries and/or values are added/updated in $\mathbf{C}$ after eliminating a pivot?

If we consider only the structural properties of the $\mathbf{C}$ entries to select a pivot, only the pattern of $\mathbf{C}$ needs to be updated. Otherwise numerical values of the $\mathbf{C}$ entries need to be computed and/or updated.

The ordering algorithm also depends on how $\mathbf{C}$ is updated at each step. The description of **Step 2** (Section 4.3.2) implies the following two constraints:

$$\mathbf{C} \text{ must remain structurally nonsingular,} \tag{4.3.1}$$

$$\mathcal{P}attern(\mathbf{C}^{k+1}) \subset \mathcal{P}attern(\bar{\mathbf{C}^k}) \tag{4.3.2}$$

where $\bar{\mathbf{C}^k}$ represents the reduced matrix of $\mathbf{C}^k$ after current pivot p is eliminated.

There are two extreme strategies which satisfies these two conditions:

- **MATCHUPDATE** will refer to the strategy which only preserves the maximum matching property (4.3.1),

- **TOTALUPDATE** will refer to the strategy which performs all the updates in $\mathbf{C}$ ($\mathbf{C}^{k+1} = \bar{\mathbf{C}}^k$).

These two strategies are discussed in details in Section 4.6.4.

It is worth noting that when $StructThresh = n$ in Algorithm 4.3.1 we only keep the maximum matching in $\mathbf{C}$. Conditions 4.3.1 and 4.3.2 will thus limit the pivot choice to the matching and `CMLS` will behave similarly to diagonal Markowitz with local symmetrization `DMLS` [12] since, in this case, pivot selection is restricted to a transversal of the original matrix.

4.3.4 A specific implementation of Step 1: MC64 based preprocessing

In Algorithm 4.3.2, we describe a specific implementation of the generic Algorithm 4.3.1. In our preprocessing algorithm, we use a weighted bipartite matching and scaling algorithm implemented in `MC64` [40] to find the scaling matrices $\mathbf{D}_r$ and $\mathbf{D}_c$, such that the magnitudes of the entries of $\mathbf{A} \leftarrow \mathbf{D}_r\mathbf{A}\mathbf{D}_c$ are bounded by 1, and the entries corresponding to the matched edges in $\mathcal{M}$ are ± 1. We then construct $\mathbf{C}$ from $\mathbf{A}$ by dropping the entries that are smaller than a certain threshold ($NumThresh$). Since the scaled $\mathbf{A}$ satisfies $|a_{ij}| = 1$ for $(i,j) \in \mathcal{M}$ (`MC64` option 5), if $NumThresh \leq 1$ then all the entries in the matching will be selected in $\mathbf{C}$.

Algorithm 4.3.2 `MC64` preprocessing phase ($NumThresh$, $StructThresh$)

Check $0.0 \leq NumThresh \leq 1.0$ and $StructThresh \geq n$.
Apply the weighted bipartite matching algorithm (`MC64`) to $\mathbf{A}$ to obtain both
- row/column scalings $\mathbf{D}_r$ and $\mathbf{D}_c$, and
- a maximum weighted matching $\mathcal{M}$.

$\mathbf{A} \leftarrow \mathbf{D}_r\mathbf{A}\mathbf{D}_c$.
Let $\mathbf{C} = (c_{ij})$ such that:
for all entries of $\mathbf{A}$ **do**
  Numerical selection :
  if $|a_{ij}| > NumThresh$ **then**
    $c_{ij} = a_{ij}$
  else
    $c_{ij} = 0$
  end if
end for
Structural limitation :
while $size\ of\ \mathbf{C} > StructThresh$ **do**
  Remove entries (not in $\mathcal{M}$) in $\mathbf{C}$.
end while

4.4 Experiment in MATLAB

Before going any deeper in the description of the proposed algorithms, we report in this section a “simple” and partial Matlab based implementation of our two step approach. We are thus not concerned here with the memory and cost complexity of our implementation. We only want to do a preliminary study about in order to validate the effect of our

heuristics. In this section, we limit our study to a small set of relatively small matrices on which we illustrate the main features of our orderings. A complete and detailed analysis on larger problems (see Section 4.7) follows a complete description of the algorithms (see Section 4.6).

4.4.1 Strategies and parameters

4.4.1.1 Step 1

Algorithm 4.3.2 is used with $NumThresh = 0.9$ and no entries are removed in $\mathbf{C}$: $StructThresh$ value is not considered or is set to $+\infty$.

4.4.1.2 Step 2

In this step, the $\mathbf{C}$ matrix constructed in **Step 1** serves as a set of candidate pivots. The ordering algorithm depends on the structural metric that is computed from the structure of $\mathbf{A}$ and that is used to select the pivots in $\mathbf{C}$. Two structural metrics are considered:

- Markowitz cost (MC): we select the pivot which modifies the minimum number of entries in the reduced matrix.

$$m(i,j) = (row\_degree\_in\_\mathbf{A}(i) - 1)(col\_degree\_in\_\mathbf{A}(j) - 1).$$

- Minimum Fill (MF): we select the pivot that introduces the minimum number of new entries in the reduced matrix of $\mathbf{A}$.

The ordering algorithm also depends on how $\mathbf{C}$ is updated and how the numerical values in $\mathbf{C}$ are considered for the selection of a pivot. We use three updating strategies to illustrate the main feature of the approaches:

- Updating strategy 1 (**strat1**) : We do not consider numerical values in $\mathbf{C}$ and perform MATCHUPDATE.
  Advantage: it is in-place, and limits the cost of the $\mathbf{C}$ updates.
  Drawback: no numerical information is used during pivot selection.

- Updating strategy 2 (**strat2**) : We do not consider numerical values in $\mathbf{C}$ and perform TOTALUPDATE.
  Advantage: it gives a lot of flexibility to choose the pivots.
  Drawback: it is not in-place, and the $\mathbf{C}$ updates can be costly. No numerical information is used during pivot selection.

- Updating strategy 3 (**strat3**) : We consider numerical values in $\mathbf{C}$ and perform TOTALUPDATE. A pivot can be selected if and only if $|c_{ij}|/||c_{*j}||_\infty > 0.1$.
  Advantage: numerical information is used during pivot selection. It gives a lot of flexibility in choosing the pivots. We could also do dropping/no update for some rows/columns when they are too small and then have a less expensive update than strategy 2 (this is not considered in our MATLAB implementation).
  Drawback: it is not in-place, and numerical updates can be costly.

4.4.2 Results

Table 4.4.1 presents the test problems we used for this Matlab evaluation and also shows the maximum size of the matrices $\mathbf{C}^k$ which are introduced in Section 4.3.2. We notice that this maximum size is always equal to the initial size of $\mathbf{C}$ on our test set (in particular, $\max_k nnz(C^k)$ does not depend on the strategy used), which shows that our algorithm might not be too costly in terms of memory even when a TOTALUPDATE, which is not theoretically in-place, is used.

matrix	n	$nnz(\mathbf{A})$	$\max_k nnz(\mathbf{C}^k)$
MAHINDAS	1258	7682	2160
LHR01	1477	18427	2769
B_DYN	1089	4144	2213
B2_SS	1089	3895	2623
RADFR1	1048	13299	2356
GRE_1107	1107	5664	2719
EX23	1409	42760	2760

Table 4.4.1: Test problems and maximum size of $\mathbf{C}^k$ matrices.

In Tables 4.4.2, 4.4.3 and 4.4.4 we compare a Matlab implementation of our unsymmetric ordering with an approach combining `MC64` and `DMLS` [12].

matrix	Markowitz-cost metric						Minimum-fi ll metric			
	strat1		strat2		strat3		strat2		strat3	
	$ILU(C)$	ratio	$LU(C)$	ratio	$LU(C)$	ratio	$LU(C)$	ratio	$LU(C)$	ratio
MAHINDAS	2288	17	2398	18	2398	18	2400	19	2400	19
LHR01	2997	3	4014	6	3877	6	3686	7	3657	7
B_DYN	2301	27	2811	28	2889	34	2745	34	2718	33
B2_SS	2774	35	3773	49	3058	39	3687	49	3046	40
RADFR1	2602	7	3834	10	3699	8	3694	12	3457	10
GRE_1107	3073	5	6347	13	6003	12	6267	14	5960	13
EX23	2944	2	6053	5	5593	4	6402	7	6075	7

Table 4.4.2: Size of $\mathbf{C}$ "factors" versus size of $\mathbf{A}$ factors. The column $rat = |LU(C)|/|LU(A)|$ is given as a percentage.

Table 4.4.2 presents the sum of the row and column degrees of the pivots in $\mathbf{C}^k$ matrices (it corresponds to the size of LU factors of $\mathbf{C}$ for strategies 2 and 3 and to the size of an incomplete factorization for strategy 1) and the ratio between these quantities and the size of the factors of $\mathbf{A}$. We can see that the size of $\mathbf{C}$ does not increase too much for these problems and remains reasonable relative to the factors of $\mathbf{A}$. It gives us a first feeling about the complexity of our approach: the cost of $\mathbf{C}$ updates on this set of problems is dominated by the cost of the factorization step.

In Table 4.4.3, we compare the symbolic quality (number of nonzeros computed by the symbolic factorization) of the different strategies with that of the `DMLS` ordering. We use the Matlab LU factorization with no pivoting ($u = 0$) to evaluate the number of nonzeros in the factors predicted by the symbolic phase. The `DMLS` fill-in is always larger than the fill-in of our different strategies when using a minimum-fill metric (except with

matrix	Markowitz-cost metric				Minimum-fill metric		
	strat1	strat2	strat3	DMLS	strat2	strat3	DMLS
MAHINDAS	13358	12883	12883	16946	12574	12574	15699
LHR01	87247	69656	65032	65317	51829	52216	59070
B_DYN	8337	8162	8168	8802	8055	8071	8795
B2_SS	7887	7627	7657	9032	7392	7484	8393
RADFR1	37413	35922	46649	69165	31273	34986	31373
GRE_1107	59104	46137	46633	84457	42864	42887	70113
EX23	141853	122437	134312	96513	94837	93744	131583

Table 4.4.3: Number of nonzeros predicted by analysis for the factors of $\mathbf{A}$. Comparison of MC64+DMLS with a Matlab implementation of MC64+CMLS.

RADFR1 for which factors are 11.5% more dense). Our Markowitz-cost metric seems to have difficulties on EX23.

Matrix	strat2		strat3		DMLS	
	lu u=0	lu u=0.01	lu u=0	lu u=0.01	lu u=0	lu u=0.01
MAHINDAS	12574	12574	12574	12574	15699	15699
LHR01	51829	52613	52216	52216	59070	59023
B_DYN	8055	8074	8071	8071	8795	8795
B2_SS	7392	7478	7484	7484	8393	8455
RADFR1	31273	61681	34986	34030	31373	32935
GRE_1107	42864	46495	42887	44662	70113	70721
EX23	94837	109531	93744	93744	131583	134791

Table 4.4.4: Number of nonzeros in the factors and reliability of the forecast factors size: comparison of MC64+DMLS with a Matlab implementation of MC64+CMLS. Minimum fill metric is used in both cases.

Matrix	strat2		strat3		MC64+DMLS	
	lu u=0	lu u=0.01	lu u=0	lu u=0.01	lu u=0	lu u=0.01
MAHINDAS	7e-16	7e-16	7e-16	7e-16	4e-16	4e-16
LHR01	1e-10	1e-14	5e-15	5e-15	2e-14	6e-15
B_DYN	7e-15	1e-15	7e-16	7e-16	4e-16	4e-16
B2_SS	NaN	7e-14	4e-15	4e-15	NaN	3e-15
RADFR1	NaN	7e-10	3e-11	1e-11	NaN	5e-11
GRE_1107	1e-10	3e-11	5e-11	7e-12	4e-11	2e-11
EX23	1e14	6e-14	8e-14	8e-14	2e01	3e-14

Table 4.4.5: Numerical accuracy comparison of MC64+DMLS with a MATLAB implementation of our unsymmetric ordering (Approximate minimum fill metric is used in both cases).

Table 4.4.4 compares the number of nonzeros in the factors and Table 4.4.5 compares the forward error ($err = ||xtrue - x||_1/||x||_1$). We use the Matlab LU factorization both with no pivoting ($u = 0$) and with a more common pivoting threshold $u = 0.01$ (this will introduce extra fill-ins due to numerical pivoting). We see that, except for RADFR1 with Strategy 2 for which the size of the factors increases from 31273 to 61681, the extra freedom in selecting off-diagonal pivots in $\mathbf{C}^k$ with Strategies 2 and 3 results in a reduction of the fill-in with respect to MC64+DMLS approach. The size of the factors does not increase much from columns "$u = 0$" to columns "$u = 0.01$" in Table 4.4.4, and

this is so even with strategy 2 where we only consider the pattern of $\mathbf{C}$. Finally, we see in Table 4.4.5 that strategy 3 is very stable even when no numerical pivoting is performed during the factorization. Using Strategy 3, we should be able to relax the numerical threshold during the factorization when using a solver that performs numerical pivoting. Moreover, the static pivoting scheme used in `SuperLU_DIST` may get benefits from Strategy 3 with the improved accuracy and/or the reduction of the number of iterative refinement steps. This last point will be stressed in Section 4.7.3.

4.5 Notations and definitions

4.5.1 Bipartite Graphs

The structure of an unsymmetric matrix can be represented as a bipartite graph, and Gaussian elimination can be efficiently modelled by a bipartite quotient graph [86]. In this section, we describe the main properties of bipartite graphs and bipartite quotient graphs and their relationship with Gaussian elimination. We introduce the notations that will be used to describe our algorithms. Note that we will use calligraphic letter only for notations concerning quotient graph.

4.5.1.1 Bipartite graph

Let us first recall the notations about bipartite graphs that we presented in the introduction of the thesis. Let $\mathbf{B} = (b_{ij})$ be a matrix and $G_B = (V_r, V_c, E)$ be its associated bipartite graph. V_r is the set of row vertices and V_c is the set of column vertices. Edges of E only go from V_r to V_c ($E \subset V_r \times V_c$). An edge $(i,j) \in E$ exists if and only if $b_{ij} \neq 0$. Let R_i denotes the structure of row i, *i.e.*, $R_i = \{j \in V_c \ s.t. \ (i,j) \in E\}$. Let C_j denote the structure of column j, *i.e.*, $C_j = \{i \in V_r \ s.t. \ (i,j) \in E\}$.

After a pivot (i,j) is eliminated in $\mathbf{B}$, a new matrix which we denote by $\bar{\mathbf{B}}$ is obtained. $\bar{\mathbf{B}}$ is constructed from $\mathbf{B}$ by removing row i and column j and by adding the Schur complement entries. In terms of graph manipulations, this elimination adds edges in the bipartite graph of $\mathbf{B}$ to connect each row adjacent to j to all the columns adjacent to i. This set of connected rows and columns is referred to as a **bi-clique**.

The symbolic factorization of $\mathbf{B}$ is done by building $\mathbf{B}^p$ for $p = 1$ to n, with $\mathbf{B}^1 = \mathbf{B}$. After eliminating the p^{th} pivot $(rowp, colp)$ in $\mathbf{B}^p$, we compute $\mathbf{B}^{p+1} = \bar{\mathbf{B}}^p$.

More formally, the bipartite graph changes from G_{B^p} to $G_{B^{p+1}}$ with

$$R_i^{p+1} = (R_i^p \cup R_{colp}^p) \setminus \{rowp\} \text{ for } i \in C_{colp}^p$$

and

$$C_j^{p+1} = (C_j^p \cup C_{rowp}^p) \setminus \{colp\} \text{ for } j \in R_{rowp}^p$$

and $rowp$ and $colp$ are removed from the vertex sets of $G_{B^{p+1}}$. At each step p of the symbolic factorization, we define

$$k^p = \max(\max_{i \in V_r} |R_i^p|, \max_{j \in V_c} |C_j^p|).$$

This corresponds to the largest row or column length and will be used for complexity analysis. When there is no ambiguity, we will omit the superscript p from the graph and matrix notations. Moreover, we will use the notation L_p for R^p_{rowp} and U_p for C^p_{colp}.

4.5.1.2 Bipartite quotient graph

In the previous section we have shown that, to update the bipartite graph at each elimination step, we must add the entries to the Schur complement matrix. The associated subgraphs has been referred to as bi-cliques in the bipartite graph and may be costly to update and to store. The main idea of quotient graphs is to use compact representations to implicitly store edges between vertices which would exist if a simple bipartite graph were used. In symbolic LU factorization, a bipartite quotient graph can be used to represent the edges in a bi-clique. It can be shown that a bipartite quotient graph can be used to model the elimination in space bounded by the size of the original matrix $\mathbf{A}$ [86]. In this section, we first explain why the quotient graph model leads to more complex algorithms on the bipartite graphs (unsymmetric matrices) than on the undirected graphs (symmetric matrices). We then briefly define element absorption and local symmetrization and explain why they help reduce the quotient graph complexity. Finally we introduce the notations that will be used to describe our algorithms.

Let $\mathcal{G}^k$ be the bipartite quotient graph used to represent the structure of the reduced submatrix after k steps of elimination. Both row and column vertices of the graph are partitioned into two sets referred to as **variables** (uneliminated vertices) and **elements** (eliminated vertices). Initially the bipartite quotient graph $\mathcal{G}^1$ is identical to the bipartite graph G^1 and all the vertices are variables. At step k of Gaussian elimination, an eliminated pivot $e = (r_e, c_e)$ has two vertices r_e and c_e in $\mathcal{G}^k$, referred to as the **coupled row and column elements**. All row and column vertices not belonging to any coupled elements are the row and column variables of $\mathcal{G}^k$. If there exists a path between two variables i and j then j is **reachable** from i and vice-versa. The adjacency of a row (resp. column) variable in the bipartite graph G^k is the set of reachable columns (resp. rows) through the coupled row/column elements associated with the previous pivots in $\mathcal{G}^k$. Note that in a unsymmetric context, this process may involve paths of arbitrary length in $\mathcal{G}^k$ [86] and in particular through more than one coupled element. For example on Figure 4.5.1, we need to consider the path $rowp \rightarrow c_{e1} \rightarrow r_{e1} \rightarrow c_{e2} \rightarrow r_{e2}$ to know that the row structure of $rowp$ contains the row structure of $e1$.

In the context of sparse Cholesky factorization, an undirected quotient graph (the row and column vertices are merged) is preferred and commonly used to compute the orderings for symmetric matrices, e.g., Multiple Minimum Degree [80] and Approximate Minimum Degree [4]. Moreover, the structure of the factors can be computed following paths of length at most two in this quotient graph. The sparse Cholesky factorization can be also viewed in the context of a bipartite quotient graph. The structure of the factors can then be computed following the paths of length at most three in the bipartite quotient graph (the path in $\mathcal{G}^k$, $r_i \rightarrow c_e \rightarrow r_e \rightarrow c_j$ indicates that column index $c_j \in \mathcal{V}_c$ is adjacent to index $r_i \in \mathcal{V}_r$ through the element $e = (r_e, c_e)$). Moreover, there are no edges between the elements which belong to different pivots.

In the unsymmetric case, let $\mathcal{G}^k = (\mathcal{V}_r \cup \overline{\mathcal{V}}_r, \mathcal{V}_c \cup \overline{\mathcal{V}}_c, \mathcal{E} \cup \overline{\mathcal{E}})$ be the bipartite quotient graph.

Vertices in $\mathcal{V}_r$ (respectively $\mathcal{V}_c$) correspond to row (respectively column) **variables** (uneliminated vertices). Vertices in $\overline{\mathcal{V}}_r$ (respectively $\overline{\mathcal{V}}_c$) correspond to row (respectively column) **elements** (already selected pivots). $\mathcal{V}_r$, $\mathcal{V}_c$, $\overline{\mathcal{V}}_r$, $\overline{\mathcal{V}}_c$ is a partition of the vertices of the graph, $\mathcal{E} \subset (\mathcal{V}_r \times \mathcal{V}_c)$ and $\overline{\mathcal{E}} \subset (\mathcal{V}_r \times \overline{\mathcal{V}}_c) \cup (\overline{\mathcal{V}}_r \times \mathcal{V}_c) \cup (\overline{\mathcal{V}}_r \times \overline{\mathcal{V}}_c)$. An entry (r, c) corresponds to a nonzero in the factors if and only if there exists a path joining r and c which only visits the elements and for which an edge in the even position corresponds to a selected pivot. In other words, the structure of a row i at step k is the set of reachable columns j through all the paths of the form $i \rightarrow c_{e_1} \rightarrow r_{e_1} \ldots \rightarrow c_{e_l} \rightarrow r_{e_l} \rightarrow j$ where $e_i = (r_{e_i}, c_{e_i}), 1 \leq i \leq l$ are coupled elements associated with the pivots. Similarly, the structure of a column j at step k is the set of reachable rows i through all the paths of the form $j \rightarrow r_{e_1} \rightarrow c_{e_1} \ldots \rightarrow r_{e_l} \rightarrow c_{e_l} \rightarrow i$. When a pivot $p = (rowp, colp)$ is selected, if there exists a cycle of the form $rowp \rightarrow c_{e_1} \rightarrow r_{e_1} \ldots \rightarrow c_{e_l} \rightarrow r_{e_l} \rightarrow colp \rightarrow rowp$ (referred to as the **strongly connected components** in [86]) then, except for $rowp$ and $colp$, the row and column elements in the cycle are not needed any more to retrieve the structures of the remaining variables. Firstly, all the row (resp. column) variables adjacent to these column (resp. row) elements will be included in the adjacency of $colp$ (resp. $rowp$), and secondly, the row (resp. column) variables which were adjacent to one of the removed elements will be adjacent to $colp$ (resp. $rowp$). This process will be called **element absorption** and is illustrated in Figure 4.5.1.

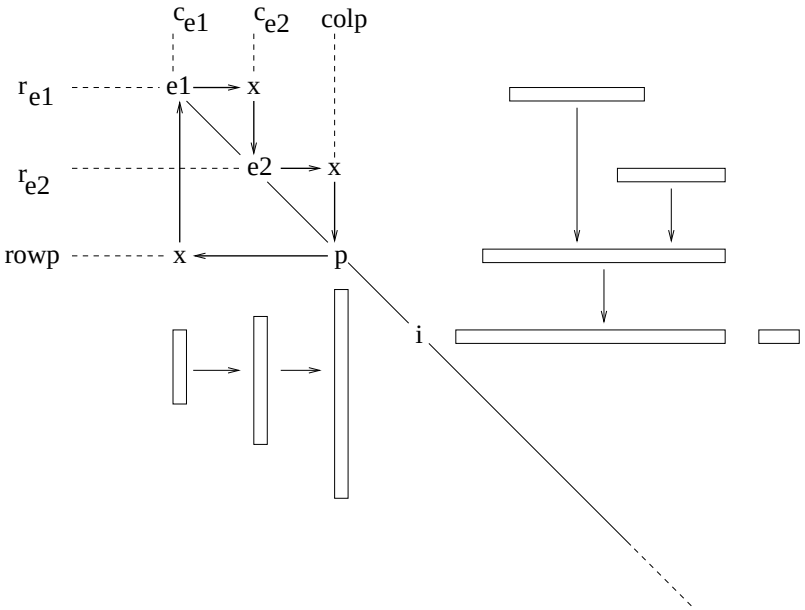


Figure 4.5.1: Illustration of a cycle ($rowp \rightarrow c_{e_1} \rightarrow r_{e_1} \rightarrow c_{e_2} \rightarrow r_{e_2} \rightarrow colp \rightarrow rowp$).

To avoid long search paths, we decide to relax the element absorption rule. A row (resp. column) element is absorbed by the current row (resp. column) pivot if either it is adjacent to the column (resp. row) pivot or its associated column (resp. row) element is adjacent to the row (resp. column) pivot. This is referred to as **local symmetrization** [12]. It implies that the resulting quotient graph $\mathcal{G}^k$ at step k models only an approximation of the structure of the reduced submatrix. It has been shown in [12] that the exploitation of element absorption combined with local symmetrization results in an in-place algorithm:

at each step of the Gaussian elimination, the size of the quotient graph is bounded by the size of $\mathcal{G}_1$. The main argument of the proof is that the space required to build the row and column structures of a pivot is smaller that the sum of the size of the structures of the absorbed elements. Note that because of local symmetrization, an approximation of the symbolic factors can be computed following paths of length at most three of the form $i \rightarrow c_e \rightarrow r_e \rightarrow j$ with (r_e, c_e) denoting a coupled row and column element.

To simplify the description of how the bipartite quotient graph is modified at each elimination step, we define $\overline{\mathcal{V}} \subset (\overline{\mathcal{V}}_r \times \overline{\mathcal{V}}_c)$ to be the set of row-column couples (r_l, c_l) corresponding to the pivots eliminated at steps l , $1 \leq l \leq k$. The entries of the set $\overline{\mathcal{V}}$ will be referred to as **coupled elements** or **elements** when it is clear from the context. Let $\mathcal{U}_p$ (resp. $\mathcal{L}_p$) be the row (resp. column) variables adjacent in $\mathcal{G}^k$ to the row (resp. column) element of a pivot $p = (rowp, colp)$. In the context of local symmetrization, the concept of absorption can be extended to the coupled element: an element $e = (r_e, c_e)$ such that $(rowp, c_e) \in \overline{\mathcal{E}}$ **or** $(r_e, colp) \in \overline{\mathcal{E}}$ is said to be absorbed by p when p is selected as a pivot.

For each row variable $i \in \mathcal{V}_r$ and column variable $j \in \mathcal{V}_c$, we define the element lists as follows:

$$\mathcal{R}_i = \{e = (r_e, c_e) \in \overline{\mathcal{V}} \text{ s.t. } (i, c_e) \in \overline{\mathcal{E}}\}$$

and

$$\mathcal{C}_j = \{e = (r_e, c_e) \in \overline{\mathcal{V}} \text{ s.t. } (r_e, j) \in \overline{\mathcal{E}}\}.$$

Let $e = (r_e, c_e)$ be an element, if $e \in \mathcal{R}_i$ then we will say that **element** e **is adjacent to row variable** i. Similarly, if $e \in \mathcal{C}_j$ we will say that **element** e **is adjacent to column** j.

Using this notation, the adjacency of a row variable i (resp. column j) in $\mathcal{G}$ consists of a list of column variables denoted as $\mathcal{A}_{i*}$ (resp. a list of row variables $\mathcal{A}_{*j}$) and a list of elements $\mathcal{R}_i$ (resp. $\mathcal{C}_j$). At the beginning $\mathcal{R}_i = \mathcal{C}_j = \emptyset$ and $\mathcal{A}_{i*} = \mathcal{A}_{i*}^{(0)}$ and $\mathcal{A}_{*j} = \mathcal{A}_{*j}^{(0)}$ contain the original entries of $\mathbf{A}$. Each step of Gaussian elimination involves changes in the set $\mathcal{R}_i$ and $\mathcal{C}_j$ as well as the computation of the structure of a current pivot p (computation of $\mathcal{L}_p$ and $\mathcal{U}_p$). The variable lists $\mathcal{A}_{i*}$ and $\mathcal{A}_{*j}$ can also be pruned. Indeed, the edges in $\mathcal{G}$ between the variables and the elements implicitly represent the bi-clique of the element and can thus be used to remove the redundant entries in $\mathcal{A}_{i*}$ and $\mathcal{A}_{*j}$. This important point will be further discussed in detail in Section 4.6.5. It can be seen that manipulating the sets $\mathcal{R}_i$, $\mathcal{C}_j$, $\mathcal{A}_{i*}$ and $\mathcal{A}_{*j}$ is equivalent to manipulating the vertices and the edges of the bipartite quotient graph. Therefore, we will limit our algorithmic description to the manipulation of these sets.

When $(i, j) \in \mathcal{V}_r \times \mathcal{V}_c$ is selected as the next pivot we build the element $p = (i, j) \in \mathcal{G}$ such that:

$$\mathcal{U}_p = \mathcal{A}_{i*} \cup \bigcup_{e \in \mathcal{R}_i} \mathcal{U}_e \cup \bigcup_{e \in \mathcal{C}_j} \mathcal{U}_e \tag{4.5.1}$$

and

$$\mathcal{L}_p = \mathcal{A}_{*j} \cup \bigcup_{e \in \mathcal{C}_j} \mathcal{L}_e \cup \bigcup_{e \in \mathcal{R}_i} \mathcal{L}_e. \tag{4.5.2}$$

The third term in each of the two equations above results from local symmetrization and will enable the current pivot to absorb all the elements which it was adjacent to. For

example, let us suppose that the entry $p1$ is selected as pivot in Figure 4.5.2. Since col_{p1} is adjacent to $e1$, the local symmetrization add the virtual $Sp1$ and so the row structure of $p1$ contains L_{e1}. Note that this symmetrization only concerns dependencies between elements. Let $\mathcal{F} = \mathcal{C}_j \cup \mathcal{R}_i$ be the set of elements adjacent to the current pivot. The elements in $\mathcal{F}$ are absorbed by p and can be removed from the quotient graph. For each column variable j_1 in $\mathcal{U}_p$ (resp. i_1 in $\mathcal{L}_p$) element p is added to its adjacency and the elements in $\mathcal{F}$ are removed: $\mathcal{R}_{j_1} \leftarrow (\mathcal{R}_{j_1} \setminus \mathcal{F}) \cup \{p\}$ (resp. $\mathcal{C}_{i_1} \leftarrow (\mathcal{C}_{i_1} \setminus \mathcal{F}) \cup \{p\}$). The structure of row i_1 of the factors in the reduced matrix is then given by

$$\mathcal{A}_{i_1 *} \cup \bigcup_{e \in \mathcal{R}_{i_1}} \mathcal{U}_e \ ,$$

and the structure of column j_1 of the factors is

$$\mathcal{A}_{* j_1} \cup \bigcup_{e \in \mathcal{C}_{j_1}} \mathcal{L}_e \ .$$

Note that this information on the structure of the reduced submatrix, although correct, is not useful for predicting what happens after the next pivot is selected and so will not be used in our algorithms to compute the structural metrics. Indeed, if (i_1, j_1) is selected as the next pivot, then the correctly computed structure of the reduced matrix should include the local symmetrization terms (similar to equations (4.5.1) and (4.5.2)). Thus, the local symmetrization terms will be taken into account for computing the structural metrics of all the entries in the reduced matrix.

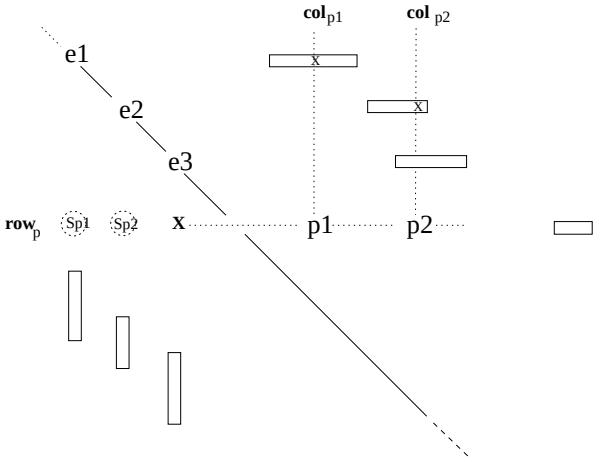


Figure 4.5.2: Influence of local symmetrization on the pivot structure

In Figure 4.5.2, we illustrate the effect of local symmetrization on the structure of the selected pivot. Let us consider two candidate pivots belonging to the same row row_p, $p1 = (row_p, col_{p1})$ and $p2 = (row_p, col_{p2})$. We assume that all the elements in $\mathcal{G}$ adjacent to $p1$ and $p2$ are indicated in the figure. The structure of row_p is then given by $\mathcal{A}_{row_p *} \cup \mathcal{U}_{e_3}$. This however does not give enough information on the structure of row_p if either $p1$ or $p2$ were selected as the next pivot. Local symmetrization would apply (see equation (4.5.1)) so that if $p1$ were the next pivot then the structure of row_p would be

Bipartite graph G		Bipartite quotient graph $\mathcal{G}$	
R_i	structure of row i	$\mathcal{R}_i$	elements adjacent to row i
		$\mathcal{A}_{i*}$	variables adjacent to row i
A_{i*}	initial structure of row i	$\mathcal{A}_{i*}^{(0)}$	initial structure of row i
C_j	structure of column j	$\mathcal{C}_j$	elements adjacent to column j
		$\mathcal{A}_{*j}$	variables adjacent to column j
A_{*j}	initial structure of column j	$\mathcal{A}_{*j}^{(0)}$	initial structure of column j
U_p	row structure of the p^{th} pivot	$\mathcal{U}_p$	row structure of the p^{th} pivot
L_p	column structure of the p^{th} pivot	$\mathcal{L}_p$	column structure of the p^{th} pivot
		$\mathcal{F}$	elements that are adjacent either to row i or column j ($\mathcal{R}_i \cup \mathcal{C}_j$)
k^p	at step p, the size of the largest adjacency		

Table 4.5.1: Notations used for bipartite graph and bipartite quotient graph.

given by :

$$\mathcal{U}_{p1} = \mathcal{A}_{i*} \cup \mathcal{U}_{e_3} \cup \mathcal{U}_{e_1}$$

because of the local symmetrization entry S_{p1}. If $p2$ were the next pivot then the structure of row_p would be given by :

$$\mathcal{U}_{p2} = \mathcal{A}_{i*} \cup \mathcal{U}_{e_3} \cup \mathcal{U}_{e_2}$$

because of the local symmetrization entry S_{p2}. This shows that, even if we cannot anticipate the effect of local symmetrization on the quotient graph $\mathcal{G}$ before the pivot selection, we should anticipate its effect on the metrics used to select the best pivot between p_1 and p_2.

4.5.2 Metrics and ordered relations

As observed in Section 4.4, hybrid metrics (that is combining structural and numerical criteria) might be considered for selecting a pivot. In this section, we define the notations and relationships that will be needed to describe the pivot selection strategies in the `CMLS` algorithms (see Section 4.6.3)

DEFINITION 5.1. *For each entry (i, j), $m_0(i, j)$ will represent a structural metric, such as an approximate Markowitz cost or an approximate minimum fill-in.*

If we consider two entries (i_1, j_1) and (i_2, j_2) such that $m_0(i_1, j_1) < m_0(i_2, j_2)$ then we will say that (i_1, j_1) **is structurally better than** (i_2, j_2).

When considering numerical values, we need to determine if a pivot is good/bad (in an absolute sense) and/or if it is better than the other pivots (for example, relative to the largest value in magnitude in its row/column). The rest of this section is organized as follows:

- we first define the ordered relation $\leq^{th}$ as a **function of a numerical threshold** th;
- this relation is then used to define the relations $\leq_{th}$ and $\leq_{rel\theta}^{abs\theta}$ that combine numerical and structural criteria.

In the following, th, $rel\theta$ and $abs\theta$ are positive real values. In the CMLS algorithm, $rel\theta$ and $abs\theta$ will be criteria associated with a relative numerical threshold and an absolute numerical threshold respectively.

DEFINITION 5.2. *Let $\leq^{th}$ be an ordered relation such that for two values v_1 and v_2 we have:*

$$v_1 =^{th} v_2 \Leftrightarrow \begin{cases} |v_1| \geq th \text{ and } |v_2| \geq th\ , \text{ or} \\ |v_1| < th \text{ and } |v_2| < th \end{cases}$$

$$v_1 <^{th} v_2 \Leftrightarrow |v_1| \geq th \text{ and } |v_2| < th.$$

The above inequality is not presented the wrong way round. When we use $v_1 \leq^{th} v_2$ we really mean that v_1 is better than v_2 by analogy to the sense of the inequality in the context of a structural metric. Thus, if $v_1 <^{th} v_2$, we will say that v_1 **is numerically better than** v_2 **according to** th. The $\leq^{th}$ relation simply indicates that the numerical quality of two entries with absolute values greater than th is the same and that the numerical quality of two entries with absolute values smaller than th is also the same. A strict inequality means that exactly one value is greater than th.

DEFINITION 5.3. *Let $\leq_{th}$ be an ordered relation such that for two entries (i_1, j_1) and (i_2, j_2) with associated numerical values v_1 and v_2, we have:*

$$(i_1, j_1, v_1) \leq_{th} (i_2, j_2, v_2) \Leftrightarrow \begin{cases} v_1 <^{th} v_2, \text{ or} \\ v_1 =^{th} v_2 \text{ and } m_0(i_1, j_1) \leq m_0(i_2, j_2)) \end{cases}.$$

This relation is nothing but a lexicographical order where priority is given to numerical information. The structural metric is used in this comparison only when the numerical metrics are identical with respect to the $\leq^{th}$ relation. This means that the structural metric will be considered only when both values are numerically good (greater than th in magnitude) or numerically bad (smaller than th in magnitude).

For example, for two entries of a constraint matrix $\mathbf{C}$ in positions (i_1, j_1) and (i_2, j_2), we may define the numerical values

$$v_1 = \frac{c_{i_1 j_1}}{\max_k |c_{k j_1}|} \text{ and } v_2 = \frac{c_{i_2 j_2}}{\max_k |c_{k j_2}|}$$

where we have scaled the values by the infinite norm of the column. Then we prefer to select (i_1, j_1) as pivot if v_1 is numerically better than v_2 according to th or if both entries are of comparable numerical quality (both magnitudes of v_1 and v_2 are either lower or greater than th) and the entry (i_1, j_1) is structurally better than the entry (i_2, j_2). In that case, $(i_1, j_1, v_1) <_{th} (i_2, j_2, v_2)$ and we say that (i_1, j_1, v_1) **is better than** (i_2, j_2, v_2) **according to the hybrid metric with the single threshold** th.

Relation $\leq_{th}$ can be extended to two numerical criteria. Before presenting this metric with two thresholds, let us give some motivation for this approach (this approach is also discussed in Section 4.6.3). If we consider the matrix $\mathbf{A} = \begin{pmatrix} 1 & 10^{-8} & \dots \\ \vdots & \vdots & \dots \\ 1 & 1 & \dots \end{pmatrix}$ and

its constraint matrix $\mathbf{C} = \begin{pmatrix} 1 & 10^{-8} & \dots \\ \vdots & \vdots & \dots \\ 1 & 0 & \dots \end{pmatrix}$ where the 1 entry in the last row and second column of $\mathbf{A}$ has been dropped. We want then to decide which entry of $\mathbf{C}$ is the "best". For example, we want to decide between entries in positions $(1,1)$ and $(1,2)$. If we use an ordered relation with the single threshold and if we scale each value by the infinite norm of its column, our decision will be guided by the structural metric (we have $v_1 =^{th} v_2$). Selecting the entry in position $(1,2)$ will be less safe than selecting the entry in position $(1,1)$ since a stable Gaussian elimination has to be performed on $\mathbf{A}$. Thus, as c_{12} is significantly smaller than c_{11} we prefer to select the entry in position $(1,1)$ without taking into account the structural metric. That is why we decide to add a second numerical criterion. Definition 5.4 presents the ordered relation that we will use to perform this pivot selection. It takes into account two values, that we call **the scaled value** and **the unscaled value**.

DEFINITION 5.4. *Let $\leq_{rel\theta}^{abs\theta}$ be an ordered relation such that for two entries (i_1, j_1) and (i_2, j_2), each with two associated numerical values, s_1 and u_1 for (i_1, j_1), and s_2 and u_2 for (i_2, j_2), we have:*

$$(i_1, j_1, s_1, u_1) \leq_{rel\theta}^{abs\theta} (i_2, j_2, s_2, u_2) \Leftrightarrow \begin{cases} s_1 <^{rel\theta} s_2 \\ \text{or } (s_1 =^{rel\theta} s_2 \text{ and } (i_1, j_1, u_1) \leq_{abs\theta} (i_2, j_2, u_2)) \end{cases}.$$

This relation corresponds to a lexicographical order in which priority is given firstly to a numerical metric based on the relation $\leq^{rel\theta}$, secondly to a numerical metric based on the relation $\leq^{abs\theta}$, and finally to a structural metric m_0. For example, for two entries (i_1, j_1) and (i_2, j_2) of a constraint matrix $\mathbf{C}$, we set

$$s_1 = \frac{c_{i_1 j_1}}{\max_k |c_{k j_1}|}, s_2 = \frac{c_{i_2 j_2}}{\max_k |c_{k j_2}|}, u_1 = c_{i_1 j_1} \text{ and } u_2 = c_{i_2 j_2}.$$

Then we prefer to select (i_1, j_1) as pivot if s_1 is numerically better than s_2 according to $rel\theta$ or if both entries are of comparable numerical quality and (i_1, j_1, u_1) is numerically better than (i_2, j_2, u_2) according to the hybrid metric with the single threshold $abs\theta$. In that case $(i_1, j_1, s_1, u_1) <_{rel\theta}^{abs\theta} (i_2, j_2, s_2, u_2)$ and we say that (i_1, j_1, s_1, u_1) **is better than** (i_2, j_2, s_2, u_2) **according to the hybrid metric with two thresholds** $rel\theta$ **and** $abs\theta$.

Note that the ordered relation induced by m_0 corresponds to the case $\leq_{th}$ with $th = 0$ and that $\leq_{th}$ corresponds to the order $\leq_{rel\theta}^{abs\theta}$ with $abs\theta = 0$ and $rel\theta = th$.

4.6 CMLS algorithm

In this section, we describe the different steps of the `CMLS` algorithm and their properties. Algorithm 4.6.3 presents the main steps of our approach. Section 4.6.1 discuss algorithmic choices about the structures of $\mathbf{A}$ and $\mathbf{C}$. Section 4.6.2 gives details about the initialization of the constraint matrix. In Section 4.6.3, we expose algorithms that are used to select pivots. Sections 4.6.4 and 4.6.5 show how the structures of $\mathbf{C}$ and $\mathbf{A}$ respectively are updated. Section 4.6.6 describes how for each entry in $\mathbf{C}$ our structural

metrics that take into account the information stored in $\mathbf{A}$ are updated. Section 4.6.7 shows how supervariables are used in CMLS and Section 4.6.8 discusses about aggressive absorption.

Algorithm 4.6.3 CMLS main steps.

(1) Initialization of $\mathbf{C}$ (see Section 4.6.2 and Algorithm 4.3.1).
(2) Main loop
while $k \leq n$ **do**
(2.1) Select pivot $p = (rowp, colp)$ in $\mathbf{C}$ with priority based on structural and/or numerical metric (see Section 4.6.3).
(2.2) Build $\mathcal{L}_p$, $\mathcal{U}_p$ and let $\mathcal{F}_p = \mathcal{C}_{colp} \cup \mathcal{R}_{rowp}$.
(2.3) Update $\mathbf{C}$ (strategies described in Section 4.6.4) considering numerical information if a *hybrid metric* is to be used (see Section 4.6.6).
(2.4) Update quotient graph $\mathcal{G}$ of $\mathbf{A}$ (see Section 4.6.5).
(2.5) For each $(i,j) \in \mathbf{C}$ such that $i \in \mathcal{L}_p$ or $j \in \mathcal{U}_p$, update the structural metric (see Sections 4.6.6).
(2.6) Convert variable p to an element and remove the absorbed elements
$k = k + 1$
end while

4.6.1 Representations of $\mathbf{C}$ and $\mathbf{A}$: two graphs

In the CMLS algorithm, we need to know the exact structure and the metric of each nonzero entry in $\mathbf{C}$. It is natural to use a bipartite graph (with possibly weighted edges) for $\mathbf{C}$. Each edge that corresponds to a nonzero entry may have one or more weights that will be used to compute the numerical metric. For example, if the $\leq_{rel\theta}^{abs\theta}$ ordered relation is used, two numerical values are needed per edge.

On the other hand, in order to have a fast computation of the structural metrics based on the pattern $\mathbf{A}$ and to have an in-place algorithm, $\mathbf{A}$ is represented by its quotient graph.

4.6.2 Initialization of C

We mentioned in Section 4.3.1 that $\mathbf{C}$ must contain numerically significant information and have a reasonable size. In practice, the number of entries in $\mathbf{C}$ set smaller than $\alpha \times n$ and $\beta \times nnz(\mathbf{A})$ where $\alpha \geq 1$, $\beta \leq 1$ are positive numbers. β prevents us from having too many entries in $\mathbf{C}$. In practice, $\alpha \in [2,3]$ and $\beta \leq 0.25$ (see Section 4.7.1.3 for more details).

To satisfy the above constraints, a threshold-based dropping is done on a scaled matrix $\mathbf{A}$. If after dropping entries in Algorithm 4.3.1, $\mathbf{C}$ does not satisfy memory constraints, further dropping is done (see Section 4.3.1).

The dropping strategy can be static (referred to as **STCDROP**) or dynamic (referred to as **DYNDROP**). For **STCDROP**, the $NumThresh$ value does not depend on the values of the entries in the matrix. In practice it is set between 0.9 and 0.99. For **DYNDROP**, the $NumThresh$ value is set with respect to the distribution of the values of $\mathbf{A}$. A minimum dropping value ($mindrop$) is set to guarantee the relevance of the numerical information and $NumThresh$ ($\geq mindrop$) is determined to satisfy the memory constraints. The

$[0,1]$ interval is partitioned into subintervals and the count of a subinterval is incremented whenever an entry is in this subinterval. Then the threshold is computed according to the counts of the subintervals so that the number of retained entries satisfies the memory constraints. Thus the complexity of this approach is $\mathcal{O}(nnz(\mathbf{A}))$. It may happen even with the **DYNDROP** strategy that further dropping is required in some pathological cases where many entries are equal to the maximum entry. In practice, $mindrop$ is set between 0.1 and 0.99.

4.6.3 Pivot selection

At each step, the pivot with minimal numerical and/or structural metrics is selected. As is often the case in sparse matrix factorization we want to preserve the sparsity of the factors while controlling the growth in the factors. Numerical thresholds are introduced to give freedom for the pivot selection to balance numerical precision with sparsity preservation. To reduce the complexity of the algorithm, it is also common to limit the pivot search to a set of candidate pivots. For example [27] visits the entries of a fixed number of columns using Zlatev-style search [96]. We use a slightly different algorithm to limit our searches. At each step of the ordering, we look for the best entry, $p = (rowp, colp)$, with respect to an ordered relation within a subset, say S, of the entries in the bipartite graph G_{C^k} (see Section 4.5). Let $m_0(i,j)$ be a structural metric. The subset S is controlled by two threshold parameters $MS > 0$ and $NCOL \geq 0$ used as follows. The smallest MS entries with respect to the structural metric m_0 are first added to S together with the nonzeros of the first $NCOL$ columns encountered while adding the MS entries to S. The set S is thus composed of a first set of MS entries, the MS-set, and a second set, $NCOL$-set $= S \setminus MS$-set.

The choice of an ordered relation implies the underlying algorithmic strategy and data structures that will be described in full detail in Section A.2. We say that we use **structural strategies** in our algorithms when entries are selected with respect to only structural metrics whereas **hybrid strategies** correspond to a combination of structural and numerical metrics. The ordered relations $\leq_{th}$ of definition 5.3 and $\leq_{relθ}^{absθ}$ of definition 5.4, will be used to introduce **one** or **two thresholds** in the hybrid metrics respectively.

For each entry $(i,j) \in \mathbf{C}^k$ we define $v_1(i,j) = c_{ij}/||c_{.j}||_\infty$. A pivot in position (i,j) is said to be *numerically acceptable* (or acceptable) according to $\leq_{th}$ if and only if $|c_{ij}| \geq th \times ||c_{.j}||_\infty$ where $th \in [0,1]$. A pivot in position (i,j) is said to be *numerically acceptable* according to $\leq_{th}^{absθ}$ if and only if $|c_{ij}| \geq th \times ||c_{.j}||_\infty$ and $|c_{ij}| \geq absθ$ where $th \in [0,1]$ and $absθ \geq 0$. If there is no ambiguity we will avoid the ordered relation when we speak about acceptable pivots.

Let $th \in [0,1]$, equation (4.6.1) characterizes the *hybrid strategy with one threshold*.

$$p = arg \quad \min_{\leq_{th}}(i,j,v_1(i,j)) \\ (i,j) \in S \tag{4.6.1}$$

This comes from the definitions given in the following three remarks that will be used to implement the algorithm in Section A.2.

REMARK 6.1. *If the MS-set is processed in order with respect to metric m_0 then the first numerically acceptable entry is the minimum with respect to the ordered relation $\leq_{th}$ on the complete set S.*

REMARK 6.2. *If none of the values in the MS-set entries is numerically acceptable and $NCOL > 0$ then we are sure that at least one entry per column present in S will be numerically acceptable since $th \leq 1$. In this case, however, all the entries of S need to be considered to obtain the minimum on S with respect to $\leq_{th}$.*

REMARK 6.3. *If none of the values in the MS-set entries is numerically acceptable and $NCOL = 0$ then the first entry of the MS-set is the solution of equation (4.6.1).*

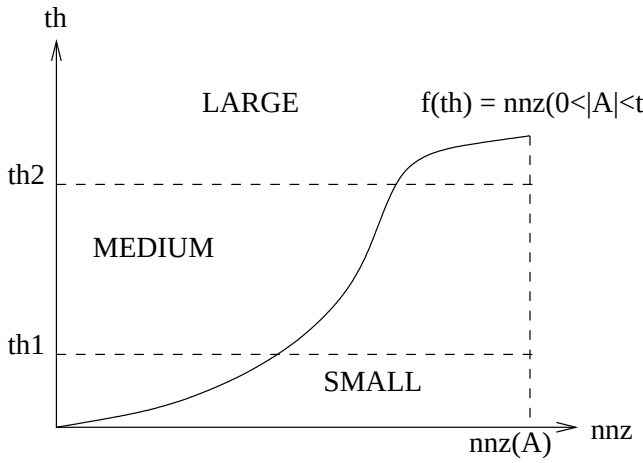


Figure 4.6.1: Example of profile zones of a matrix.

The hybrid selection with one threshold might not be suitable for situations where, for example, $\mathbf{C}$ is composed of a set of relatively large entries and a structural matching which contains relatively small entries ($\mathbf{C}$ must be structurally nonsingular). It will be the case when we do not use `MC64`. Entries in $\mathbf{C}$ can then be spread over the three zones ($SMALL$, $MEDIUM$ and $LARGE$) as shown in Figure 4.6.1. The problem with equation (4.6.1) comes from the fact that if a column of $\mathbf{C}$ holds only $SMALL$ entries then at least one is numerically acceptable according to $\leq_{th}$. This $SMALL$ entry might come from the initial matrix or from a cascade effect. During the ordering, updates of the type $LARGE \times SMALL/LARGE = SMALL$ might occur whereas updates of the type $LARGE \times MEDIUM/LARGE = MEDIUM$ are not performed because $MEDIUM$ entries have been dropped. Therefore the selection of a pivot according to a relative criterion on $\mathbf{C}$ can select a $SMALL$ entry because updates involving $MEDIUM$ dropped entries have not been done. With an absolute threshold $abs\theta$, we want to prevent our pivot selection algorithm from considering as numerically good a $SMALL$ entry. That is why a second absolute threshold $abs\theta$, proportional to the largest entry in $\mathbf{C}^k$ is introduced. The equation (4.6.2) defines the *hybrid strategy with two thresholds*.

$$p = arg \quad \min_{\leq_{th}^{abs\theta}}(i, j, v_1, c_{i,j}) \quad (i, j) \in S \tag{4.6.2}$$

Remark 6.1 still holds with the ordered relation $\leq_{th}^{abs\theta}$ and the next two remarks will be used to implement the selection algorithm (see Section A.2).

REMARK 6.4. *If none of the entries in the MS-set is numerically acceptable according to $\leq_{th}^{abs\theta}$ and if there exists one acceptable entry then all the entries in the columns of S need to be considered to obtain the minimum with respect to $\leq_{th}^{abs\theta}$.*

REMARK 6.5. *If none of the values in S is numerically acceptable then a solution of equation (4.6.1) is also a solution of equation (4.6.2).*

We will show in Appendix A that the global minimum can be obtained for *structural strategies* in $O(1)$ time. Concerning *hybrid strategies*, to bound the complexity of the algorithm, the search is limited to the subset S of the $\mathbf{C}$ entries.

4.6.4 Update of $\mathbf{C}$

As described before, a bipartite graph (and not a bipartite quotient graph) is used to represent $\mathbf{C}$. Therefore, if at each step k, new entries in $\mathbf{C}^{k+1}$ corresponding to fill-ins are added to $G_{C^{k+1}}$ then extra memory might be needed. Indeed, the symbolic factorization using a bipartite graph does not lead to an in-place algorithm. In our case however, $G_{C^{k+1}}$ holds the set of candidate pivots for step $k+1$ and during the update of $G_{C^{k+1}}$ we only need to guarantee that the nonsingular Property 4.3.1 and the inclusion Property 4.3.2 hold.

Algorithm 4.6.4 Updating $\mathbf{C}$ (`UpdateStrategy, NumericalStrategy`)

Let $(rowp, colp)$ be the pivot selected at step k, $U_p = R^C_{rowp}$ and $L_p = C^C_{colp}$
if ((`UpdateStrategy` == MATCHUPDATE) **or** (not enough memory)) **then**
  perform MATCHUPDATE and set $\mathcal{S} = \{(i,j) \in \mathbf{C}^{k+1} \cap (U_p \times L_p)\}$.
else
  perform TOTALUPDATE and set $\mathcal{S} = U_p \times L_p$.
end if
if (`NumericalStrategy`==*hybrid strategy*) **then**
  for all $(i,j) \in \mathcal{S}$ **do**
    $c_{ij}^{(k+1)} = c_{ij}^{(k)} - \frac{c_{i\ colp}^{(k)} c_{rowp\ j}^{(k)}}{c_{rowp\ colp}^{(k)}}$
  end for
end if

These properties can be kept with an in-place implementation. Indeed to maintain Property 4.3.1, we can perform the MATCHUPDATE strategy proposed in Section 4.3.3. At most one entry needs to be added and at least one entry (the pivot) is removed. Let $p = (rowp, colp)$ be the current pivot. Let $(rowp, match\_col)$ and $(match\_row, colp)$ be the matched entries of $\mathbf{C}$ in row $rowp$ and column $colp$, respectively. That is, $(rowp, match\_col) \in \mathcal{M}$ and $(match\_row, colp) \in \mathcal{M}$. If these entries are the same (*i.e.*, $(rowp, colp)$ is a matched entry), nothing needs to be done to maintain Property 4.3.1. Otherwise entry $(match\_row, match\_col)$ can be added to maintain Property 4.3.1. Moreover, this entry corresponds to an entry in $\mathcal{P}attern(\bar{C}^k)$, so that Property 4.3.2 is also maintained. The implementation of this MATCHUPDATE strategy will be fully described by Algorithm 1.4.3 from Section A.4.

As mentioned in Section 4.3.3, another possibility is to try to perform a complete update of the $\mathbf{C}$ matrix. Since this TOTALUPDATE strategy is not in place, its activation as shown in Algorithm 4.6.4 will depend on the memory available. Finally, when a *hybrid strategy* is used we also want to update/compute the numerical values of the entries in the reduced matrix $\mathbf{C}^{k+1}$. Algorithm 4.6.4 summarizes all the possibilities offered for the update of $\mathbf{C}$.

4.6.5 Update of the quotient graph

Algorithm 4.6.5 Update of the quotient graph $\mathcal{G}^k$

Let $p = (rowp, colp)$ be the current pivot at step k and $\mathcal{F}_p = \mathcal{R}_{rowp} \cup \mathcal{C}_{colp}$.
if $\mathcal{U}_p \neq \emptyset$ and $\mathcal{L}_p \neq \emptyset$ **then**
  for each row $i \in \mathcal{L}_p$ **do**
    $\mathcal{A}_{i*} = (\mathcal{A}_{i*} \setminus \mathcal{U}_p) \setminus colp$ /* **variable elimination in row direction** */
    $\mathcal{R}_i = (\mathcal{R}_i \setminus \mathcal{F}_p) \cup p$
  end for
  for each column $j \in \mathcal{U}_p$ **do**
    $\mathcal{A}_{*j} = (\mathcal{A}_{*j} \setminus \mathcal{L}_p) \setminus rowp$ /* **variable elimination in column direction** */
    $\mathcal{C}_j = (\mathcal{C}_j \setminus \mathcal{F}_p) \cup p$
  end for
  compute $|\mathcal{U}_e \setminus \mathcal{U}_p|$, $|\mathcal{L}_e \setminus \mathcal{L}_p|$ for each element e and optionally do aggressive absorption.
else
  /* delete all that is related to p, at most one of the 2 loops below is empty */
  for each row $i \in \mathcal{L}_p$ **do**
    $\mathcal{R}_i = (\mathcal{R}_i \setminus \mathcal{F}_p)$
    $\mathcal{A}_{i*} = \mathcal{A}_{i*} \setminus colp$
  end for
  for each column $j \in \mathcal{U}_p$ **do**
    $\mathcal{C}_j = (\mathcal{C}_j \setminus \mathcal{F}_p)$
    $\mathcal{A}_{*j} = \mathcal{A}_{*j} \setminus rowp$
  end for
  set $|\mathcal{U}_e \setminus \mathcal{U}_p| = |\mathcal{U}_e|$, $|\mathcal{L}_e \setminus \mathcal{L}_p| = |\mathcal{L}_e|$ for each element e
end if

Algorithm 4.6.5 describes how the quotient graph structure of the original matrix is updated. It illustrates an important difference with the DMLS algorithm relative to the structural metric computation and pruning of the bipartite quotient graph. This difference leads to a modified behaviour with respect to irreducibility detection. We explain and illustrate these aspects in the following paragraph and will further discuss the full details and the consequences for the computation of the structural metrics (degree and fill-in estimations) in Section 4.6.6.

When the pivot choice is limited to the diagonal of the permuted matrix (as in the DMLS algorithm) it is possible to anticipate where local symmetrization will occur. For each row i in $\mathcal{L}_p$, all the variables belonging to $\mathcal{L}_p$ can then be removed from $\mathcal{A}_{*i}$ even when $i \notin \mathcal{U}_p$. This is illustrated in Figure 4.6.2 where shaded areas correspond to variables which can be removed from the variable adjacency lists because they are implicitly stored through the element p. All of this cleaning cannot be performed during the CMLS algorithm. For example, if in Figure 4.6.2 the next pivot selected is (j, i) ($i \in \mathcal{L}_p$,

$j \notin \mathcal{L}_p$ and $i \notin \mathcal{U}_p$), then the element p is not needed (there is no local symmetrization) to build the row and column adjacency of (j,i). Therefore, the column structure of the element p should not be used to store entries in the column structure of variables in column i. Similarly, as shown in Figure 4.6.2, with the DMLS algorithm we can also remove from $\mathcal{A}_{j*}$ all variables belonging to $\mathcal{U}_p$ for each column j in $\mathcal{U}_p$. In other words, when the pivot choice is limited to the diagonal (as in the DMLS algorithm) we know where the local symmetrization will be applied so that we can prune both the row and the column adjacency of a variable belonging to $\mathcal{U}_p$ **or** $\mathcal{L}_p$. In the following, we will say that the DMLS algorithm performs elimination in both row and column directions that will be referred to as **both way variable elimination**.

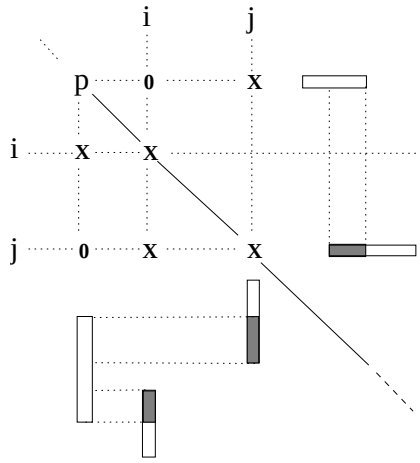


Figure 4.6.2: Illustration of both way variable elimination.

On the other hand, CMLS does only **variable elimination in one direction** because it does not know where the local symmetrization will be done. Thus in Algorithm 4.6.5 the variable elimination in the row direction and the variable elimination in the column direction are done in two different loops. If variables are removed from row i and column i, this means that $i \in \mathcal{L}_p$ **and** $i \in \mathcal{U}_p$.

PROPERTY 6.1. *If the variable elimination is done in both row and column directions then the current pivot, say p, can safely be removed from the quotient graph only when $\mathcal{U}_p = \emptyset$ and $\mathcal{L}_p = \emptyset$.*

This property results from two observations. Firstly, p can be removed from the quotient graph when $\mathcal{U}_p = \emptyset$ and $\mathcal{L}_p = \emptyset$. Secondly, let us suppose without loss of generality that $\mathcal{U}_p = \emptyset$ and $\mathcal{L}_p \neq \emptyset$. In this case, p cannot be removed from the quotient graph because there may exist a variable $i \in \mathcal{L}_p$ for which entries in $\mathcal{A}_{*i}^{(0)}$ have been pruned because of local symmetrization with respect to p. This is illustrated in Figure 4.6.3 where the dark area (1) in column i is first stored through element e then stored through element p (after absorption of element e by pivot p). The dashed area (2) moves from $\mathcal{A}_{*i}$ to $\mathcal{L}_p$. Property 6.1 thus illustrates a drawback of both way variable elimination. Local symmetrization has been anticipated and we cannot eliminate elements because they may hold information required to describe the adjacency structure of uneliminated variables. For example, if the DMLS algorithm is applied on a triangular matrix, it will

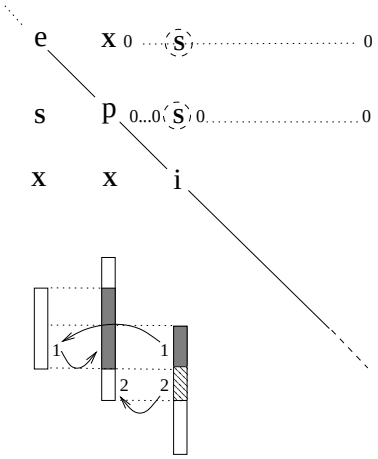


Figure 4.6.3: Effect of variable elimination in both directions on reducibility detection (S indicates the positions of local symmetrization).

create artificial dependency and fill-in in the factors because of the anticipation of the local symmetrization. Thus the triangular form will be perturbed.

One could suggest postponing the pruning of $\mathcal{A}_{*i}$ (with entries in $\mathcal{L}_p$) until knowing whether $\mathcal{U}_p = \emptyset$ or not. Even in this case, when $\mathcal{U}_p$ is detected as empty, it is in general impossible to know whether $\mathcal{L}_p$ does not already include the absorption of the column entries coming from another coupled element adjacent to p and therefore it is impossible to remove p from the quotient graph. This is illustrated in Figure 4.6.3 where it is shown that the dark area of $\mathcal{L}_p$ also carries part of the contribution of e that contains the column structure of i. This is so even if the column structure of i does not depend on the column structure of any element adjacent to p since $\mathcal{U}_p = \emptyset$. This suggests the following property which is a direct consequence of Property 6.1.

PROPERTY 6.2. *If the variable elimination is done in both row and column directions then the current pivot, say p, can be removed from the quotient graph (from $\mathcal{V}_r$, $\mathcal{V}_c$, and $\mathcal{E}$) before being eliminated only when ($\mathcal{U}_p = \emptyset$ and $\mathcal{L}_p = \emptyset$) or when ($\mathcal{U}_p = \emptyset$ and $\mathcal{R}_p = \emptyset$ and $\mathcal{C}_p = \emptyset$) or when ($\mathcal{L}_p = \emptyset$ and $\mathcal{R}_p = \emptyset$ and $\mathcal{C}_p = \emptyset$).*

Property 6.2 can be used to design a preprocessing of the `DMLS` algorithm that will detect the triangular form of a matrix having nonzeros on the diagonal. This very simple preprocessing was added to the `DMLS` code and is used in all our experiments.

From Property 6.1 we see that using variable elimination in only one direction should improve the behaviour of the algorithm on reducible matrices.

PROPERTY 6.3. *If the variable elimination is done in one direction only then the current pivot, say p, can be removed from the quotient graph when $\mathcal{U}_p = \emptyset$ or $\mathcal{L}_p = \emptyset$.*

Property 6.3 comes from the fact that the local symmetrization is not, and cannot be, forecast. That is why when $\mathcal{U}_p = \emptyset$ or $\mathcal{L}_p = \emptyset$, the element p no longer contains useful information. From Property 6.3 we see that `CMLS` will naturally detect a triangular matrix. In the rest of this section, we discuss the extension of the property to the detection and/or the exploitation of the BTF (Block Triangular Form). We show that, under some

assumptions, the CMLS algorithm can exploit and detect the BTF which is in general not the case with the DMLS algorithm.

DEFINITION 6.1. *We say that an ordering is* CMLS **compatible** *with the BTF when it firstly selects only pivots within the diagonal blocks of the BTF; secondly it eliminates all pivots in a block before processing another block; thirdly the order in which the BTF block are considered must be such that each block is not adjacent to any remaining block in either the row or the column direction.*

The third condition of the previous definition allows a block to be considered in the order of the BTF but also allows some slight variation of this form as will be shown in the example of Figure 4.6.5.

PROPERTY 6.4. *If* CMLS *selects pivots with a* CMLS *compatible ordering with a BTF then the sparsity resulting from the block triangular form will be fully preserved and* CMLS *will generate a forest, where each tree corresponds to an irreducible component of the BTF.*

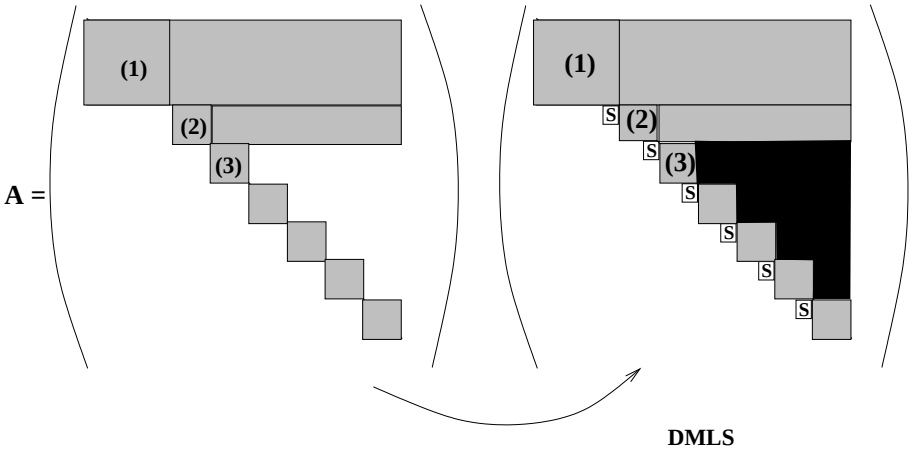


Figure 4.6.4: The initial matrix is in BTF (CMLS does not introduce any fi ll-in outside the diagonal blocks if this ordering is used).

In the following three examples (corresponding to three matrices that are reducible), we assume that the ordering provided on the left hand-side on the figures is used by both the CMLS and DMLS algorithms. We then comment on the difference between CMLS and DMLS and illustrate that when Property 6.4 is respected the BTF is fully exploited by CMLS.

In Figure 4.6.4, the matrix is in BTF and we show on the left-hand side the original matrix in BTF where the gray areas represent the initial full blocks. The structure resulting from the application of DMLS is provided on the right-hand side. The fill-in is represented by dense black blocks and **S** indicates entries resulting from local symmetrization. We see that, with DMLS, local symmetrization interconnects all blocks of the BTF (we lose the BTF forest) and that by creating a link between block (2) and block (3) through the first line of block (3) it completely fills-up the first line of block (3) which in turn will create fill-in in all lines of block (3) and in all lines of other blocks. With CMLS the last pivot of

block (1) will be removed. We do not introduce local symmetrization because its column structure is empty so that Property 6.3 can be applied. The same property holds for each of the following blocks so that CMLS does not introduce any fill-in and produces a forest of trees associated with each diagonal block.

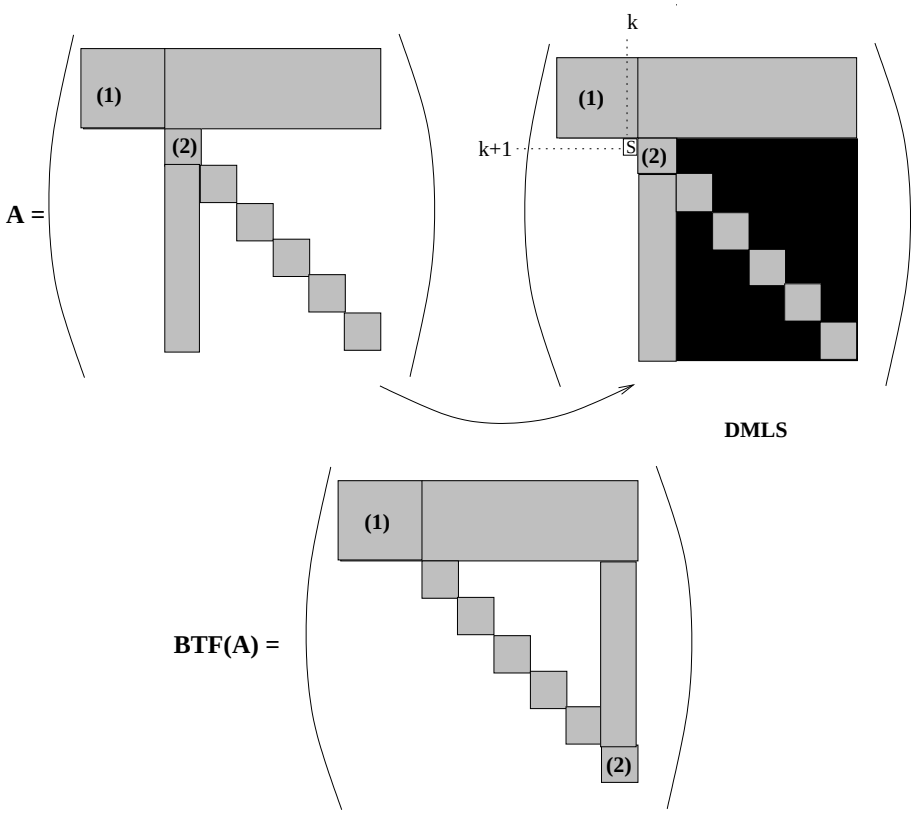


Figure 4.6.5: The initial matrix is not in BTF, but the initial ordering is CMLS compatible (CMLS does not introduce any fi ll-in outside the diagonal blocks if this ordering is used).

We now consider a matrix in Figure 4.6.5 that is not in BTF. The pivot order is however CMLS compatible with the BTF since even if blocks are not taken in the order of the BTF the third condition of Definition 6.1 holds. In this example, we see that Property 6.3 can be applied since, at the end of the processing of each block, either $\mathcal{U}_p$ or $\mathcal{L}_p$ will be empty (this results from using the third condition of Definition 6.1). CMLS will fully exploit and preserve the BTF of the matrix. The last pivot selected in each block will become the root of the tree associated with the block of the BTF to which it belongs. We see however that, with DMLS, the s entry in position $(k+1, k)$ will completely fill row $k+1$ and that after elimination of the $(k+1)^{st}$ diagonal entry the remaining submatrix is full.

As is illustrated in Figure 4.6.6 when the order of the blocks does not respect the third condition of Definition 6.1, CMLS partially loses the BTF of the matrix. In our example, local symmetrization between blocks (1) and (2) will create a dependency between them and fills one rectangular block because of the propagation of this dependency. Block (3) will however remain an independent subtree because the last pivot, say p, in block (2)

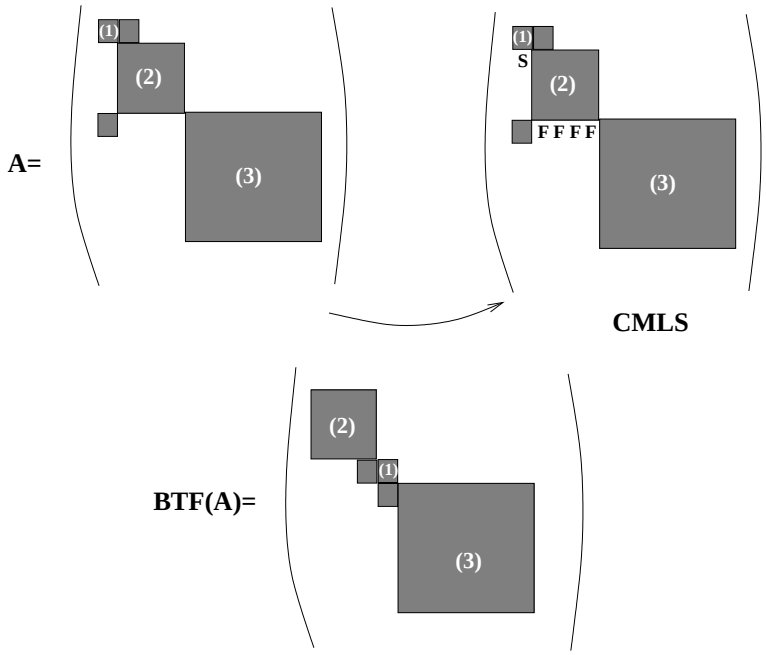


Figure 4.6.6: The initial matrix is not in BTF and the ordering is not CMLS compatible

will have $\mathcal{U}_p = \emptyset$ and p will become the root of the subtree containing blocks (1) and (2).

4.6.6 Update of the structural metric

In this section, we describe two classes of local heuristics to estimate the structural quality of a pivot. Each class of heuristics is characterized by a metric that is locally minimized at each step of the elimination. In the preamble section, we first describe the common framework (independent of the metric used) and introduce a generic algorithm describing how to compute the approximate external row and column degrees that are needed to compute all metrics. In Section 4.6.6.2, the Markowitz cost, a metric related to the number of operations involved during elimination, is briefly introduced. We then describe two metrics based on an upper bound of the fill-in involved at each step of the elimination. In Section 4.6.6.3 a basic approximation of the fill-in is introduced. The resulting heuristic will be referred to as AMF. In Section 4.6.6.4, a more accurate approximation of the fill-in is proposed. This approximation of the fill-in has been observed by the authors of AMD [4] in the context of symmetric matrices. We provide, in Section 4.6.6.4, a generalization of this approximation to unsymmetric matrices and prove that this approximation is a tighter upper bound of the fill-in metric than the approximations proposed for symmetric matrices in [92]. Note that concerning the deficiency approximation in [85], there is no guaranty that it will be an upper bound of the fill-in. We also describe how to compute this approximation efficiently in the context of our CMLS algorithm. The approximate minimum fill-in heuristic based on the new approximation will be referred to as AMFI.

In Section 4.6.6.5, we finally discuss the complexity of the algorithms used to update the structural metrics.

4.6.6.1 Preamble

Let us suppose that the p^{th} pivot $(rowp, colp)$ has been selected. All the entries in $(\mathcal{L}_p \times V_c \cup V_r \times \mathcal{U}_p) \cap \mathcal{P}attern(C_{p+1})$ (at the intersection between the shaded areas of Figure 4.6.7 and the matrix $\mathbf{C}$ at step $p+1$) are involved in the structural metric updates since the structure of rows in $\mathcal{L}_p$ and columns in $\mathcal{U}_p$ might have changed. The size of this area is thus larger than the area involved in the update of the structure of $\mathbf{C}$ (see Section 4.6.4) since columns (resp. rows) of $\mathcal{U}_p \setminus R^C_{rowp}$ (resp. of $\mathcal{L}_p \setminus C^C_{colp}$) need also be considered. The algorithm to update the structural metrics will thus be one of the most costly steps of our algorithm. This will be further commented on in the implementation appendix.

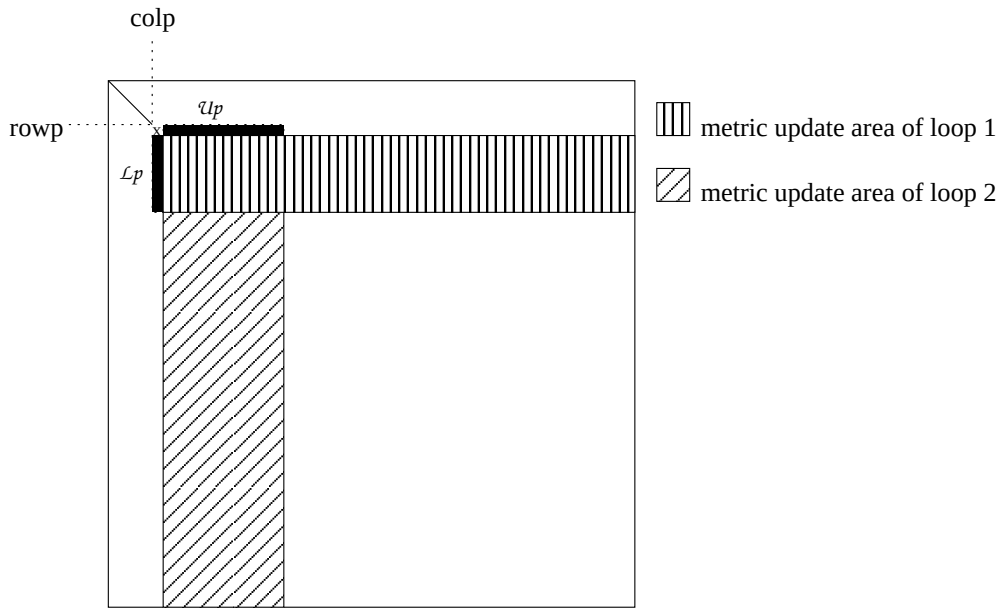


Figure 4.6.7: Area involved in the structural metric updates. $\mathcal{U}_p$ and $\mathcal{L}_p$ represent the row and column structure of the pivot in $\mathbf{A}$ and the metric update concern the entries of $\mathbf{C}$ that intersect with the striped areas.

As already explained in Section 4.5.1.2, we want the metrics to reflect the quality of an entry if it were selected as the next pivot. That is why we compute metrics which are related to the structure of our quotient graph on which local symmetrization has been applied at each step of the elimination. For the sake of clarity, in the following we will omit to say that degrees, approximate degrees, fill-ins and approximate fill-ins are related to this quotient graph structure. Note that, since on symmetric matrices our quotient graph becomes the standard symmetric quotient graph, all the properties demonstrated in this section for unsymmetric matrices naturally apply to symmetric matrices.

Equations (4.5.1) and (4.5.2) that include local symmetrization could be used to compute the exact external degrees, but it would be costly. Instead, similarly to `AMD` and `DMLS`

algorithms, approximate row and columns external degrees can be computed. The AMD like approximate external row and columns degree, $amd_r(i,j)$ and $amd_c(i,j)$ respectively, are then defined by the following two equations:

$$amd_r(i,j) = |\mathcal{A}_{i*} \setminus \mathcal{U}_p| + |\mathcal{U}_p \setminus j| + \sum_{e \in \mathcal{R}_i \setminus \mathcal{C}_j} (|\mathcal{U}_e \setminus \mathcal{U}_p|) + \sum_{e \in \mathcal{C}_j} (|\mathcal{U}_e \setminus \mathcal{U}_p|) - \alpha_j, \quad \text{with } \alpha_j = \max(|\mathcal{C}_j|, 1) \text{ if } j \notin \mathcal{U}_p \text{ else } \alpha_j = 0. \tag{4.6.3}$$

$$amd_c(i,j) = |\mathcal{A}_{*j} \setminus \mathcal{L}_p| + |\mathcal{L}_p \setminus i| + \sum_{e \in \mathcal{R}_i} (|\mathcal{L}_e \setminus \mathcal{L}_p|) + \sum_{e \in \mathcal{C}_j \setminus \mathcal{R}_i} (|\mathcal{L}_e \setminus \mathcal{L}_p|) - \beta_i, \quad \text{with } \beta_i = \max(|\mathcal{R}_i|, 1) \text{ if } i \notin \mathcal{L}_p \text{ else } \beta_i = 0. \tag{4.6.4}$$

As observed in the DMLS algorithm, degree corrections (α_j and β_i in equations (4.6.3) and (4.6.4)) are introduced to improve the approximations of the row and column external degrees. To justify these correction terms, one can observe that if $j \notin \mathcal{U}_p$ then j is counted in each $\mathcal{U}_e \setminus \mathcal{U}_p$ such that e is adjacent to column j ($e \in \mathcal{C}_j$). Furthermore, if $\mathcal{C}_j$ is empty and $j \notin \mathcal{U}_p$ then column j has been counted in $\mathcal{A}_{i*} \setminus \mathcal{U}_p$ and should then be subtracted. This explains the use of α_j in the correction, β_i can be justified in a similar way. The correction terms α_j and β_i differ slightly from the DMLS correction terms because DMLS removes diagonal entries from the quotient graph.

The $|\mathcal{U}_e \setminus \mathcal{U}_p|$ and $|\mathcal{L}_e \setminus \mathcal{L}_p|$ quantities are computed similarly to the AMD and DMLS algorithms and will also be used to compute areas that can be subtracted during the approximate minimum fill algorithm (see Section 4.6.6.4).

Moreover, since variable elimination is done in only one direction, the computation of the metric is less accurate with CMLS than with DMLS. With DMLS, for any element e, row index $i \in \mathcal{U}_p$ and column index $j \in \mathcal{L}_p$, we have $\mathcal{A}_{i*} \cap \mathcal{U}_e = \emptyset$ and $\mathcal{A}_{*j} \cap \mathcal{L}_e = \emptyset$. It is no longer the case with CMLS but was exploited in Section 4.6.5 to help CMLS discover the BTF form of the matrix.

Algorithm 4.6.6 describes the main components of a generic algorithm to compute the approximate external row and column degrees that will be used to update the structural metrics. In *Loop 1*, the metric of the entries in the horizontal rectangle of Figure 4.6.7 ($\mathcal{L}_p \times V_c \ \cap \ \mathcal{P}attern(C)$) is updated. In *Loop 2* only the metric of the entries in the vertical rectangle of Figure 4.6.7 that are not already visited is then updated.

4.6.6.2 Approximation of Markowitz cost

The approximation of the Markowitz cost comes directly from our approximation of the external row and column degrees. After eliminating the k^{th} pivot, we define

$$\overline{amd}_r(i,j) = \min(amd_r(i,j), n-k-1), \tag{4.6.5}$$

and

$$\overline{amd}_c(i,j) = \min(amd_c(i,j), n-k-1). \tag{4.6.6}$$

Algorithm 4.6.6 Update of the structural metric.

for each row $i \in \mathcal{L}_p$ **do** /* *Loop 1:* */
1 $rowdeg\_save = |\mathcal{A}_{i*} \setminus \mathcal{U}_p| + \sum_{e \in \mathcal{R}_i} |\mathcal{U}_e \setminus \mathcal{U}_p|$
2 $coldeg\_save = \sum_{e \in \mathcal{R}_i} |\mathcal{L}_e \setminus \mathcal{L}_p|$
for each column j such that $(i,j) \in \mathbf{C}$ **do**
3 $amd_r(i,j) = rowdeg\_save + \sum_{e \in \mathcal{C}_j \setminus \mathcal{R}_i} |\mathcal{U}_e \setminus \mathcal{U}_p| + |\mathcal{U}_p \setminus \{j\}| - \alpha_j$
4 $amd_c(i,j) = coldeg\_save + |\mathcal{A}_{*j} \setminus \mathcal{L}_p| + \sum_{e \in \mathcal{C}_j \setminus \mathcal{R}_i} |\mathcal{L}_e \setminus \mathcal{L}_p| + |\mathcal{L}_p \setminus \{i\}|$
5 $metric(i,j) = f(amd_r(i,j), amd_c(i,j))$
end for
end for
for each column $j \in \mathcal{U}_p$ **do** /* *Loop 2:* */
6 $rowdeg\_save = \sum_{e \in \mathcal{C}_j} |\mathcal{U}_e \setminus \mathcal{U}_p|$
7 $coldeg\_save = |\mathcal{A}_{*j} \setminus \mathcal{L}_p| + \sum_{e \in \mathcal{C}_j} |\mathcal{L}_e \setminus \mathcal{L}_p|$
for each row i such that $(i,j) \in \mathbf{C}$ and $i \notin \mathcal{L}_p$ **do**
8 $amd_r(i,j) = rowdeg\_save + |\mathcal{A}_{i*} \setminus \mathcal{U}_p| + \sum_{e \in \mathcal{R}_i \setminus \mathcal{C}_j} |\mathcal{U}_e \setminus \mathcal{U}_p| + |\mathcal{U}_p \setminus \{j\}|$
9 $amd_c(i,j) = coldeg\_save + \sum_{e \in \mathcal{R}_i \setminus \mathcal{C}_j} |\mathcal{L}_e \setminus \mathcal{L}_p| + |\mathcal{L}_p| - \beta_i$
10 $metric(i,j) = f(amd_c(i,j), amd_r(i,j))$
end for
end for

The metric associated with the approximate Markowitz cost is then defined as

$$metric^{(k+1)}(i,j) = \min \left\{ \begin{array}{ll} \overline{amd}_r(i,j) \times \overline{amd}_c(i,j) & \\ metric^{(k)}(i,j) & +|\mathcal{U}_p \setminus j| \times \overline{amd}_c(i,j) \\ & +|\mathcal{L}_p \setminus i| \times \overline{amd}_r(i,j) \\ & -|\mathcal{U}_p \setminus j| \times |\mathcal{L}_p \setminus i|. \end{array} \right. \tag{4.6.7}$$

We use here the convention that if (i,j) is a new entry in $\mathbf{C}$, then $metric^{(k)}(i,j)$ in equation (4.6.7) is set to $+\infty$. Note that during the update of the degrees, contrary to DMLS, we do not use the values of approximate row and column degrees computed at the previous step. This could have been done but would have required us to store two other arrays (one for row degrees, one for column degrees) of size $|\mathbf{C}|$.

4.6.6.3 Basic approximation of the fi ll-in

The approximation for the Markowitz cost computes an upper bound of the area of the Schur complement block. With a minimum fill-in based metric, we want to estimate the new fill-in that would occur in the reduced matrix if an entry were selected as the next pivot. For the sake of completeness, one should mention that the fill-in metric of variables at distance two from the pivot might also decrease. In this work, we will only consider variables adjacent to the pivot, $i.e.$ at distance one, since results on symmetric matrices have shown that considering distance two variables significantly increases the complexity of the algorithm for relatively little gain in the quality of the ordering [85, 92].

A first upper bound of the fill-in that would occur can be obtained by removing the area corresponding to $\mathcal{L}_p \times \mathcal{U}_p$ from the Markowitz cost. This metric will be referred to as

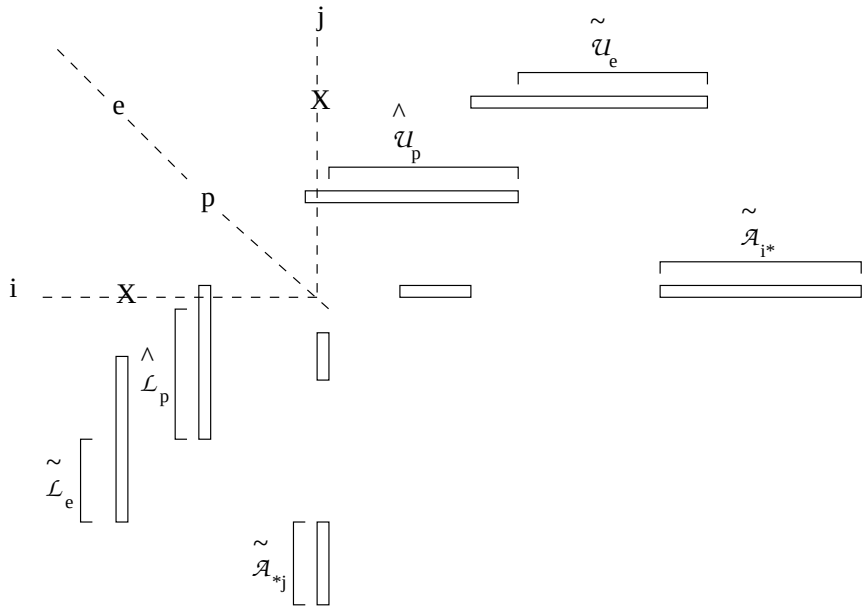


Figure 4.6.8: $\tilde{\mathcal{L}_e}$, $\tilde{\mathcal{U}_e}$, $\tilde{\mathcal{A}_{*j}}$, $\tilde{\mathcal{A}_{i*}}$, $\hat{\mathcal{L}_p}$ and $\hat{\mathcal{U}_p}$ quantities.

AMF in the remainder of this document and is computed as follows:

$$metric^{(k+1)}(i,j) = \min \begin{cases} \overline{amd}_r(i,j) \times \overline{amd}_c(i,j) - |\mathcal{U}_p \setminus j| \times |\mathcal{L}_p \setminus i| \\ metric^{(k)}(i,j) \quad +|\mathcal{U}_p \setminus j| \times \overline{amd}_r(i,j) \\ \qquad\qquad\quad +|\mathcal{L}_p \setminus i| \times \overline{amd}_c(i,j) \\ \qquad\qquad\quad -2 \times |\mathcal{U}_p \setminus j| \times |\mathcal{L}_p \setminus i| \end{cases} \tag{4.6.8}$$

The second term of equation (4.6.8) holds the maximum increase in the metric due to the elimination of pivot p. It should be noted that $|\mathcal{U}_p \setminus j| \times |\mathcal{L}_p \setminus i|$ is removed from $metric^{(k+1)}(i,j)$ in both terms of the minimum.

4.6.6.4 Improved approximation of the fi ll-in

A tighter approximation of the fill-in to the factors can be obtained by removing all the areas already filled during the elimination of the previous elements. The approximation described in this section is a generalization to unsymmetric matrices of an observation made in the context of symmetric matrices but never formally expressed/proved. Finally we explain how to compute this new upper bound using already computed information and local correction terms.

Suppose that $i \in \mathcal{L}_p$ or $j \in \mathcal{U}_p$. Let $\mathcal{F} = \mathcal{R}_i \cup \mathcal{C}_j$. Let e be an element that belongs to $\mathcal{F}$. To simplify the notations we note $\tilde{\mathcal{L}_e} = (\mathcal{L}_e \setminus \mathcal{L}_p) \setminus \{i\}$, $\tilde{\mathcal{U}_e} = (\mathcal{U}_e \setminus \mathcal{U}_p) \setminus \{j\}$, $\tilde{\mathcal{A}_{*j}} = (\mathcal{A}_{*j} \setminus \mathcal{L}_p) \setminus \{i\}$, $\tilde{\mathcal{A}_{i*}} = (\mathcal{A}_{i*} \setminus \mathcal{U}_p) \setminus \{j\}$, $\hat{\mathcal{L}_e} = \mathcal{L}_e \setminus \{i\}$ and $\hat{\mathcal{U}_e} = \mathcal{U}_e \setminus \{j\}$. This quantities are represented on Figure 4.6.8.

Let $d_r(i,j)$ and $d_c(i,j)$ denote the external row and column external degrees of entry

(i,j) respectively. With our notations we have:

$$d_r(i,j) = |(\mathcal{U}_p \cup \mathcal{A}_{i*} \cup \bigcup_{e\in\mathcal{F}} \mathcal{U}_e) \setminus \{j\}| \leq |\hat{\mathcal{U}}_p| + |\tilde{\mathcal{A}}_{i*}| + |\bigcup_{e\in\mathcal{F}} \tilde{\mathcal{U}}_e| \quad (4.6.9)$$

$$d_c(i,j) = |(\mathcal{L}_p \cup \mathcal{A}_{*j} \cup \bigcup_{e\in\mathcal{F}} \mathcal{L}_e) \setminus \{i\}| \leq |\hat{\mathcal{L}}_p| + |\tilde{\mathcal{A}}_{*j}| + |\bigcup_{e\in\mathcal{F}} \tilde{\mathcal{L}}_e|. \quad (4.6.10)$$

Let $S(i,j)$ denote the union of the area associated with all the elements adjacent to entry (i,j):

$$S(i,j) = |\bigcup_{e\in\mathcal{F}} (\hat{\mathcal{L}}_e \times \hat{\mathcal{U}}_e) \setminus (\mathcal{L}_p \times \mathcal{U}_p)|.$$

Ideally one might want to also subtract $S(i,j)$ from the basic AMF metric. This tight upper bound of the fill-in that would occur (including local symmetrization) in our quotient graph if an entry (i,j) were eliminated is:

$$d_r(i,j)d_c(i,j) - |\hat{\mathcal{U}}_p||\hat{\mathcal{L}}_p| - S(i,j).$$

Note that this approximation of the fill-in would be much tighter than any of the approximated upper bounds proposed in [92]. Concerning the deficiency approximation in [85], there is no guaranty that it will be an upper bound of the fill-in.

In [92], the authors have observed that instead of using the exact external degrees one could use the approximate (in the sense of the AMD algorithm) external degrees since both produce results of comparable quality and since AMD based metrics are significantly faster to compute. In this context, the corresponding upper bound of the fill-in metric becomes

$$amd_r(i,j)amd_c(i,j) - |\hat{\mathcal{U}}_p||\hat{\mathcal{L}}_p| - S(i,j). \quad (4.6.11)$$

Let AS be an overestimation of area S,

$$AS(i,j) = \sum_{e\in\mathcal{F}} |(\hat{\mathcal{L}}_e \times \hat{\mathcal{U}}_e) \setminus (\mathcal{L}_p \times \mathcal{U}_p)|. \quad (4.6.12)$$

Note that $AS(i,j)$ should be easier to compute than $S(i,j)$ since we also have :

$$AS(i,j) = \sum_{e\in\mathcal{F}} |\tilde{\mathcal{L}}_e||\hat{\mathcal{U}}_e| + |\hat{\mathcal{L}}_e \cap \mathcal{L}_p||\tilde{\mathcal{U}}_e|. \quad (4.6.13)$$

Using Property 6.5 we will prove that the area $AS(i,j)$ can in fact be subtracted, instead of $S(i,j)$, from equation (4.6.11) to obtain a better upper bound of the fill metric.

An intuitive proof of Property 6.5 is that, during the computation of the degree approximation, the matrix is expanded in such a way that it is as if the intersections between $\tilde{\mathcal{U}}_e$ and $\tilde{\mathcal{L}}_e$ were empty. The area AS corresponds to a real surface in the expanded matrix and can be removed from the area $amd_r(i,j)amd_c(i,j)$ to compute the fill-in that would occur on the expanded matrix (see Figure 4.6.9) if (i,j) were selected as next pivot. Moreover this fill-in on the expanded matrix is an upper bound of the exact fill-in on the quotient graph.

PROPERTY 6.5. *$amd_r(i,j)amd_c(i,j) - |\hat{\mathcal{U}}_p||\hat{\mathcal{L}}_p| - AS(i,j)$ is an upper bound of the fill-in that would occur in the quotient graph if (i,j) were eliminated.*

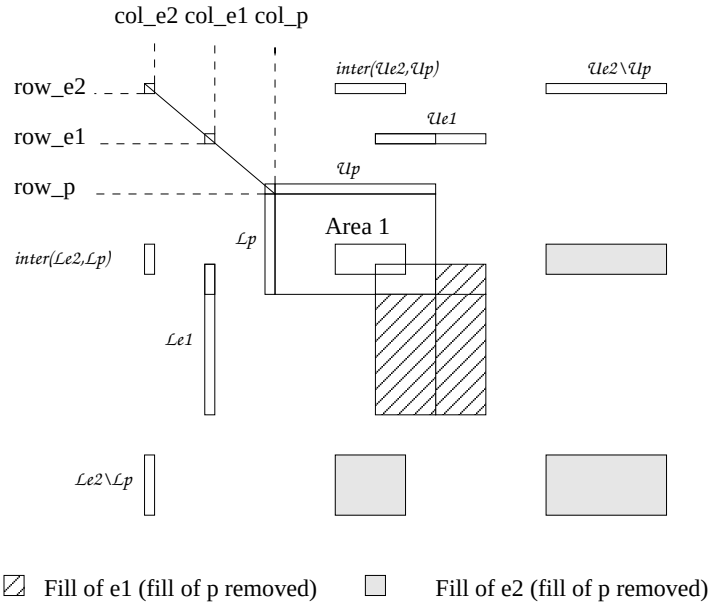


Figure 4.6.9: `AMFI` areas in the expanded matrix.

PROOF. In the following we provide a theoretical proof of Property 6.5. To establish our property we will in fact prove that

$$amd_r(i,j)amd_c(i,j) - |\hat{\mathcal{U}_p}||\hat{\mathcal{L}_p}| - AS(i,j) \geq d_r(i,j)d_c(i,j) - |\hat{\mathcal{U}_p}||\hat{\mathcal{L}_p}| - S(i,j).$$

With our definitions of $amd_r(i,j)$ and $amd_c(i,j)$, we have :

$$amd_r(i,j) \geq |\tilde{\mathcal{A}_{i*}}| + |\hat{\mathcal{U}_p}| + \sum_{e\in\mathcal{F}} |\tilde{\mathcal{U}_e}| \tag{4.6.14}$$

and

$$amd_c(i,j) \geq |\tilde{\mathcal{A}_{*j}}| + |\hat{\mathcal{L}_p}| + \sum_{e\in\mathcal{F}} |\tilde{\mathcal{L}_e}|. \tag{4.6.15}$$

In the above formula, the difference between the left hand side and the right hand side of either inequality is 0 or 1. For equation (4.6.14), the strict inequality corresponds to $\mathcal{C}_j \neq \emptyset$, $j \in \mathcal{A}_{i*}$ and $j \notin \mathcal{U}_p$ since in this case the α_j term of equation (4.6.3) cannot take into account the fact that $j \in \mathcal{A}_{i*}$.

From equations (4.6.14) and (4.6.9) we have

$$d_r(i,j) \leq amd_r(i,j) - \sum_{e\in\mathcal{F}} |\tilde{\mathcal{U}_e} \cap \left(\bigcup_{e'\in\mathcal{F},\, e'>e} \tilde{\mathcal{U}_{e'}} \right)|. \tag{4.6.16}$$

From equations (4.6.15) and (4.6.10) we have

$$d_c(i,j) \leq amd_c(i,j) - \sum_{e\in\mathcal{F}} |\tilde{\mathcal{L}_e} \cap \left(\bigcup_{e'\in\mathcal{F},\, e'>e} \tilde{\mathcal{L}_{e'}} \right)|, \tag{4.6.17}$$

Furthermore, we also have

$$AS = S + \sum_{e\in\mathcal{F}} |(\hat{\mathcal{L}}_e \times \hat{\mathcal{U}}_e) \cap \left(\bigcup_{e'\in\mathcal{F},\, e'>e} (\mathcal{L}_{e'} \times \mathcal{U}_{e'}) \right) \setminus (\mathcal{L}_p \times \mathcal{U}_p)|. \tag{4.6.18}$$

Note that $\hat{}$ has been removed from over the e' terms of equation (4.6.18) because of the intersection with the $\hat{\mathcal{L}}_e \times \hat{\mathcal{U}}_e$ term.

From (4.6.16) and (4.6.17) we have

$$\begin{aligned} d_r(i,j)d_c(i,j) \;\leq\; & amd_r(i,j)amd_c(i,j) \\ & + \overbrace{\left(\sum_{e\in\mathcal{F}} |\tilde{\mathcal{U}}_e \cap \left(\bigcup_{e'\in\mathcal{F},\, e'>e} \tilde{\mathcal{U}}_{e'} \right)|\right)\left(\sum_{e\in\mathcal{F}} |\tilde{\mathcal{L}}_e \cap \left(\bigcup_{e'\in\mathcal{F},\, e'>e} \tilde{\mathcal{L}}_{e'} \right)|\right)}^{T_1} \\ & \overbrace{- amd_c(i,j)\left(\sum_{e\in\mathcal{F}} |\tilde{\mathcal{U}}_e \cap \left(\bigcup_{e'\in\mathcal{F},\, e'>e} \tilde{\mathcal{U}}_{e'} \right)|\right)}^{T_2} \\ & \overbrace{- amd_r(i,j)\left(\sum_{e\in\mathcal{F}} |\tilde{\mathcal{L}}_e \cap \left(\bigcup_{e'\in\mathcal{F},\, e'>e} \tilde{\mathcal{L}}_{e'} \right)|\right)}^{T_3} \end{aligned} \tag{4.6.19}$$

From (4.6.18),

$$\begin{aligned} AS(i,j) - S(i,j) \;=\; & \textstyle\sum_{e\in\mathcal{F}} |(\hat{\mathcal{L}}_e \times \hat{\mathcal{U}}_e) \cap \left(\bigcup_{e'\in\mathcal{F},\, e'>e} (\mathcal{L}_{e'} \times \mathcal{U}_{e'})\right) \setminus (\mathcal{L}_p \times \mathcal{U}_p)| \\ = \; & \textstyle\sum_{e\in\mathcal{F}} |(\tilde{\mathcal{L}}_e \times \hat{\mathcal{U}}_e) \cap \left(\bigcup_{e'\in\mathcal{F},\, e'>e} (\tilde{\mathcal{L}}_{e'} \times \mathcal{U}_{e'})\right)| \\ & + \textstyle\sum_{e\in\mathcal{F}} |(\hat{\mathcal{L}}_p \times \tilde{\mathcal{U}}_e) \cap \left(\bigcup_{e'\in\mathcal{F},\, e'>e} (\mathcal{L}_{e'} \times \tilde{\mathcal{U}}_{e'})\right)| \\ \leq \; & \overbrace{\sum_{e\in\mathcal{F}} |\tilde{\mathcal{L}}_e \cap \left(\bigcup_{e'\in\mathcal{F},\, e'>e} \tilde{\mathcal{L}}_{e'} \right)| \times |\hat{\mathcal{U}}_e|}^{T_4} \\ & \overbrace{+ |\hat{\mathcal{L}}_p| \sum_{e\in\mathcal{F}} |\tilde{\mathcal{U}}_e \cap \left(\bigcup_{e'\in\mathcal{F},\, e'>e} \tilde{\mathcal{U}}_{e'} \right)|}^{T_5}. \end{aligned} \tag{4.6.20}$$

From (4.6.19) and (4.6.20) we have

$$d_r(i,j)d_c(i,j) + AS(i,j) - S(i,j) \leq amd_r(i,j)amd_c(i,j) + T_1 - T_2 - T_3 + T_4 + T_5. \tag{4.6.21}$$

Moreover, $\forall e \in \mathcal{F},\ |\mathcal{U}_e| \leq amd_r(i,j)$ which implies that

$$T_4 - T_3 \leq 0. \tag{4.6.22}$$

Furthermore, since

$$|\hat{\mathcal{L}}_p| + \sum_{e \in \mathcal{F}} |\tilde{\mathcal{L}}_e \cap \left(\bigcup_{e' \in \mathcal{F},\, e' > e} \tilde{\mathcal{L}}_{e'} \right)| \leq amd_c(i,j),$$

we thus have

$$T_1 + T_5 - T_2 \leq 0. \tag{4.6.23}$$

From (4.6.21), (4.6.22) and (4.6.23), we have

$$d_r(i,j)d_c(i,j) - amd_r(i,j)amd_c(i,j) + AS - S \leq 0, \tag{4.6.24}$$

which proves Property 6.5. □

We now explain that, although computing $AS(i,j)$ is not trivial it is however not costly since it can be based on quantities already used in Algorithm 4.6.6 to compute the approximate degrees combined with local correction terms. Algorithm 4.6.7 will then revisit Algorithm 4.6.6 to show how to compute the new `AMFI` metric.

To compute the area AS, we need to evaluate $\hat{\mathcal{U}}_e$, $\hat{\mathcal{L}}_e$, $\tilde{\mathcal{U}}_e$ and $\tilde{\mathcal{L}}_e$ (see equation (4.6.13)). It appears to be difficult to compute these quantities directly (without an extra pass on the element structures) during the loops of Algorithm 4.6.6. Indeed only the quantities $|\mathcal{U}_e|$, $|\mathcal{L}_e|$, $|\mathcal{U}_e \setminus \mathcal{U}_p|$, $|\mathcal{L}_e \setminus \mathcal{L}_p|$, $|\mathcal{U}_e \cap \mathcal{U}_p|$ and $|\mathcal{L}_e \cap \mathcal{L}_p|$ are known. For each entry (i,j) involved in the metric update, two quantities are computed. An approximation of $AS(i,j)$, $area_{ij}$, such that

$$area_{ij} = \sum_{e \in \mathcal{F}} (|\mathcal{U}_e \setminus \mathcal{U}_p||\mathcal{L}_e| + |\mathcal{L}_e \setminus \mathcal{L}_p||\mathcal{U}_e \cap \mathcal{U}_p|). \tag{4.6.25}$$

Local correction terms, cor_loc_{ij}, are then computed so that

$$AS(i,j) = area_{ij} - cor\_loc_{ij}. \tag{4.6.26}$$

Computing $area_{ij}$ only involves already known quantities. To compute the local correction terms cor_loc_{ij} we have to take into account for each $e \in \mathcal{F}$ all possible cases: $e \in \mathcal{R}_i \setminus \mathcal{C}_j$, $e \in \mathcal{C}_j \setminus \mathcal{R}_i$, and $e \in \mathcal{C}_j \cap \mathcal{R}_i$.

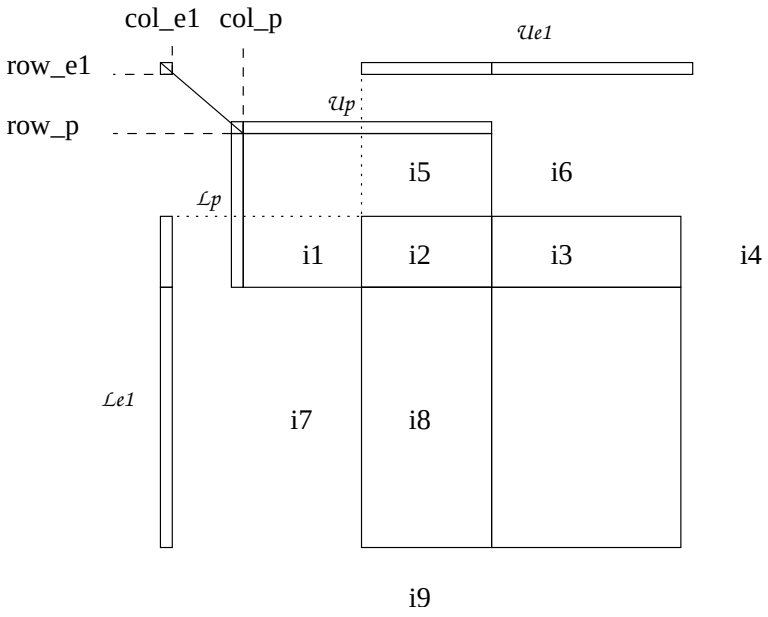


Figure 4.6.10: AMFI: different cases of local symmetrization. $i1, i2, i3, i4, i5, i6$ are cases encountered in *loop 1*, $i7, i8, i9$ are cases encountered in *loop 2*.

Algorithm 4.6.7 explains how to modify Algorithm 4.6.6 to compute both $area_{ij}$ and cor_loc_{ij} from which the AMFI metric can be computed. To help understanding how the local correction terms are computed, we have shown in Figure 4.6.10 the areas corresponding to different values of the local correction. In the following, we explain how the local correction terms of *Loop 1* of the algorithm are computed. For $i \in \mathcal{L}_p$ (*Loop 1*), we note that, for each element $e \in \mathcal{R}_i$, $|\mathcal{U}_e \setminus \mathcal{U}_p|$ must be added to the correction term since it was taken into account by mistake in $|\mathcal{U}_e \setminus \mathcal{U}_p||\mathcal{L}_e|$. This applies to cases i1, i2, i3 and i4 and not to cases i5 and i6 for which $|\mathcal{U}_e \setminus \mathcal{U}_p|$ is not added at line [1] of the algorithm since $e1 \notin \mathcal{R}_{i5}$ and $e1 \notin \mathcal{R}_{i6}$ in Figure 4.6.10. We then see at line [2] of the algorithm that, since $j \in \mathcal{U}_p$, $|\mathcal{L}_e \setminus \mathcal{L}_p|$ must be added to the local correction (case i2). For case i3 at line [3] of the algorithm, the complete column structure $|\mathcal{L}_e|$ has been taken into account in $area_{ij}$ and must be added to the correction term. Furthermore, since the entry (i, j) had already been counted in $|\mathcal{U}_e \setminus \mathcal{U}_p|$, one should remove it from the correction term at line [3] of the algorithm. For case i5 (i6) in Figure 4.6.10, we add $|\mathcal{L}_e \setminus \mathcal{L}_p|$ ($|\mathcal{L}_e|$) that were counted by mistake in term $|\mathcal{L}_e \setminus \mathcal{L}_p||\mathcal{U}_e \cap \mathcal{U}_p|$ ($|\mathcal{U}_e \setminus \mathcal{U}_p||\mathcal{L}_e|$) of $area_{ij}$.

Using the modifications proposed in Algorithm 4.6.7, we define the AMFI metric as follows:

$$metric^{(k+1)}(i,j) = \min \left\{ \begin{array}{ll} amd_r(i,j)amd_c(i,j) - |\mathcal{U}_p \setminus j||\mathcal{L}_p \setminus i| - AS(i,j) \\ metric^{(k)}(i,j) & +|\mathcal{U}_p \setminus j| \times amd_r(i,j) \\ & +|\mathcal{L}_p \setminus i| \times amd_c(i,j) \\ & -2 \times |\mathcal{U}_p \setminus j| \times |\mathcal{L}_p \setminus i| \end{array} \right. \tag{4.6.27}$$

Algorithm 4.6.7 Revisiting Algorithm 4.6.6 to compute `AMFI`.

for $i \in \mathcal{L}_p$ **do** /* *Loop 1* */
  $area\_save_i = \sum_{e \in \mathcal{R}_i} (|\mathcal{U}_e \setminus \mathcal{U}_p||\mathcal{L}_e| + |\mathcal{L}_e \setminus \mathcal{L}_p||\mathcal{U}_e \cap \mathcal{U}_p|)$
1 $cor\_loc\_save_i = \sum_{e \in \mathcal{R}_i} |\mathcal{U}_e \setminus \mathcal{U}_p|$ /* cases $i1, i2, i3$ and $i4$ */
  for each j such that $(i,j) \in \mathbf{C}$ **do**
    $area_{ij} = area\_save_i$, $cor\_loc_{ij} = cor\_loc\_save_i$
    for each $e \in \mathcal{C}_j$ **do**
      if $e \in \mathcal{R}_i$ **then** /* e already visited, its fi ll-in already counted in $area_{ij}$ */
        if $j \in \mathcal{U}_p$ **then**
2          $cor\_loc_{ij} = cor\_loc_{ij} + |\mathcal{L}_e \setminus \mathcal{L}_p|$ /* case i2 */
        else
3          $cor\_loc_{ij} = cor\_loc_{ij} + |\mathcal{L}_e| - 1$ /* case i3 */
        end if
      else /* e not already visited, we need to count its corresponding area */
        $area_{ij} = area_{ij} + |\mathcal{U}_e \setminus \mathcal{U}_p||\mathcal{L}_e| + |\mathcal{L}_e \setminus \mathcal{L}_p||\mathcal{U}_e \cap \mathcal{U}_p|$
        if $j \in \mathcal{U}_p$ **then**
4          $cor\_loc_{ij} = cor\_loc_{ij} + |\mathcal{L}_e \setminus \mathcal{L}_p|$ /* case i5 */
        else
5          $cor\_loc_{ij} = cor\_loc_{ij} + |\mathcal{L}_e|$ /* case i6 */
        end if
      end if
    end for
  end for
end for
for each $j \in \mathcal{U}_p$ **do** /* *Loop 2* */
  $area\_save_j = \sum_{e \in \mathcal{C}_j} (|\mathcal{U}_e \setminus \mathcal{U}_p||\mathcal{L}_e| + |\mathcal{L}_e \setminus \mathcal{L}_p||\mathcal{U}_e \cap \mathcal{U}_p|)$
  $cor\_loc\_save_j = \sum_{e \in \mathcal{C}_j} |\mathcal{L}_e \setminus \mathcal{L}_p|$ /* cases $i8$ and $i9$ */
  for each row i such that $(i,j) \in \mathbf{C}$ and $i \notin \mathcal{L}_p$ **do**
    $area_{ij} = area\_save_j$; $cor\_loc_{ij} = cor\_loc\_save_j$
    for each $e \in \mathcal{R}_i$ **do**
      if $e \notin \mathcal{C}_j$ **then**
        $area_{ij} = area_{ij} + |\mathcal{U}_e \setminus \mathcal{U}_p||\mathcal{L}_e| + |\mathcal{L}_e \setminus \mathcal{L}_p||\mathcal{U}_e \cap \mathcal{U}_p|$
        $cor\_loc_{ij} = cor\_loc_{ij} + |\mathcal{U}_e|$ /* case i7 */
      else
        $cor\_loc_{ij} = cor\_loc_{ij} + |\mathcal{U}_e| - 1$ /* case i8 */
      end if
    end for
  end for
end for

In practice we use $\overline{amd}_r(i,j)$ and $\overline{amd}_c(i,j)$ as defined in equations (4.6.5) and (4.6.6) instead of $amd_r(i,j)$ and $amd_c(i,j)$.

Because of that it may happen that $\overline{amd}_r(i,j)\overline{amd}_c(i,j) - |\mathcal{U}_p \setminus j||\mathcal{L}_p \setminus i| - AS(i,j)$ becomes negative. In such cases, as is done in the symmetric code or in the diagonal Markowitz code, one can artificially set the metric to 0.

We propose here an alternative that could also be applied to the other approaches. When our metric becomes negative, we propose reducing the size of the area AS. We introduce a row scaling term $rowscale$, and a column scaling term $colscale$. Using the notation of equations (4.6.5) and (4.6.6) we have

$$rowscale = \frac{\overline{amd}_r(i,j)}{amd_r(i,j)} \text{ and } colscale = \frac{\overline{amd}_c(i,j)}{amd_c(i,j)}.$$

If one scales the area AS by $rowscale \times colscale$, then we ensure a positive metric and avoid tie-breaking problems due to metrics equal to 0. Our final `AMFI` metric is then defined as follows:

$$metric^{(k+1)}(i,j) = \min \left\{ \begin{array}{ll} \overline{amd}_r(i,j)\overline{amd}_c(i,j) - |\mathcal{U}_p \setminus j||\mathcal{L}_p \setminus i| & \\ \quad -rowscale\ colscale\ AS(i,j) & \\ metric^{(k)}(i,j) & +|\mathcal{U}_p \setminus j| \times \overline{amd}_r(i,j) \\ & +|\mathcal{L}_p \setminus i| \times \overline{amd}_c(i,j) \\ & -2 \times |\mathcal{U}_p \setminus j| \times |\mathcal{L}_p \setminus i| \end{array} \right. \tag{4.6.28}$$

4.6.6.5 Complexity

It is clear from our algorithms that the complexity of all the metrics described in this section is comparable. That is why we will concentrate our analysis on Algorithm 4.6.6 which computes the approximation of the Markowitz cost at step p of Gaussian elimination. The quantities $|\mathcal{A}_{i*} \setminus \mathcal{U}_p|$ and $|\mathcal{A}_{*j} \setminus \mathcal{L}_p|$ are computed only once. The total computation time for these quantities is bounded by

$$\mathcal{O}(\sum_i |\mathcal{A}_{i*}| + \sum_j |\mathcal{A}_{*j}|) = \mathcal{O}(|E|).$$

The first loop is visited $|\mathcal{L}_p|$ times. The computation time of all the $rowdeg\_save$ and $coldeg\_save$ is thus

$$\mathcal{O}(\sum_{i \in \mathcal{L}_p} |\mathcal{R}_i|).$$

The computation of all the amd_r and amd_c terms takes time

$$\mathcal{O}\left(\sum_{i \in \mathcal{L}_p} |\mathcal{R}_i| + \sum_{i \in \mathcal{L}_p} \sum_{j \in R_i^C} |\mathcal{C}_j|\right). \tag{4.6.29}$$

Let us define $n_1, \ldots, n_{|\mathcal{L}_p|}$ by induction:

$$n_1 = arg\max |\mathcal{C}_j|$$

and for $i > 1$,

$$n_i = arg \max_{j \neq n_{i-1},\ldots,n_1} |\mathcal{C}_j|.$$

The double sum of equation (4.6.29) contains at most $k^p|\mathcal{L}_p|$ terms (k^p was defined in Section 4.5.1.1 and corresponds to the size of the largest row/column at step p). Note that at most k^p terms can be equal to $\mathcal{C}_{n_i}$ for all $i \in [1, |\mathcal{L}_p|]$. Thus, we have the following inequality:

$$\sum_{i \in \mathcal{L}_p} \sum_{j \in R_i^C} |\mathcal{C}_j| \leq k^p \times \sum_{1 \leq i \leq |\mathcal{L}_p|} |\mathcal{C}_{n_i}|.$$

Then we have

$$\sum_{i \in \mathcal{L}_p} (|\mathcal{R}_i| + \sum_{j \in R_i^C} |\mathcal{C}_j|) \leq k^p \times \left(\sum_{i \in \mathcal{L}_p} |\mathcal{R}_i| + \sum_{1 \leq i \leq |\mathcal{L}_p|} |\mathcal{C}_{n_i}| \right).$$

Finally, thanks to the in-place property of our algorithm to handle the bipartite quotient graph we obtain

$$\sum_{i \in \mathcal{L}_p} |\mathcal{R}_i| + \sum_{1 \leq i \leq |\mathcal{L}_p|} |\mathcal{C}_{n_i}| \leq |E|.$$

We can do the same computation for the second loop. Finally, the total cost for updating the metric is $\mathcal{O}(k^p|E|)$ per elimination step. The total time complexity is $\mathcal{O}(\sum_{1<p\leq n} k^p|E|)$. If k^p is bounded, the complexity becomes $\mathcal{O}(n|E|)$ which is also a bound for the `DMLS` time. In practice, we observe a linear behaviour compared with `DMLS` (see Section 4.7).

4.6.7 Supervariables and mass elimination

For the sake of clarity, the algorithms described in the previous section did not include supervariables. In this section, we first define our generalization of *supervariables* and *mass elimination* to bipartite quotient graphs with off-diagonal pivots. We then revisit the previous algorithms and explain what has to be modified to detect and exploit supervariables.

4.6.7.1 Adaptation of CMLS main scheme

Supervariables have been successfully used in the context of quotient graphs based orderings for symmetric matrices [4, 43, 56] and in the context of bipartite quotient graph when local symmetrization is performed and pivots are chosen from the diagonal [12]. It has been shown that the use of supervariables leads to a decrease in the number of metric computations and thus to a significant reduction in the cost of the ordering. With the `CMLS` algorithm we cannot use exactly the same kinds of supervariables because the approaches used in [4, 12, 43, 56] assume that pivots are on the diagonal so that a row can be associated with a column before being selected as pivot. That is why our concept of supervariable is closer to the one used in [48]: we define *indistinguishable row variables* (resp. *indistinguishable column variables*) as row variables (resp. columns

variables) which have the same adjacency in $\mathcal{G}$. Note that indistinguishability in $\mathcal{G}$ implies indistinguishability in G and that the reverse is not true.

If i and j are two indistinguishable row variables, they are replaced in $\mathcal{G}$ by a *row supervariable* containing both i and j, labelled by its *principal row variable* (i, say) [42, 43, 44]. The notation $\mathbf{i}$ is used to denote this row supervariable and $\mathbf{i} = \{i, j\}$. j is said to be a *subordinate row variable* of $\mathbf{i}$. i and j are said to be *constituent row variables* of the row supervariable $\mathbf{i}$ and the notation $i \in \mathbf{i}$ and $j \in \mathbf{i}$ is used. At the beginning of the Gaussian elimination, the row variables are said to be *simple row variables*. Each simple row variable i can also be seen as a row supervariable $\mathbf{i} = \{i\}$. For each row supervariable $\mathbf{i}$, $|\mathbf{i}|$ corresponds to its size, $i.e.$ its number of constituent variables.

Similar definitions and notation can be introduced for the *column supervariables*, the *principal column variables*, the *subordinate column variables*, the *constituent column variables* and the *simple column variables*. When it is clear from the context, we do not differentiate between a column or a row supervariable.

Let r_1 and r_2 be two row variables which belong to the same row supervariable $\mathbf{r}$ and c_1 and c_2 be two column variables which belong to the same column supervariable $\mathcal{M}$. After eliminating the pivot $p_1 = (r_1, c_1)$, $p_2 = (r_2, c_2)$ can be eliminated in $\mathcal{G}$ without causing extra fill-in. This process is called *mass elimination* [55]. Moreover the elimination of these two pivots creates a new element $p = (\mathbf{r}, \mathcal{M})$. Indeed since both rows and columns have the same adjacency structure ($\mathcal{L}_{p_1} = \mathcal{L}_{p_2}$ and $\mathcal{U}_{p_1} = \mathcal{U}_{p_2}$) only one representative is needed in $\mathcal{G}$. To simplify the description of the algorithms, for each constituent variable i of a supervariable $\mathbf{v}$, $\mathbf{i}$ will also be used to denote the supervariable $\mathbf{v}$.

Algorithm 4.6.8 is the adaptation of Algorithm 4.6.3 to supervariables. Note that on the bipartite quotient graph of $\mathbf{A}$, we manipulate supervariables whereas on the bipartite graph of $\mathbf{C}$ we only use simple variables. In the following paragraphs we comment on the modifications due to the introduction of supervariables.

The first modification of the algorithm concerns the use of a scaled structural metric during the pivot selection at step (2.1.1). The structural metric is divided by $min(|\mathbf{r}|, |\mathcal{M}|)$ since it corresponds to the size of the largest pivot block which could be eliminated if a pivot at the intersection of these row and column supervariables were selected.

The second modification of the algorithm concerns the elimination of supervariables. It is done in three steps. During the first step (2.1.1), a pivot is selected in $\mathbf{C}$. During the second step (2.1.2), we retrieve its associated row and column supervariable in $\mathbf{A}$. During the third step (2.3.1), we eliminate "as many as possible" variables in $\mathbf{C}$ belonging to the intersection of this row and column supervariables. Note that the meaning of "as many as possible" will depend on the context. If a hybrid strategy is used then pivot entries might be rejected because of numerical criteria. Furthermore, since the $\mathbf{C}$ matrix is updated during the elimination of a pivot, new nonzeros entries that might be at the intersection between the pattern of $\mathbf{C}$ and the supervariables need also be considered. The same three steps are applied to the mass elimination at step (2.5.3). Note that after step (2.3.1), if some constituent variables of a supervariable have not been eliminated, then we have to build a new supervariable at step (2.3.2) and to insert it in $\mathcal{G}$.

Algorithm 4.6.8 CMLS main steps with supervariables.

(1) Initialization of $\mathbf{C}$.
(2) Main loop
while there are still uneliminated variables **do**
(2.1.1) Select pivot (r, c) in $\mathbf{C}$ with priority based on structural and/or numerical metrics : the structural metric was scaled with $min(|\mathbf{r}|, |\mathcal{M}|)$.
(2.1.2) Let $p = (\mathbf{r}, \mathcal{M})$ be the new element.
(2.2) Build $\mathcal{L}_p$, $\mathcal{U}_p$ and let $\mathcal{F}_p = \mathcal{C}_{\mathcal{M}} \cup \mathcal{R}_{\mathbf{r}}$.
(2.3.1) Eliminate "as much as possible" entries $(i, j) \in \mathbf{C} \cap (\mathbf{r} \times \mathcal{M})$.
(2.3.2) Update supervariables and quotient graph:
Let r' be the set of constituent variables of $\mathbf{r}$ which have not been eliminated.
Let c' be the set of constituent variables of $\mathcal{M}$ which have not been eliminated.
if $r' \neq \emptyset$ **then**
Form a new row supervariable $\mathbf{r}'$ with $\mathcal{R}_{\mathbf{r}'} = \{p\}$ and $\mathcal{A}_{\mathbf{r}'*} = \emptyset$.
$\mathcal{L}_p = \mathcal{L}_p \cup \{\mathbf{r}'\}$
end if
if $c' \neq \emptyset$ **then**
Form a new row supervariable $\mathcal{M}'$ with $\mathcal{C}_{\mathcal{M}'} = \{p\}$ and $\mathcal{A}_{*\mathcal{M}'} = \emptyset$.
$\mathcal{U}_p = \mathcal{U}_p \cup \{\mathcal{M}'\}$
end if
(2.4) Update quotient graph $\mathcal{G}$ of $\mathbf{A}$ and compute hash keys for each row/column supervariable in $\mathcal{L}_p/\mathcal{U}_p$.
(2.5.1) *Update the structural metric and mark candidates for mass elimination (Algorithm 4.6.9).*
(2.5.2) Detect row and column supervariables using hash values.
(2.5.3) Mass elimination:
for all $(i, j) \in \mathbf{C}$ marked at step (2.5.1) **do**
Eliminate "as much as possible" pivots $(k, l) \in \mathbf{C} \cap (\mathbf{i} \times \mathbf{j})$ (remove k from $\mathbf{i}$ and remove l from $\mathbf{j}$).
if $\mathbf{i} = \emptyset$ **then** remove $\mathbf{i}$ from $\mathcal{G}$ and $\mathcal{L}_p = \mathcal{L}_p \setminus \{\mathbf{i}\}$.
if $\mathbf{j} = \emptyset$ **then** remove $\mathbf{j}$ from $\mathcal{G}$ and $\mathcal{U}_p = \mathcal{U}_p \setminus \{\mathbf{j}\}$.
end for
(2.6) Remove the absorbed elements.
end while

The third modification concerns the update of the bipartite quotient graph (step 2.4). For each row supervariable in $\mathcal{L}_p$ and for each column supervariable in $\mathcal{U}_p$, we compute a hash value (see for example [15]) to reduce the pairs of supervariables whose structure will be compared during step (2.5.2).

The final modification concerns the update of the structural metric at step (2.5.1) that will be fully described in Section 4.6.7.2. Note that at step (2.5.1) we also mark candidates for mass elimination that will be eliminated at step (2.5.3).

4.6.7.2 Revisiting computation of the structural metrics

When using supervariables, after eliminating pivot p, the AMD like approximate external row and column degrees (equations (4.6.3) and (4.6.4)) become:

$$amd_r(i,j) = \begin{array}{l} |\mathcal{A}_{\mathbf{i}*} \setminus \mathcal{U}_p| + |\mathcal{U}_p \setminus \{\mathbf{j}\}| + \sum_{e \in \mathcal{R}_{\mathbf{i}} \cup \mathcal{C}_{\mathbf{j}}} (|\mathcal{U}_e \setminus \mathcal{U}_p|) - \alpha_{\mathbf{j}} \, |\mathbf{j}|, \\ \text{with } \alpha_{\mathbf{j}} = \max(|\mathcal{C}_{\mathbf{j}}|, 1) \text{ if } \mathbf{j} \notin \mathcal{U}_p \text{ else } \alpha_{\mathbf{j}} = 0. \end{array} \tag{4.6.30}$$

$$amd_c(i,j) = \begin{array}{l} |\mathcal{A}_{*\mathbf{j}} \setminus \mathcal{L}_p| + |\mathcal{L}_p \setminus \{\mathbf{i}\}| + \sum_{e \in \mathcal{R}_\mathbf{i} \cup \mathcal{C}_\mathbf{j}} (|\mathcal{L}_e \setminus \mathcal{L}_p|) - \beta_\mathbf{i} \, |\mathbf{i}|, \\ \text{with } \beta_\mathbf{i} = \max(|\mathcal{R}_\mathbf{i}|, 1) \text{ if } \mathbf{i} \notin \mathcal{L}_p \text{ else } \beta_\mathbf{i} = 0. \end{array} \tag{4.6.31}$$

Indeed the metric computation has to take into account the area of the intersection between row and column supervariables. Equations (4.6.5) and (4.6.6) then become:

$$\overline{amd}_r(i,j) = \min(amd_r(i,j), n - k - |\mathbf{j}|), \tag{4.6.32}$$

and

$$\overline{amd}_c(i,j) = \min(amd_c(i,j), n - k - |\mathbf{i}|). \tag{4.6.33}$$

In Algorithm 4.6.9 we revise Algorithm 4.6.6 to include supervariables. We recall that this algorithm is composed of two external loops. *Loop 1* refers to the pass over variables in $\mathcal{L}_p$ and *Loop 2* refers to the pass over variables in $\mathcal{U}_p$. Note that supplementary external loops have been added since, for each supervariable, we need to visit all of its constituent variables.

We save the values of the metrics which can be reused in the future and thus benefit from a double improvement. Firstly, at steps 1, 2, 6 and 7 all quantities are computed once for each supervariable instead of once for each variable. Secondly, if an entry of $\mathbf{C}$ belongs to the intersection between a row and a column supervariable which has already been visited, its metric is saved at step (5b) and can be reused. Since the saved quantities are no longer needed when a row supervariable is finished, saving this information requires at most an array of size n. The same property is used at steps 8, 9 and 10 of Loop 2. Finally, at step 5c candidates for mass elimination can be easily detected and are also marked.

We now discuss the modifications resulting from the use of supervariables on the computation of the approximate Markowitz cost and of the `AMF` and `AMFI` metrics. Equations (4.6.7) and (4.6.8), with the above redefinition of the approximate external degrees, are still valid to compute the approximate Markowitz cost and the `AMF` metric respectively. Adapting equation (4.6.28) to compute the `AMFI` metric requires some more care and will be described in the next paragraph.

In Section 4.6.6.4, we have shown that to evaluate the `AMFI` metric of an entry (i,j) we need to compute the area $AS(i,j)$ (see equations (4.6.27) and (4.6.28)). To compute $AS(i,j)$ we have also shown that it is easier to define it as as the sum of two terms $area_{ij}$ and cor_loc_{ij} (see equation (4.6.26)). The $area_{ij}$ term (see equation (4.6.25)) is related to the areas of all the bi-cliques of the already eliminated elements so that its computation does not depend on the use of supervariables. The computation of the local correction terms cor_loc_{ij} must however be revisited. We split this correction term into two parts. Each part, $row\_cor\_loc_{ij}$ and $col\_cor\_loc_{ij}$, refers to entries in the row supervariable and to entries in the column supervariable respectively so that the new equation for defining the area AS is :

$$AS(i,j) = area_{ij} - row\_cor\_loc_{ij} \, |\mathbf{i}| - col\_cor\_loc_{ij} |\mathbf{j}|. \tag{4.6.34}$$

For an entry $(i,j) \in \mathbf{C}$, the expression of the local corrections for the different cases of Figure 4.6.10 is:

- Case $i1$: $row\_cor\_loc = |\mathcal{U}_e \setminus \mathcal{U}_p|$ and $col\_cor\_loc = 0$,

- Case $i2$: $row\_cor\_loc = |\mathcal{U}_e \setminus \mathcal{U}_p|$ and $col\_cor\_loc = |\mathcal{L}_e \setminus \mathcal{L}_p|$,
- Case $i3$: $row\_cor\_loc = |\mathcal{U}_e \setminus \mathcal{U}_p|$ and $col\_cor\_loc = |\mathcal{L}_e| - |\mathbf{i}|$,
- Case $i4$: $row\_cor\_loc = |\mathcal{U}_e \setminus \mathcal{U}_p|$ and $col\_cor\_loc = 0$,
- Case $i5$: $row\_cor\_loc = 0$ and $col\_cor\_loc = |\mathcal{L}_e \setminus \mathcal{L}_p|$,
- Case $i6$: $row\_cor\_loc = 0$ and $col\_cor\_loc = |\mathcal{L}_e|$,
- Case $i7$: $row\_cor\_loc = |\mathcal{U}_e|$ and $col\_cor\_loc = 0$,
- Case $i8$: $row\_cor\_loc = |\mathcal{U}_e| - |\mathbf{j}|$ and $col\_cor\_loc = |\mathcal{L}_e \setminus \mathcal{L}_p|$,
- Case $i9$: $row\_cor\_loc = 0$ and $col\_cor\_loc = |\mathcal{L}_e \setminus \mathcal{L}_p|$.

It is now straightforward to accumulate these corrections by adapting Algorithm 4.6.7 to the modified scheme described in Algorithm 4.6.9. Note that we could have accumulated all the corrections in one term, but we would then have needed to multiply the above corrections by the size of the row or the column supervariable at each step of the accumulation. With our approach, we thus only perform two multiplications to compute the total correction.

Algorithm 4.6.9 Update of the structural metric using supervariables.

for each row supervariable $\mathbf{r} \in \mathcal{L}_p$ **do** /* *Loop 1:* */
1 $rowdeg\_save = |\mathcal{A}_{\mathbf{r}*} \setminus \mathcal{U}_p| + \sum_{e \in \mathcal{R}_\mathbf{r}} |\mathcal{U}_e \setminus \mathcal{U}_p|$
2 $coldeg\_save = \sum_{e \in \mathcal{R}_\mathbf{r}} |\mathcal{L}_e \setminus \mathcal{L}_p|$
for each row constituent $i \in \mathbf{r}$ **do**
for each column j such that $(i,j) \in \mathbf{C}$ **do**
if the metric of $(\mathbf{r}, \mathbf{j})$ has not already been computed **then**
3 $amd_r(\mathbf{r}, \mathbf{j}) = rowdeg\_save + \sum_{e \in \mathcal{C}_\mathbf{j} \setminus \mathcal{R}_\mathbf{r}} |\mathcal{U}_e \setminus \mathcal{U}_p| + |\mathcal{U}_p \setminus \{\mathbf{j}\}| - \alpha_\mathbf{j} |\mathbf{j}|$
4 $amd_c(\mathbf{r}, \mathbf{j}) = coldeg\_save + |\mathcal{A}_{*\mathbf{j}} \setminus \mathcal{L}_p| + \sum_{e \in \mathcal{C}_\mathbf{j} \setminus \mathcal{R}_\mathbf{r}} |\mathcal{L}_e \setminus \mathcal{L}_p| + |\mathcal{L}_p \setminus \{\mathbf{r}\}|$
5 $metric(\mathbf{r}, \mathbf{j}) = f(amd_r(\mathbf{r}, \mathbf{j}), amd_c(\mathbf{r}, \mathbf{j}))$
5b save $metric(\mathbf{r}, \mathbf{j})$ until the end of the processing of $\mathbf{r}$.
if $\mathbf{j} \in \mathcal{U}_p$ and $amd_r(\mathbf{r}, \mathbf{j}) = |\mathcal{U}_p \setminus \{\mathbf{j}\}|$ and $amd_c(\mathbf{r}, \mathbf{j}) = |\mathcal{L}_p \setminus \{\mathbf{r}\}|$ **then**
5c mark (i,j) as candidate for mass elimination.
end if
end if
$metric(i,j) = metric(\mathbf{r}, \mathbf{j})$
end for
end for
end for
for each column supervariable $\mathcal{M} \in \mathcal{U}_p$ **do** /* *Loop 2:* */
6 $rowdeg\_save = \sum_{e \in \mathcal{C}_\mathcal{M}} |\mathcal{U}_e \setminus \mathcal{U}_p|$
7 $coldeg\_save = |\mathcal{A}_{*\mathcal{M}} \setminus \mathcal{L}_p| + \sum_{e \in \mathcal{C}_\mathcal{M}} |\mathcal{L}_e \setminus \mathcal{L}_p|$
for each column constituent $j \in \mathcal{M}$ **do**
for each row i such that $(i,j) \in \mathbf{C}$ and $\mathbf{i} \notin \mathcal{L}_p$ **do**
if the metric of $(\mathbf{i}, \mathcal{M})$ has not already been computed **then**
8 $amd_r(\mathbf{i}, \mathcal{M}) = rowdeg\_save + |\mathcal{A}_{\mathbf{i}*} \setminus \mathcal{U}_p| + \sum_{e \in \mathcal{R}_\mathbf{i} \setminus \mathcal{C}_\mathcal{M}} |\mathcal{U}_e \setminus \mathcal{U}_p| + |\mathcal{U}_p \setminus \{\mathcal{M}\}|$
9 $amd_c(\mathbf{i}, \mathcal{M}) = coldeg\_save + \sum_{e \in \mathcal{R}_\mathbf{i} \setminus \mathcal{C}_\mathcal{M}} |\mathcal{L}_e \setminus \mathcal{L}_p| + |\mathcal{L}_p| - \beta_i |\mathbf{i}|$
10 $metric(\mathbf{i}, \mathcal{M}) = f(amd_c(\mathbf{i}, \mathcal{M}), amd_r(\mathbf{i}, \mathcal{M}))$
save $metric(\mathbf{i}, \mathcal{M})$ until the end of the processing of $\mathcal{M}$.
end if
$metric(i,j) = metric(\mathbf{i}, \mathcal{M})$
end for
end for
end for

4.6.8 Aggressive absorption

4.6.8.1 Classical aggressive absorption: advantages and drawbacks

The aggressive absorption process performed in the `AMD` and `DMLS` algorithms can be generalized to the `CMLS` algorithm. For an element e, if the condition

$$\mathcal{U}_e \subset \mathcal{U}_p \text{ and } \mathcal{L}_e \subset \mathcal{L}_p, \tag{4.6.35}$$

holds then e is no longer needed because variables adjacent to e will also be adjacent to p (left picture of Figure 4.6.11). If $e \notin \mathcal{U}_p$ and $e \notin \mathcal{L}_p$, e is not absorbed in a classical way. As already observed, aggressive absorption may improve the precision of the approximate degrees because it may reduce the number of element candidates for overlapping. Furthermore, when an element is aggressively absorbed its parent is set to p

in the elimination tree. This can be a drawback because elimination trees with long chains can appear and limit the parallelism due to the sparsity (middle picture of Figure 4.6.11).

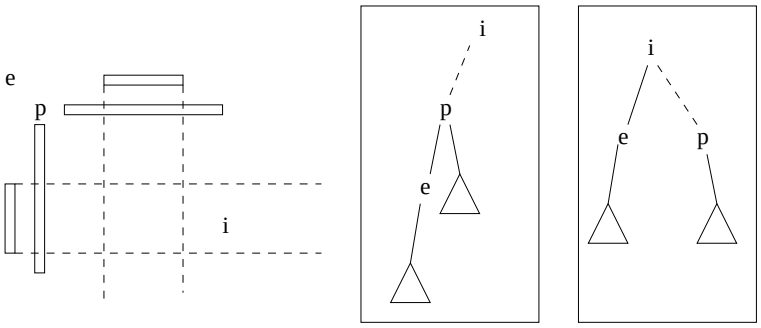


Figure 4.6.11: Aggressive absorption. From left to right: condition for aggressive absorption/ classical aggressive absorption elimination tree/ modified aggressive absorption elimination tree.

4.6.8.2 New aggressive absorption

To avoid this lost of parallelism, we propose performing the aggressive absorption in two steps. Firstly, when an element e satisfies condition (4.6.35), its adjacency is removed. This element becomes an element such that $|\mathcal{U}_e| = 0$ and $|\mathcal{L}_e| = 0$. Finally when row $i \in \mathcal{L}_e$ or column $j \in \mathcal{U}_e$ is removed e is attached to the current pivot (i, j) in the elimination tree (right picture of Figure 4.6.11). This new aggressive absorption can also be applied to the `DMLS` code and to even more classical minimum degree orderings for symmetric matrices and has the advantage of better preserving parallelism while improving the quality of the approximate degrees and reducing the size of the quotient graph.

4.7 Experiments

In this section we analyse the influence of the preprocessing and the updating strategies of the CMLS ordering on the performance of sparse solvers. Our new ordering will be compared to the combination of DMLS+MC64 because it is the most robust in-place local heuristic (better than AMD see [12]) in terms of numerical stability and fill-in in the factors.

With the CMLS ordering, our pivot sequence results from a combination of structural and numerical information (even when only structural metrics are used to select the pivots, the initialization of our constraint matrix is based on numerical considerations). Therefore it is important to analyse the numerical quality of the proposed sequence of pivots. In this context, for a very different motivation, we may want to experiment both an approach that performs partial pivoting to preserve numerical stability and with an approach based on static pivoting. In the first case, the numerical quality of the proposed sequence of pivots is not so critical to obtaining a backward stable factorization and we expect to improve the sparsity of the factors because of the freedom to select entries in the constraint matrix $\mathbf{C}$. In the case of a static pivoting based factorization, we expect that the capacity of CMLS to select pivots according to numerical criteria can be used to control the numerical quality of the sequence better while still offering some more freedom than a diagonal Markowitz algorithm. In fact with the CMLS algorithm we can define a family of orderings and can expect that two probably different members of this family can be used to satisfy these two cases: a CMLS ordering in which $\mathbf{C}$ offers a lot of freedom to choose the pivots and a CMLS ordering in which the selection of the pivots is strongly guided by the numerical values in $\mathbf{C}$.

To represent each class of approaches to numerical factorization, we consider the multifrontal code MA41_UNS [13, 6]) which performs numerical pivoting during the factorization and the supernodal code SuperLU_DIST [79] which performs static pivoting. Both codes are run in sequential mode. As shown in [9, 13, 67, 26] the approaches used to factorize the matrix in MA41_UNS and SuperLU_DIST are very competitive in shared/sequential and distributed memory environments respectively. Note also that because of the important algorithmic similarities between MA41_UNS and the distributed memory code MUMPS, this work will be also very beneficial to the distributed memory code.

In Section 4.7.1, we present our experimental environment. In Section 4.7.2, we focus on structural metrics. In Section 4.7.3, we illustrate the benefits of using a hybrid strategy with one threshold. In Section 4.7.4, we present an alternative strategy for the preprocessing step and analyse its influence on the performance. Performance in terms of time and memory used during factorization is reported in Section 4.7.5.

4.7.1 Experimental environment

4.7.1.1 What do we want to measure ?

To evaluate the quality of our sequence of pivots with MA41_UNS, we measure the ratio between the number of nonzeros in the factors and the forecast number of nonzeros in the factors (delayed pivots will increase this ratio). Note that from a software point of

view it is critical that the estimations reflect the reality. Concerning SuperLU_DIST, we measure the component-wise backward error of the solution [14] before and after iterative refinement and the number of steps of iterative refinement. Note that one step of iterative refinement costs at least as much as one forward and backward substitution. The cost of the solution phase is thus very much related to the number of steps of iterative refinement. Note that when we report the number of operations in our statistics, it always corresponds to the number of operations actually performed during the factorization phase. With MA41_UNS, the extra cost due to numerical pivoting is thus always included.

We also want to evaluate the gains from our new ordering and what its cost is. On the one hand, we expect our ordering to be slower than DMLS because it performs more metric computations and because it has to perform the explicit storage and manipulation of the constraint matrix $\mathbf{C}$. Indeed even if the pivot sequence were limited to the transversal of the matrix then one should expect CMLS to be more costly than DMLS. On the other hand, we expect to decrease the fill-in in the factors and the number of operations performed during the factorization phase, and so to decrease the factorization time.

Moreover both factorization codes are sensitive, though in a different way, to the amalgamation of nodes of the elimination tree for MA41_UNS and of the *edag* (elimination direct acyclic graph) [57] for SuperLU_DIST. For our performance analysis, we prefer to separate the influence of the ordering on number of nonzeros in the factors from the fill-in increase, even if relatively small, due to this amalgamation. Therefore the amalgamation will be activated only when we are concerned by the time and the memory for factorization in Section 4.7.5. Otherwise no amalgamation is performed.

To be less sensitive to the effects of tie-breaking, we systematically apply random row and column permutations to our initial matrix. We run each problem with five random permutations and select the run whose ordering returns the median fill-in in the factors. When, in Section 4.7.5, we analyse the influence of the orderings on the factorization time then we activate the amalgamation process and select the execution which corresponds to the median time.

4.7.1.2 Test matrices and computing environment

A representative set of all the matrices of order between 10000 and 100000 has been selected (see Table 4.7.1) from Tim Davis' collection [25]. All our results have been obtained on a Linux PC computer (Pentium 4, 2.8 GHz, 2 GBytes of memory and 1 MByte of cache).

If we consider a matrix $\mathbf{A} = (a_{ij})$, its **structural symmetry**, $s(\mathbf{A})$ is defined as:

$$s(\mathbf{A}) = \frac{\text{size } \{(i,j) \text{ s.t. } a_{ij} \neq 0 \text{ and } a_{ji} \neq 0\}}{nnz(\mathbf{A})}.$$

In the remainder of this section, the symmetry of a matrix always refers to the structural symmetry once the MC64 permutation has been applied (see column sym of Table 4.7.1). We expect our CMLS ordering to be more efficient on very structurally unsymmetric matrices and have thus decided to separate our test problems into two sets. The first set with 21 matrices (Set 1) refers to the matrices with a structural symmetry lower than 0.5 and the second set with 19 matrices (Set 2) corresponds to all the others.

Group/Matrix	n	nnz $\times 10^3$	sym	description
Set 1				
Vavasis/av41092	41092	1683	0.08	Unstructured finite element
Hollinger/g7jac140sc	41490	565	0.10	Economic model
Hollinger/g7jac120sc	35550	475	0.10	Economic model
Hollinger/jan99jac120sc	41374	260	0.16	Economic model
Hollinger/jan99jac100sc	34454	215	0.16	Economic model
Grund/bayer10	13436	94	0.18	Chemical process simulation
Grund/bayer04	20545	159	0.19	Chemical process simulation
Mallya/lhr34c	35152	764	0.19	Light hydrocarbon recovery
Mallya/lhr71c	70304	1528	0.20	Light hydrocarbon recovery
Hollinger/mark3jac120sc	54929	342	0.21	Economic model
Hollinger/mark3jac140sc	64089	399	0.21	Economic model
Hohn/sinc15	11532	568	0.27	Single-material crack problem (sinc-basis)
Hohn/sinc18	16428	973	0.27	Single-material crack problem (sinc-basis)
Zhao/Zhao2	33861	166	0.27	Electromagnetics
Hohn/fd18	16428	63	0.28	Single-material crack problem (finite difference)
Hohn/fd15	11532	44	0.29	Single-material crack problem (finite difference)
Sandia/mult_dcop_03	25187	193	0.36	Circuit simulation
Sandia/mult_dcop_02	25187	193	0.36	Circuit simulation
ATandT/onetone1	36057	341	0.43	Harmonic balance method
Grund/poli_large	15575	33	0.47	Chemical process simulation
Simon/bbmat	38744	1771	0.49	2D airfoil, turbulence
Set 2				
Shen/shermanACb	18510	145	0.50	Matrix from Kai Shen
Goodwin/rim	22560	1014	0.54	Finite-element method, fluid mechanics problem
ATandT/onetone2	36057	227	0.56	Harmonic balance method
Shyy/shyy161	76480	329	0.69	CFD / Navier-Stokes equations
Bomhof/circuit_3	12127	48	0.70	Circuit simulation
Averous/epb2	25228	175	0.71	Static simulation of a plate fin heat exchanger
Averous/epb3	84617	463	0.72	Static simulation of a plate fin heat exchanger
Bomhof/circuit_4	80209	307	0.73	Circuit simulation
Shen/e40r0100	17281	553	0.88	Matrix from Kai Shen
FEMLAB/ns3Da	20414	1679	0.91	FEMLAB test matrix
Sanghavi/ecl32	51993	380	0.93	Semiconductor device simulation
Zhao/Zhao1	33861	166	0.93	Electromagnetics
Bai/af23560	23560	484	0.97	Airfoil, non-Hermitian eigenvalue problem
Schenk_IBMSDS/3D_28984	28984	599	0.98	Semiconductor Device Simulation (IBM)
Schenk_IBMSDS/3D_51448	51448	1056	0.99	Semiconductor Device Simulation (IBM)
Schenk_IBMSDS/ibm_matr	51448	1056	0.99	Semiconductor Device Simulation (IBM)
Schenk_IBMSDS/2D_54019	54019	996	0.99	Semiconductor Device Simulation (IBM)
Schenk_IBMSDS/2D_27628	27628	442	0.99	Semiconductor Device Simulation (IBM)
FEMLAB/sme3Da	12504	874	1.00	FEMLAB test matrix

Table 4.7.1: Test matrices.

For each table presented in this section, we select a subset of the most significant results. However, we also always report average statistics with respect to each complete set and provide the complete tables of result in Appendix B. For a given criterion c (number of operations, fill-in,...), we define the average gain of CMLS over DMLS as the mean of the $c(\texttt{DMLS})/c(\texttt{CMLS}) - 1$ quantities.

The mult_dcop_02 and mult_dcop_3 matrices are special problems since they have 7418 and 7448 irreducible components respectively. We add them to our set to illustrate the effect of the one way absorption discussed in Section 4.6.4. We also observe, as one could have expected, that if we offer freedom to CMLS to select pivots outside the irreducible components of matrices that have a large number of irreducible blocks, then the fill-in increases. Therefore, even though we provide complete results in Appendix B, we decide not to take them into account when computing the average behaviour.

4.7.1.3 CMLS default parameters

When it is not explicitly mentioned, the initialization of the constraint matrix $\mathbf{C}$ is done with the following options:

- MC64 based preprocessing, as explained in Section 4.3.4, is used.
- The dropping threshold is adjusted dynamically according to the repartition of the entries of $\mathbf{A}$ (see DYNDROP strategy of Section 4.6.2) and $mindrop = 0.1$.
- To limit the initial number of entries in $\mathbf{C}$ and the complexity of our algorithms we set the parameter $StructThresh$ of Algorithm 4.3.1 to

$$n \times \max(2, \min(3, \frac{nnz(\mathbf{A})}{4n})).$$

- To limit the number of compressions of $\mathbf{C}$ related data structures, we double the size of the data structures associated with $\mathbf{C}_0$.

With such settings the number of entries in $\mathbf{C}_0$ is bounded by $3n$ and the number of entries in $\mathbf{C}$ that can be represented by our data structures is bounded by $6n$.

When it is not explicitly mentioned, the CMLS ordering is computed with the following options:

- We use the Improved Approximate Minimum Fill metric AMFI of Section 4.6.6.4 (the same metric is also used in DMLS) since it is the most efficient metric for both orderings.
- To have an efficient in-place implementation, MATCHUPDATE is performed automatically when $(|U_p| - 1) \times (|L_p| - 1) > n$, $i.e.$ the area of the contribution block in $\mathbf{C}$ is greater than n.
- We activate our new aggressive absorption in CMLS.

4.7.2 Structural strategy

4.7.2.1 Limiting the size of $\mathbf{C}_0$ to n

In this section, we assume that the constraint matrix $\mathbf{C}_0$ contains only the entries from the MC64 matching. This approach is referred to as Diag in the tables of Appendix B. Note that in this case, since the set of candidate pivots for CMLS and DMLS is identical, one should expect a comparable behaviour in terms of fill-in in the factors. The initial objective of this test was then to quantify the extra cost for CMLS of handling more complex data structures and to the manipulation of the $\mathbf{C}$ matrix.

As expected, we see in Table 4.7.2 that CMLS is in general slower than DMLS. For example on g7jac140sc, CMLS spends more than 2.5 seconds to insert entries in the doubly linked lists and more than 3 seconds to remove entries from them. The two following opposite aspects may also influence the ordering time, but they are difficult to detect. On the one hand, the better we preserve sparsity, the smallest might be the quotient graph and so the faster we can expect to process it. On the other hand, the better we preserve sparsity, the fewer elements are absorbed, the fewer supervariables are detected and so the complexity might become higher.

However we also notice in Table 4.7.2 the difference of behaviour between the two orderings is not at all negligible. When using the CMLS ordering, MA41_UNS and SuperLU_DIST tend to have sparser factors and to do fewer operations and larger gains are obtained on more unsymmetric matrices. Moreover one should note that the numerical behaviour of both solvers is unchanged (for SuperLU_DIST, compare columns Diag and DMLS in Tables B.1.7 and B.1.8).

	MA41_UNS				SuperLU_DIST		ordering	
	size of factors		operations		size of factors		time	
Matrix	CMLS	DMLS	CMLS	DMLS	CMLS	DMLS	CMLS	DMLS
Set 1								
av41092	8687	9654	2798	3664	7223	8098	3.5	3.9
g7jac140sc	22385	24146	26452	30254	14953	15204	18.3	6.2
sinc18	32598	41563	62163	96970	28792	35287	18.1	19.7
mult_dcop_03	502	844	25	106	297	403	0.5	0.4
mult_dcop_02	439	860	14	111	292	403	0.5	0.4
bbmat	44799	53641	50887	77900	41040	41315	46.0	17.5
Avg. gain	7.5%		15.7%		7.1%		−38.7%	
Set 2								
Avg. gain	1.5%		5.4%		0.9%		−60.0%	

Table 4.7.2: Comparison between DMLS+MC64 and CMLS with $\mathbf{C}$ only the MC64 matching. Number of entries in thousands, number of operations in millions and times in seconds.

The following algorithmic differences can explain the behaviour of the CMLS and the DMLS orderings:

- DMLS performs variable elimination in two directions. The approximation of the row and column degrees and the structural metric are thus more accurate.
- Thanks to the one direction only variable elimination, CMLS can eliminate all

elements with empty adjacency in L or in U. It has thus more chance to preserve the sparsity of reducible matrices better.

- CMLS can create a row (resp. column) supervariable if two rows (resp. columns) have the same structure. DMLS can create a supervariable only if both rows and columns have the same structure. Thus, on the same quotient graph CMLS will detect more supervariables than DMLS. Note that the use of supervariables improves the precision of the structural metric. For example, if we consider variables i and j, which belong to the same row supervariable, then the entries in $\mathcal{A}_{i*}$ and $\mathcal{A}_{j*}$ will not be counted as fill-in.
- In the CMLS implementation, we use $rowscale$ and $colscale$ coefficients (see end of Section 4.6.6.4) to reduce the amount of tie-breaking between variables that would have a negative metric (reset to 0) with DMLS.

What is interesting to stress is that most of these algorithmic differences were justified by the fact that CMLS is designed to handle more complex situations than DMLS. For example, in CMLS, we are using one direction variable elimination only because we could not perform the two direction variable elimination. Supervariables are different because their definition in DMLS does not make sense with off-diagonal pivots. In fact what was not at all predicted is that the best implementation of DMLS should use the more general framework of the CMLS ordering. Note that applying these modifications to DMLS involves simpler and easier management of data structures than in CMLS.

Finally, as already mentioned before, the two mult_dcop_* matrices are reducible matrices. For each matrix, DMLS and CMLS detect 845 singletons during a common preprocessing step. Then on mult_dcop_02, DMLS detects 90 supplementary blocks versus 429 blocks for CMLS. On mult_dcop_03, DMLS detects 86 supplementary blocks versus 408 blocks for CMLS. We see in Table 4.7.2 that it has quite a significant impact on the performance of both solvers in terms of fill-in and number of operations.

4.7.2.2 $\mathbf{C}$ updates: MATCHUPDATE versus TOTALUPDATE

We now assume that we use the default parameters to construct $\mathbf{C}$ (see Section 4.7.1.3) and want to analyse the influence of the choice of structural update strategy. A first strategy is to perform only MATCHUPDATE. It is referred to as MAT in the tables and in our comments. A second strategy is to perform TOTALUPDATE on $\mathbf{C}$ (see Section A.4). This strategy is referred to as TOT in the tables and in our comments. An intermediate strategy is to perform TOTALUPDATE until a fixed number of entries with respect to the size of $\mathbf{C}_0$ have been added to $\mathbf{C}$ and then to only perform MATCHUPDATE. We expect that this strategy will limit the complexity of our ordering and that it will offer enough choices for the selection of the pivots. This third strategy is referred to as LIM in the tables and in our comments.

In Table 4.7.3 we report the size of the factors estimated by the MA41_UNS analysis phase. We also compare the number of operations. As expected larger gains are obtained for unsymmetric matrices (symmetry < 0.5). In general better results in terms of fill-in (average gain of 20% on Set 1) and number of operations (average gain of 40% on Set 1) are obtained with the CMLS ordering. Two classes of matrices in Set 1 do not follow

Matrix	Size of the factors MAT	LIM	TOT	DMLS	Number of operations MAT	LIM	TOT	DMLS
Set 1								
av41092	8171	8033	8056	9448	2650	2539	2652	3664
g7jac140sc	18284	16080	17065	24138	19656	13907	17015	30254
lhr34c	7628	8296	7775	6979	2238	2546	2403	1794
mark3jac140sc	16125	16793	17142	16372	9306	10618	11185	9295
sinc18	29842	31581	31118	41445	59757	64041	63273	96970
fd18	568	569	569	1082	44	43	45	128
mult_dcop_03	1018	991	1021	895	170	150	161	106
bbmat	39785	39034	38548	53596	40435	39553	40474	77900
Avg. gain	19.7%	22.3%	22.1%		35.5%	45.0%	40.6%	
Set 2								
onetone2	1611	1269	1280	1396	367	234	243	243
circuit_4	441	441	441	464	10	10	10	14
ibm_matr	39588	38492	38369	33202	39934	37495	38211	30736
Avg. gain	−3.0%	0.2%	0.1%		−2.5%	1.1%	1.2%	

Table 4.7.3: Estimated size of the factors and number of operations with MA41_UNS. MAT, LIM and TOT strategies are used. Number of entries in thousands and number of operations in millions.

this conclusion: the mark3jac*sc and the lhr* matrices. Furthermore, the MAT approach seems to be less robust than the two other strategies (see for example the g7jac*sc, bbmat, onetone2 and ibm_matr matrices). The LIM and TOT approaches have very similar behaviour. Table 4.7.3 also confirms that it is preferable not to offer freedom to select pivots outside the diagonal blocks of the BTF for reducible matrices (see results with mult_dcop_*).

Table 4.7.4 shows that comparable structural gains are obtained with CMLS on SuperLU_DIST (19.8% on Set 1).

Matrix	MAT	LIM	TOT	DMLS
Set 1				
g7jac140sc	13363	12062	12134	15204
lhr34c	5162	5770	6018	5787
mark3jac140sc	12822	13328	12531	12867
sinc18	25489	26162	27688	35287
fd18	523	531	520	935
mult_dcop_03	569	618	589	403
bbmat	40303	34813	35064	41315
Avg. gain	19.2%	19.7%	19.8%	
Set 2				
onetone2	1297	1034	1082	1032
ibm_matr	37631	37476	38262	32820
Avg. gain	−3.2%	−0.7%	−0.9%	

Table 4.7.4: Number of nonzeros in SuperLU_DIST factors (in thousands of reals).

In Table 4.7.5 we compare the ordering time of CMLS with the MAT, LIM and TOT strategies with the ordering time of DMLS. As expected, the three CMLS approaches are clearly slower than the DMLS ordering. CMLS pays the cost of indirect addressing and

Matrix	MAT	LIM	TOT	DMLS
Set 1				
av41092	8.8	10.4	10.4	3.9
g7jac140sc	22.5	27.7	71.3	6.2
lhr34c	6.6	8.7	8.2	3.4
mark3jac140sc	10.3	13.3	13.3	4.4
sinc18	29.6	27.3	35.3	19.7
fd18	0.3	0.4	0.4	0.1
mult_dcop_03	0.6	0.7	0.6	0.4
bbmat	51.6	57.5	115.3	17.5
Avg. gain	-55.0%	-62.9%	-65.6%	
Set 2				
onetone2	0.7	0.8	0.8	0.3
circuit_4	57.7	60.1	60.9	15.0
ibm_matr	5.2	8.3	8.9	0.6
Avg. gain	-71.7%	-76.9%	-79.1%	

Table 4.7.5: Ordering time (in seconds).

structural metric updates. Let us define for our discussion that a **quasi-dense variable** is a variable that has a significantly higher degree that the average degree of the variables in the graph. CMLS is also more sensitive to quasi-dense variables in the quotient graph than DMLS. Indeed the larger the size of $\mathbf{C}$, the higher the probability of accessing a quasi-dense structure. Moreover the same quasi-dense structure may have to be accessed more than once (see the inner loops of Algorithm 4.6.6). To illustrate this remark, we ran the CMLS ordering on circuit_4 with the following modification. Each time that we access an entry of metric greater than $5n$, we do not update its metric. By doing so, we skip the metric update $21,3 \times 10^6$ times and decrease the ordering time to 20.9 seconds (instead of 60 seconds as before). With DMLS the same modification only leads to skipping 6716 computations of the metric and does not produce any significant time reduction. Note that this modification did not change the quality of the ordering. Thus, developing quasi-dense rows/columns management [3] should reduce the gap between the two orderings.

Finally one should mention that, with CMLS, we also expect a reduction in the factorization and solve time (see Section 4.7.5). We also expect that the features of the CMLS algorithm enable us to replace the MC64 preprocessing by a cheaper preprocessing phase (see Section 4.7.4).

Since the LIM strategy is comparable to the TOT strategy in terms of sparsity and since it is significantly faster on problems such as the g7jac*sc, sinc* and bbmat matrices, it is used as our default update strategy in the remainder of this study.

4.7.2.3 Precision of the solution with structural strategies

Thanks to numerical pivoting, we have observed that with MA41_UNS we obtain the same precision (in terms of backward error) with the CMLS and the DMLS orderings. However, with MA41_UNS, numerical pivoting can significantly increase the estimated size of the factors. We see in Table B.1.3 of Appendix B that it is not the case and that the ratio

of the real factor size over the estimated size always represents an increase smaller than 10%. With MA41_UNS we thus fully benefit from the fill-in reduction of the structural strategies.

With SuperLU_DIST, we will show that the structural strategies do not provide a numerically good enough sequence of pivots. As expected we probably need to use hybrid strategies and we discuss this issue in Section 4.7.3. In the following we analyse in more detail the numerical behaviour of SuperLU_DIST with the CMLS ordering based on structural strategies. We show that the $mindrop$ parameter (defined in Section 4.6.2) used to control the numerical entries in the C_0 matrix can improve the numerical quality of our sequence of pivots.

4.7.2.4 Numerical precision of SuperLU_DIST with structural strategies

In Table 4.7.6, we report results on which at least one of our structural approaches fails to find an accurate solution. We see that it is more difficult for SuperLU_DIST to converge with the CMLS ordering than with the DMLS ordering.

Matrix	MAT	LIM	TOT	DMLS
Set 1				
av41092	1.0e+00	1.0e+00	1.0e-00	1.0e+00
g7jac140sc	1.0e+00	1.0e+00	1.0e+00	6.4e-16
g7jac120sc	1.0e+00	1.0e+00	1.0e+00	5.9e-16
bayer04	3.2e-15	1.0e+00	1.0e-00	2.1e-16
lhr34c	8.9e-04	9.9e-01	6.7e-01	9.5e-14
lhr71c	1.2e-01	2.1e-01	9.6e-01	1.7e-07
mark3jac120sc	1.0e+00	9.9e-16	1.0e+00	4.3e-16
mark3jac140sc	2.1e-15	6.8e-01	9.9e-01	4.2e-16
sinc18	5.5e-01	8.7e-01	8.8e-01	7.9e-01
sinc15	4.8e-15	9.7e-01	9.3e-01	9.0e-15
Zhao2	1.0e+00	1.0e+00	1.0e+00	1.0e+00
bbmat	9.2e-01	6.9e-15	9.7e-01	4.5e-16
Set 2				
rim	1.8e-05	9.7e-01	9.9e-01	9.0e-13
shyy161	4.6e-16	9.9e-01	1.0e+00	2.0e-16
e40r0100	4.8e-12	9.9e-01	1.0e-00	4.1e-16
af23560	4.6e-13	3.9e-16	8.2e-01	3.1e-16
3D_28984	1.0e+00	1.0e+00	9.9e-01	1.2e-14
3D_51448	1.9e-15	9.9e-01	1.0e+00	3.6e-16
ibm_matr	1.9e-15	1.0e-00	6.1e-01	2.7e-16
2D_54019	1.6e-05	7.2e-01	1.9e-01	3.6e-16
2D_27628	6.6e-16	1.0e-00	9.9e-01	2.9e-16

Table 4.7.6: SuperLU_DIST: component-wise backward error after iterative refi nement.

We now study the influence of the $mindrop$ threshold parameter introduced in Section 4.6.2. We recall that this threshold value is designed to control the relevance of numerical information in C_0. We expect that if this parameter is large enough then the numerical stability will be improved but at the cost of less freedom to select pivots (reduction of the size of C). We mentioned in the previous section that MA41_UNS is

already stable with $mindrop = 0.1$. We thus focus on SuperLU_DIST. and consider $mindrop$ values of 0.1, 0.9 and 0.99.

	Number of entries in the factors				Backward error			
$mindrop =$	0.1	0.9	0.99	DMLS	0.1	0.9	0.99	DMLS
Set 1								
av41092	6769	6795	6817	8098	1.0e+00	1.0e+00	1.0e+00	1.0e+00
g7jac140sc	12062	13054	12971	15204	1.0e+00	1.0e+00	8.4e-16	6.4e-16
g7jac120sc	10393	9863	9804	14486	1.0e+00	1.0e+00	1.9e-14	5.9e-16
bayer04	475	482	487	538	1.0e+00	1.0e-00	1.0e-12	2.1e-16
lhr34c	5770	5803	6172	5787	9.9e-01	5.1e-03	3.8e-02	9.5e-14
lhr71c	11966	11790	11871	10847	2.1e-01	3.5e-03	7.3e-01	1.7e-07
mark3jac120sc	10941	10722	11324	10828	9.9e-16	6.0e-01	1.6e-15	4.3e-16
mark3jac140sc	13328	12906	12567	12867	6.8e-01	4.1e-16	1.0e+00	4.2e-16
sinc18	26162	26589	29100	35287	8.7e-01	2.7e-14	3.5e-13	7.9e-01
sinc15	12732	13863	13102	15284	9.7e-01	6.7e-01	5.2e-14	9.0e-15
Zhao2	9777	9932	9730	12226	1.0e+00	1.0e+00	1.0e+00	1.0e+00
bbmat	34813	36272	37426	41315	6.9e-15	4.7e-16	5.4e-16	4.5e-16
Avg. gain	19.7%	18.9%	18.0%					
Set 2								
rim	5332	5227	5614	5283	9.7e-01	8.1e-01	1.8e-05	9.0e-13
shyy161	3198	3088	3080	3391	9.9e-01	2.7e-16	4.6e-16	2.0e-16
e40r0100	1738	1695	1738	2028	9.9e-01	9.9e-01	4.8e-12	4.1e-16
af23560	10889	10798	11020	10961	3.9e-16	2.8e-16	4.6e-13	3.1e-16
3D_28984	13817	12261	12165	12025	1.0e+00	1.2e-07	1.0e+00	1.2e-14
3D_51448	38744	32925	32890	32448	9.9e-01	2.9e-16	1.9e-15	3.6e-16
ibm_matr	37476	33569	32551	32820	1.0e-00	3.0e-16	1.9e-15	2.7e-16
2D_54019	8315	7634	7632	7591	7.2e-01	2.7e-16	1.6e-05	3.6e-16
2D_27628	3445	3043	3043	2996	1.0e-00	2.1e-15	2.9e-16	2.9e-16
Avg. gain	−0.7%	2.1%	0.9%					

Table 4.7.7: SuperLU_DIST: number of entries in the factors and precision of the solution after iterative refinement with different value of $mindrop$. Number of entries in thousands. Component-wise backward error given after iterative refinement.

When increasing $mindrop$, the initial number of entries in C is smaller so that we have less freedom to select the pivots. In fact CMLS is not very sensitive to this restriction and even a fairly limited number of additional entries with respect to MC64 matching is sufficient. Indeed, the sparsity gain in the SuperLU_DIST factors decreases from 19.7% to only 18.9% and 18.0% for $mindrop = 0.9$, and 0.99, respectively (see Table 4.7.7).

We see also that increasing $mindrop$ improves the robustness of SuperLU_DIST static pivoting, especially when $mindrop = 0.99$. For example, we succeed in obtaining a good backward error on the g7jac*sc, bayer04 and sinc* matrices. Unfortunately, even with $mindrop = 0.99$ SuperLU_DIST is not as stable as with the DMLS+MC64 ordering.

Table 4.7.8 shows that reducing the size of $\mathbf{C}$ slightly decreases the ordering time (see for example, the g7jac*sc, lhr*, bbmat and ibm_matr matrices with $mindrop = 0.99$).

Matrix / $mindrop =$	0.1	0.9	0.99	DMLS
Set 1				
av41092	10.4	10.0	10.1	3.9
g7jac140sc	27.7	29.3	17.6	6.2
lhr34c	8.7	8.2	6.6	3.4
mark3jac140sc	13.3	12.3	12.8	4.4
sinc18	27.3	28.2	27.6	19.7
bbmat	57.5	49.9	43.7	17.5
Avg. gain	-62.9%	-60.8%	-57.9%	
Set 2				
onetone2	0.8	0.8	0.7	0.3
circuit_4	60.1	55.4	57.0	15.0
ibm_matr	8.3	4.7	4.9	0.6
Avg. gain	-76.9%	-69.6%	-68.3%	

Table 4.7.8: Ordering time (in seconds).

4.7.2.5 Comments on structural strategies

Before considering hybrid strategies let us summarize the properties of the structural strategies:

- CMLS is always slower than DMLS and quasi-dense variables detection should be implemented in CMLS to reduce the time difference.
- The exclusive use of MATCHUPDATE may be dangerous; TOTALUPDATE might be too costly. We can limit the number of calls to TOTALUPDATE (LIM strategy) and reduce the cost of ordering while still preserving the sparsity of the factors. So the LIM strategy is our default structural strategy.
- MA41_UNS factorization is numerically stable even with $mindrop = 0.1$. Significant gains in terms of fill-in (22%) and flops (45%) have been obtained.
- SuperLU_DIST is more sensitive to $mindrop$ than MA41_UNS and still has numerical problems with $mindrop = 0.99$. That is why we need hybrid approaches that are presented in the next section.

4.7.3 Hybrid strategies with MC64 based preprocessing

Based on our previous study, only SuperLU_DIST can expect benefits from a hybrid strategy. In the hybrid strategy, a relative threshold is set to avoid the selection of small pivots in $\mathbf{C}$. After testing different values, we fixed the relative threshold to 0.01. We have also observed that increasing the relative threshold too much (> 0.1) does not improve the numerical behaviour of SuperLU_DIST very much and degrades the structural property of our algorithm.

In this section, we run CMLS with different values for $mindrop$ and give results for $mindrop = 0.1$, 0.9 and 0.99. An important parameter which defines the set on which the pivot search is applied is the parameter $NCOL$ defined in Section 4.6.3. $NCOL$ defines the maximum number of columns that can be accessed to find a good pivot. We

test the particular case of $NCOL = 0$ and the more typical case of $NCOL = 10$ for each $mindrop$ value.

	CMLS							
$mindrop =$	0.1		0.9		0.99		0.1	
Matrix/ $NCOL$ =	0	10	0	10	0	10	STR	DMLS
Set 1								
g7jac140sc	12216	13949	13207	13790	12990	13642	12062	15204
bayer04	482	469	462	486	489	477	475	538
lhr34c	6238	5764	7208	5730	5621	5902	5770	5787
lhr71c	13617	12452	12188	12193	12752	12415	11966	10847
mark3jac140sc	13656	13073	13174	12621	13245	13256	13328	12867
sinc18	28362	26518	26397	26813	28072	30447	26162	35287
Zhao2	10403	11039	10497	10679	10453	11051	9777	12226
bbmat	35145	35991	37350	38658	34230	34855	34813	41315
Avg. gain	12.3%	12.4%	13.5%	14.2%	14.6%	13.6%	19.7%	
Set 2								
Avg. gain	−2.1%	−1.7%	0.3%	0.2%	0.1%	0.1%	−0.7%	

Table 4.7.9: Number of nonzeros in the SuperLU_DIST factors (in thousands of reals). STR: structural strategy, otherwise the hybrid strategy used.

Table 4.7.9 compares the fill-in of hybrid approaches, of a structural approach (column STR of the tables) and of the DMLS+MC64 approach. The hybrid strategy prevents us from taking bad numerical pivots but does not always select the best structural pivot. This may explain why the average gain in sparsity on Set 1 decreases when we use a hybrid strategy. But this trend is not confirmed when $mindrop$ is increased and the best structural gains are obtained with $mindrop = 0.9$ and 0.99. Sometimes numerical information leads to a better global structural decision than a local structural decision. Moreover, we can see that increasing $NCOL$ to 10 does not always increase the number of nonzeros in the factors.

As one can see in Table 4.7.10 and in Figure 4.7.1, the numerical behaviour of SuperLU_DIST is clearly improved on Set 1 by the hybrid strategy. Setting $NCOL$ to 10 systematically decreases the number of failures with respect to $NCOL = 0$. Finally we see that the hybrid approach with $mindrop = 0.99$ and $NCOL = 10$ is better than the DMLS+MC64 approach on Set 1. Its solution phase always converges except on the av41092 and Zhao2 matrices for which all approaches do not converge. We see that the CMLS ordering with $mindrop = 0.99$ and $NCOL = 10$ performs less iterative refinement steps for a comparable quality of solution (see for example, the lhr34c, sinc15 and bbmat matrices).

As was observed in Table 4.7.8, we see in Table 4.7.11 that reducing the freedom offered to CMLS only slightly decreases the ordering time. It is also interesting to notice that the hybrid strategies are not significantly more costly than the structural strategies (compare average gains in Table 4.7.8 and Table 4.7.11). Zhao2 is the only matrix which completely contradicts this trend. Actually after the MC64 scaling, 124731 of its 166453 entries have an absolute value equals to one. Then numerical cancellation occurs in $\mathbf{C}$ and the hybrid strategy has difficulty in selecting pivots. It spends more than 90% of its time in the pivot selection. Clearly, algorithmic improvements must be done to take into

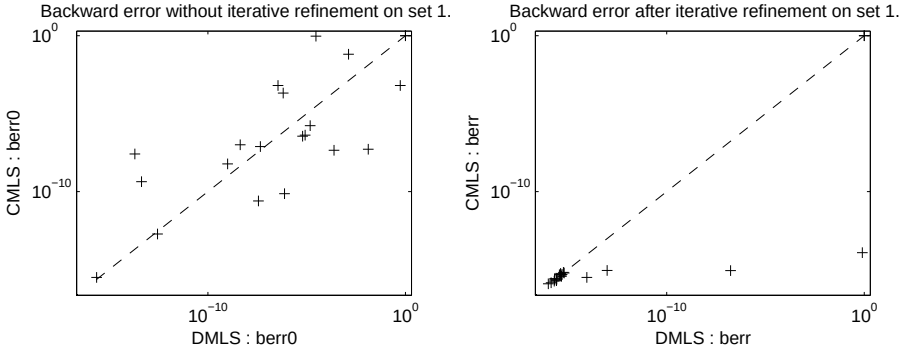


Figure 4.7.1: Component-wise backward error before and after iterative refi nement. $mindrop = 0.99$ and hybrid strategy used.

	CMLS							
$mindrop =$	0.1		0.9		0.99		0.1	
Matrix/ $NCOL =$	0	10	0	10	0	10	STR	DMLS
Set 1								
av41092	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}
g7jac140sc	9	7	5	4	4	4	2^{**}	3
jan99jac120sc	3	4	3	3	3	3	3	3
bayer04	3	3	3	3	3	4	2^{**}	4
lhr34c	5	6	10	6	5	6	2^{**}	9
lhr71c	6	7	6^{*}	5^{*}	9^{*}	5	2^{**}	3^{*}
mark3jac140sc	2^{**}	11	8	6	5	9	3^{**}	4
sinc18	2^{**}	2^{**}	5	5	4	6	2^{**}	2^{**}
sinc15	2^{**}	7	2^{**}	6	4	3	2^{**}	6
Zhao2	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}
bbmat	6	5	3	3	3	3	9	8
Set 2								
rim	5	4	4^{**}	3	5	5	2^{**}	5
shyy161	3	3	2	2	3	3	2^{**}	2
e40r0100	2^{**}	2^{**}	5^{**}	2^{**}	2^{**}	2^{**}	2^{**}	3
3D_28984	2^{**}	2^{**}	4	3	3	3	2^{**}	3
3D_51448	4	4	3	3	3	3	2^{**}	3
ibm_matr	2^{**}	4	3	3	3	3	2^{**}	3
2D_54019	5^{*}	2^{**}	5	4	4	4	2^{**}	3
2D_27628	8	4	3	3	3	3	2^{**}	3

Table 4.7.10: SuperLU_DIST: number of steps of iterative refi nement to get the precision of Table B.2.4. ** means that after iterative refi nement the component-wise backward error is greater than 10^{-4} and * means that after iterative refi nement the component-wise backward error is between 10^{-4} and 10^{-8}. STR: structural strategy, otherwise the hybrid strategy used.

	CMLS							
$mindrop =$	0.1		0.9		0.99		0.1	
Matrix/ $NCOL =$	0	10	0	10	0	10	STR	DMLS
Set 1								
av41092	10.1	10.7	10.6	10.3	10.9	10.1	10.4	3.8
g7jac140sc	36.2	56.1	50.5	36.7	18.2	18.6	28.9	5.7
bayer04	1.1	1.0	0.9	0.9	0.9	0.8	1.0	0.3
lhr34c	8.7	8.3	9.9	8.5	8.2	8.0	8.2	4.1
sinc15	17.8	19.1	12.6	13.3	11.2	11.9	16.3	3.7
Zhao2	17.6	57.2	12.0	56.3	14.6	125.6	3.5	1.0
bbmat	62.5	57.3	39.7	45.2	38.6	40.5	55.4	13.2
Avg. gain	-71.5%	-71.8%	-67.6%	-68.0%	-64.1%	-63.9%	-64.6%	
Set 2								
rim	2.1	2.1	1.6	1.7	1.2	1.2	1.9	0.4
shyy161	1.8	1.8	1.4	1.4	1.4	1.4	1.8	0.7
af23560	8.4	8.1	2.9	2.6	2.2	2.2	7.3	0.7
3D_51448	7.8	7.3	5.0	4.8	5.0	5.1	7.3	0.6
ibm_matr	7.6	7.4	5.0	5.0	5.0	5.0	8.0	0.6
2D_27628	1.1	1.1	0.5	0.6	0.6	0.6	1.0	0.1
Avg. gain	-81.7%	-81.8%	-74.4%	-74.2%	-72.6%	-72.9%	-80.8%	

Table 4.7.11: Ordering time (in seconds). STR: structural strategy, otherwise the hybrid strategy used.

account this kind of situation. Moreover the hybrid strategy is very sensitive to random permutations on this matrix: the ordering time varies from 17.5 seconds to 167.6 seconds with $mindrop = 0.99$ and $NCOL = 10$.

4.7.4 Influence of preprocessing phase

In the previous sections, all results were obtained with an MC64 based preprocessing. In order to avoid using MC64, we propose a way of computing an alternative preprocessing. We use the MC77 one-norm scaling [93] and the MC21 maximum matching [35, 36] to build $\mathbf{C}$ (see Algorithm 4.3.1): we first apply MC77 scaling and drop entries, MC21 is then used to compute a maximum matching and finally the entries of the matching not already in $\mathbf{C}$ are added. The MC77 scaling has the advantage of being easy to parallelize, its sequential complexity is bounded by $\mathcal{O}(it\_step\ nnz(A))$ where it_step is the number of iterations (in our experiments $it\_step = 20$). The MC21 algorithm only computes a structural maximum matching and is simpler than the MC64 one that needs numerical values of the original matrix. The coupling between MC77 and MC21 can certainly be enhanced but it is out of the scope of our study. This preprocessing is referred to as MC77_MC21. We analyse the performance of this preprocessing with the MA41_UNS solver.

MC77_MC21 improves the predicted size of the factors compared with MC64. It might come from the fact that the one-norm scaling tends to increase the numerical value of entries with low connectivity and to decrease the numerical value of entries with high connectivity. Thus, the $\mathbf{C}$ matrix has more chance to contain entries which are good potential pivots for numerical stability and for sparsity preservation. However, we also see in Table B.3.1 that, because of numerical pivoting, the real size of the factors increases

more with a structural strategy (STR) and the MC77_MC21 preprocessing than with the MC64 preprocessing. For example the size of the factors of lhr34c increases from 8574 predicted entries to 13629 during MA41_UNS factorization. Because of this, we also report results with a hybrid strategy with two thresholds combined with the MC77_MC21 preprocessing (the relative threshold is set to 0.01 and the absolute threshold is set to 10^{-6}). Table B.3.1 shows that the hybrid metric with two thresholds solves the estimation problem. The counterpart is that on some problems (the bbmat and mark3jac*sc matrices) it significantly increases the number of entries in the factors.

Matrix	estimated size				real size			
	MC21_MC77		MC64		MC21_MC77		MC64	
	STR	HYB	STR	DMLS	STR	HYB	STR	DMLS
Set 1								
g7jac140sc	12810	12734	16080	24138	13443 (1.05)	13113 (1.03)	16252	24146
lhr34c	8574	7268	8296	6979	13629 (1.59)	8288 (1.14)	8696	7122
lhr71c	16832	13758	16663	15450	24634 (1.46)	15888 (1.15)	17573	15824
sinc18	22968	21680	31581	41445	23785 (1.04)	22084 (1.02)	32334	41563
bbmat	42954	55530	39034	53596	48187 (1.12)	60927 (1.10)	39600	53641
Avg. gain	36.9%	22.7%	22.3%		27.4%	18.9%	22.0%	
Set 2								
Avg. gain	−4.5%	−5.5%	0.2%		−10.0%	−8.8%	−0.5%	

Table 4.7.12: MA41_UNS: number of entries in the factors with different preprocessings. Number of entries in thousands. STR: structural strategy. HYB: hybrid strategy. In parenthesis, the ratio between the real size and the estimated size of the factors.

Table 4.7.13 evaluates the gain in terms of number of operations. We see that CMLS combined with a structural strategy and a MC77_MC21 preprocessing is clearly better than the other approaches on Set 1. As already mentioned, it is not interesting to run CMLS on matrices of Set 2. Table 4.7.13 also shows large variations in the CMLS ordering time. For example, it increases for the bbmat and lhr* matrices and decreases for the av41092 and g7jac*sc matrices. The ordering time also increases to 90.2 seconds when the heuristic of the hybrid strategy increases the fill-in on the mark3jac120sc matrix.

We have also used MC77_MC21 preprocessing in a SuperLU_DIST context, but did not succeed in getting accurate solutions (see Table B.3.4). This will be discussed in our conclusion.

4.7.5 Influence on the performance of sparse solvers

In this section, we are concerned by the performance of the MA41_UNS solver (time and memory used by the factorization phase). We have thus activated the default amalgamation parameter to run our experiments.

We report the time required by the ordering, the factorization and the solution and the total time. Note that this total time may not be a good criterion to analyse performance. Firstly, because in this experimental study we decided to run the two solvers in sequential mode, but if we had run them in parallel, then the factorization time and the solution time would have been smaller. Secondly, because in a large range of applications the user

	number of operations				ordering time			
	MC21_MC77		MC64		MC21_MC77		MC64	
Matrix	STR	HYB	STR	DMLS	STR	HYB	STR	DMLS
Set 1								
av41092	1537	16321	2539	3664	8.2	12.0	10.4	3.9
g7jac140sc	11118	10708	13907	30254	20.2	23.1	27.7	6.2
lhr71c	12877	6356	5503	5281	27.4	23.9	19.0	7.9
mark3jac140sc	9429	68426	10618	9295	20.3	75.8	13.3	4.4
sinc18	40850	38391	64041	96970	24.8	35.2	27.3	19.7
Zhao2	4884	5707	6471	10080	3.3	4.6	3.4	0.9
bbmat	56950	104520	39553	77900	90.2	80.9	57.5	17.5
Avg. gain	54.1%	31.7%	45.3%		−66.4%	−73.2%	−61.6%	
Set 2								
Avg. gain	−16.5%	−16.5%	0.8%		−76.5%	−78.7%	−75.2%	

Table 4.7.13: MA41_UNS: number of operations (in millions) and ordering time (in seconds) with different preprocessings. STR: structural strategy. HYB: hybrid strategy.

may perform several factorizations of matrices which have the same pattern and similar numerical values and/or call the solution phase with multiple right-hand sides.

We see in Table 4.7.14 that the gains in terms of factorization time are slightly smaller than the gains in terms of number of operations (in Table 4.7.3 the average decrease in the number of operations is around 45% on Set 1). For example, we observed that the flops rate of MA41_UNS tends to be slowest with CMLS than with DMLS: the average flops rate is nearly 555 MFlops with the DMLS ordering whereas it is around 510 MFlops with the CMLS ordering.

We also note that on matrices that are relatively small, the factorization is too fast to compensate for the extra time spent in the analysis. On the 11 larger matrices of Table 4.7.14 CMLS does not succeed in decreasing the fill-in on the lhr* and mark3jac*sc matrices. On the other large matrices, we get a better execution time with the CMLS based approaches. Note that the time for preprocessing is not reported in our tables. We finally compare in Table 4.7.15, the memory used to perform the factorization. We see that all CMLS approaches often improve the memory used and that with the MC77_MC21 preprocessing the average reduction is around 28%.

4.8 Concluding remarks

CMLS is an ordering whose originality is to compute simultaneously a row and a column permutation with the following goals in mind: to reduce the fill-in in the factors and to preselect numerically good pivots for the factorization. It is based on a constraint matrix which contains the entries that can be selected and a quotient graph that is used to compute structural information. After having detailed our algorithmic choices, we identified its main characteristics:

- It generates factors which are sparser than the combination of DMLS and MC64. It often involves faster factorizations.

	ordering time			factorization time		
	CMLS with		DMLS	CMLS with		DMLS
Matrix	MC21_MC77	MC64		MC21_MC77	MC64	
av41092	8.3	10.3	**3.7**	**3.0**	4.4	6.7
g7jac140sc	20.1	31.2	**5.2**	**17.4**	24.0	39.0
g7jac120sc	16.2	23.2	**4.0**	**13.9**	20.7	29.9
lhr34c	10.2	8.4	**3.6**	18.3	7.8	**3.8**
lhr71c	23.7	20.8	**8.0**	39.0	15.5	**9.1**
mark3jac120sc	17.0	10.6	**3.4**	12.7	12.0	**10.1**
mark3jac140sc	21.9	12.9	**4.3**	16.0	13.9	**11.4**
sinc18	25.0	37.7	**14.7**	**62.2**	91.2	146.4
sinc15	10.6	13.2	**3.8**	**20.2**	34.7	38.8
Zhao2	3.4	3.4	**0.9**	**7.6**	11.4	15.7
bbmat	91.0	67.9	**17.5**	169.0	**67.4**	144.1
Avg. gain	−70.0%	−69.8%		35.1%	20.5%	
	solution time			total time		
	CMLS with		DMLS	CMLS with		DMLS
	MC21_MC77	MC64		MC21_MC77	MC64	
av41092	**0.7**	0.8	0.9	8.3	10.3	**3.7**
g7jac140sc	**1.0**	1.4	2.4	**38.6**	56.7	46.7
g7jac120sc	**0.9**	1.2	1.4	**31.2**	45.2	35.4
lhr34c	1.3	0.8	**0.7**	29.9	17.1	**8.1**
lhr71c	4.5	2.0	**1.7**	67.3	38.4	**18.9**
mark3jac120sc	1.7	1.3	**1.2**	31.5	23.9	**14.8**
mark3jac140sc	**1.5**	1.8	2.0	39.5	28.7	**17.7**
sinc18	**1.3**	1.8	2.2	**88.7**	130.7	163.4
sinc15	**0.7**	0.9	0.8	**31.6**	48.9	43.4
Zhao2	0.9	**0.7**	0.9	**12.0**	15.6	17.6
bbmat	4.3	**2.6**	3.5	264.4	**138.0**	165.1
Avg. gain	14.0%	12.7%		−8.8%	−17.2%	

Table 4.7.14: MA41_UNS ordering, factorization, solution and total time (in seconds).

	CMLS		
Matrix	MC21_MC77	MC64	DMLS
Set 1			
av41092	**6594**	8891	10662
g7jac140sc	**14245**	19850	27537
g7jac120sc	**13396**	17098	20669
lhr34c	10800	10014	**7967**
lhr71c	27067	20279	**16657**
mark3jac120sc	**14481**	15722	**14599**
mark3jac140sc	**17592**	**17740**	**17588**
sinc18	**29831**	42553	51498
sinc15	**13441**	20560	21059
Zhao2	**10422**	12641	15344
bbmat	57290	**40030**	53132
Avg. gain	28.6%	10.1%	

Table 4.7.15: MA41_UNS memory used (in thousands of reals).

- The larger and the more structurally unsymmetric are our matrices, the more we improve the total execution time of MA41_UNS.
- CMLS can be parametrized to perform a stable factorization in a MA41_UNS framework using a structural metric. In a SuperLU_DIST framework using a hybrid metric, we obtain an accurate solution with SuperLU_DIST on matrices for which DMLS+MC64 did not converge.
- It enables cheaper preprocessings than MC64 when combined with a solver such as MA41_UNS which performs numerical pivoting. This point has to be quantified in our future work.
- It is slower than DMLS. We will discuss future improvements at the end of the conclusions.
- If $\mathbf{C}$ is restricted to the entries of the MC64 matching, then CMLS benefits from some algorithmic choices that could be implemented in DMLS.

This last point emphasizes that DMLS could also benefit from our new aggressive absorption, our approach to scaled metrics in order to avoid tie-breaking and our supervariables which improve the metric computation. These changes may be done without extra cost and could even decrease the ordering time. Let us illustrate this on a small example. Let i and j be two indices, let r_i and r_j be the corresponding rows and let c_i and c_j be the corresponding columns. If r_i and r_j have the same pattern and c_i and c_j have different patterns then DMLS will not recognize any supervariables. During the update of the quotient graph and the metric computation it will access the four structures. This represents a complexity of $\mathcal{O}(|\mathcal{R}_i| + |\mathcal{R}_j| + |\mathcal{C}_i| + |\mathcal{C}_j| + |\mathcal{A}_{i*}| + |\mathcal{A}_{j*}| + |\mathcal{A}_{*i}| + |\mathcal{A}_{*j}|)$. If DMLS had decorrelated row and column supervariables, it would have recognized a row supervariable. During the update of the quotient graph and the computation of the metric it would access one row supervariable and the structure of the two columns. So the complexity would decrease to $\mathcal{O}(|\mathcal{R}_i| + |\mathcal{C}_i| + |\mathcal{C}_j| + |\mathcal{A}_{i*}| + |\mathcal{A}_{*i}| + |\mathcal{A}_{*j}|)$.

We have the following perspectives to improve CMLS. We want to improve the MC77_MC21 preprocessing to make it robust in a SuperLU_DIST context. Indeed doubly stochastic scalings have not been precisely studied in our context. In particular, such a preprocessing could be interesting in the case of non-trivial reducible matrices (like the mult_dcop_* matrices) since the doubly stochastic limit will have zeros instead of the entries outside the diagonal blocks. If the iterative process of MC77 is close enough to convergence, these entries will not appear in $\mathbf{C}$. Unfortunately, the convergence of these entries is often very slow. That is why one of our targets is to accelerate the doubly stochastic scaling. Moreover we think that tie-breaking limits MC21 performance and using information from MC77 scaling could accelerate the maximum matching algorithm.

We also noticed that CMLS is significantly slower than DMLS. As mentioned before, this gap can be reduced by quasi-dense variable detection that should benefit CMLS more than DMLS. We could also consider removing from $\mathbf{C}$ entries that are either numerically too small or that have a too large structural metric. Obviously removed variables must not belong to the matching.

The constraint matrix $\mathbf{C}$ contains information that can be seen as an incomplete factorization. It is in our intention to use it as a preconditioner and to compare its quality and cost with existing incomplete LU factorizations.

III

Chapter 5

Adapting a parallel sparse direct solver to architectures with clusters of SMPs [1]

We consider the direct solution of general sparse linear systems based on a multifrontal method. The approach combines partial static scheduling of the task dependency graph during the symbolic factorization and distributed dynamic scheduling during the numerical factorization to balance the work among the processes of a distributed memory computer. We show that to address clusters of SMP (Symmetric Multi-Processor) architectures, and more generally non-uniform memory access multiprocessors, our algorithms for both the static and the dynamic scheduling need to be revisited to take account of the non-uniform cost of communication. The performance analysis on an IBM SP3 with 16 processors per SMP node and up to 128 processors shows that we can signifi cantly reduce both the amount of inter-node communication and the solution time.

5.1 Introduction

We consider the direct solution of large sparse systems of linear equations $Ax = b$ on distributed memory parallel computers using multifrontal Gaussian elimination.

For an unsymmetric matrix, we compute its LU factorization; if the matrix is symmetric, its LDL^T factorization is computed. Because of numerical stability, pivoting is required in these cases in contrast to symmetric positive definite sparse systems where pivoting can be avoided.

The multifrontal method was initially developed for indefinite sparse symmetric linear systems [44] and was then extended to unsymmetric matrices [45]. It belongs to the class of approaches which separates the factorization into two phases. The symbolic factorization phase is not concerned with numerical values. It looks for a permutation of the matrix that will reduce the number of operations in the subsequent phase, and then computes an estimation of the dependency graph associated with the factorization. Finally, in an implementation for parallel computers, this phase partially maps the graph onto the target multiprocessor computer. The numerical factorization phase computes the matrix factors. It exploits the partial mapping of the dependency graph and performs dynamic task creation and scheduling to balance the work performed on each process [8,

[1] The `MUMPS` package, Version 4.3, is available at `http://www.enseeiht.fr/apo/mumps`. Part of this research was supported by a grant NSF-INRIA number NSF-INT-0003274. CINES (Montpellier) has provided access to their computer resources. The work of the second author was supported by the EPSRC Grant GR/R45441.

7, 11]. The work in this chapter is based on the solver, MUMPS, a MUltifrontal Massively Parallel Solver [7]. For an overview of the multifrontal method we refer to [38, 44, 82], for the discussion of other direct approaches to [28, 65, 70].

Our previous work on MUMPS implicitly assumed that our target computer was a distributed memory computer with uniform memory access and uniform cost of communication. We show in this chapter the limitations of this approach and indicate how we can remedy these limitations. Our modifications of the algorithms will affect both the symbolic factorization and the numerical factorization phases. Our experiments on the IBM SP3 from CINES (Montpellier) with 16 processors per SMP node and up to 128 processors shows that we can significantly reduce both the amount of inter-node communication and the factorization time. We also show that even larger gains can be obtained on an IBM SP4.

In Section 5.2, we briefly describe the parallelism involved in MUMPS. In Section 5.3, we present our experimental environment. We discuss in detail the the dynamic scheduling and the static mapping and the effect of architecture on the related algorithms in Section 5.4. We discuss, in Section 5.5, how to mix MPI and OpenMP in our approaches. In the final section, we give some concluding remarks.

We remark that the algorithms presented in this chapter have been integrated into the new Version 4.3 of MUMPS.

5.2 Description of the parallelism involved in the sparse solver

In this section, we describe the tasks arising in the factorization phase of a multifrontal algorithm.

The so called *elimination tree* [44, 81] represents the order in which the matrix can be factorized, that is, in which the unknowns from the underlying linear system of equations can be eliminated. For a dense matrix, the elimination tree is a chain and defines a complete ordering of the eliminations. However, for a general sparse matrix, the definition yields only a *partial* ordering which allows some freedom for the order in which pivots can be eliminated. (This graph is in the most general case a forest, but we will assume in our discussions, for the sake of clarity, that it is a tree. That is the matrix is irreducible).

One central concept of the multifrontal approach [44] is to group (or *amalgamate*) columns with the same sparsity structure to create bigger *supervariables* or *supernodes* [44, 83] in order to make use of efficient dense matrix kernels. The amalgamated elimination tree is called the *assembly tree*.

The work associated with an individual node of the assembly tree corresponds to the factorization of a so called *frontal matrix*. Frontal matrices are always considered as dense matrices and we can make use of efficient BLAS kernels and avoid indirect addressing, see for example [32]. Frontal matrices can be partitioned as shown in Figure 5.2.1.

Here, pivots can be chosen only from within the block of fully summed variables F_{11}. Once all eliminations have been performed, the Schur complement matrix $F_{22} - F_{21}F_{11}^{-1}F_{12}$ is computed and used to update later rows and columns of the overall matrix

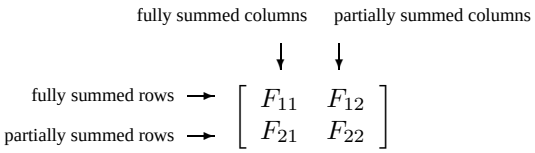
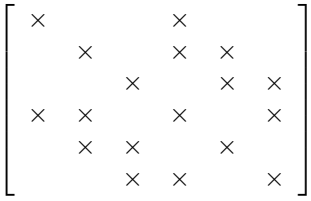


Figure 5.2.1: A frontal matrix.

which are associated with the parent nodes. We call this Schur complement matrix the *contribution block* of the node.

The notion of child nodes which send their contribution blocks to their parents leads to the following interpretation of the factorization process. This is illustrated in Figure 5.2.2 where the tree is processed from the leaf nodes to the root node. When a node of the assembly tree is being processed, it assembles the contribution blocks from all its child nodes into its frontal matrix. Afterward, the pivotal variables from the fully summed block are eliminated and the contribution block computed. The contribution block is then sent to the parent node to be assembled once all children of the parent (which are the siblings of the current node) have been processed. If some variables are not eliminated because of numerical issues, they are delayed and sent to the parent node.



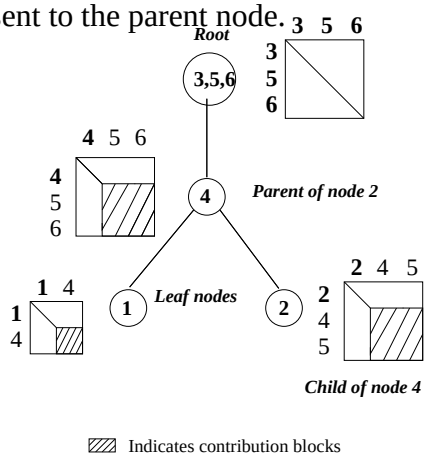


Figure 5.2.2: Matrix A and corresponding assembly tree. Fully summed variables (in boldface) are eliminated. The contribution blocks of the children (below) are assembled by the parent node (above).

A pair of nodes of the assembly tree where neither is an ancestor of the other can be factorized independently from each other, in any order or in parallel. Consequently, independent branches of the assembly tree can be processed in parallel, and we refer to this as *tree parallelism* or *type 1 parallelism*. It is obvious that, in general, tree parallelism can be exploited more efficiently in the lower part of the assembly tree than near the root node. Additional parallelism is then created using distributed memory versions of blocked algorithms to factorize the frontal matrices (see, for example, [8, 32]).

The contribution block is partitioned and each part of it is assigned to a different process. The so called *master process* is responsible for the factorization of the block of fully summed variables and will also decide (only during the numerical phase) how many and which processes (the so called *slave processes*) will be involved in the parallel activity associated with this node. We refer to this approach as *type 2 parallelism* and call the nodes concerned *type 2 nodes* (see Figure 5.2.3). Of course, if the node is not large enough, it will not be split and will be a type 1 node. Finally, the factorization of the dense root node can be treated in parallel with ScaLAPACK [23]. The root node is partitioned

and distributed to the processes using a 2D block cyclic distribution. This is referred to as *type 3 parallelism* (see Figure 5.2.3).

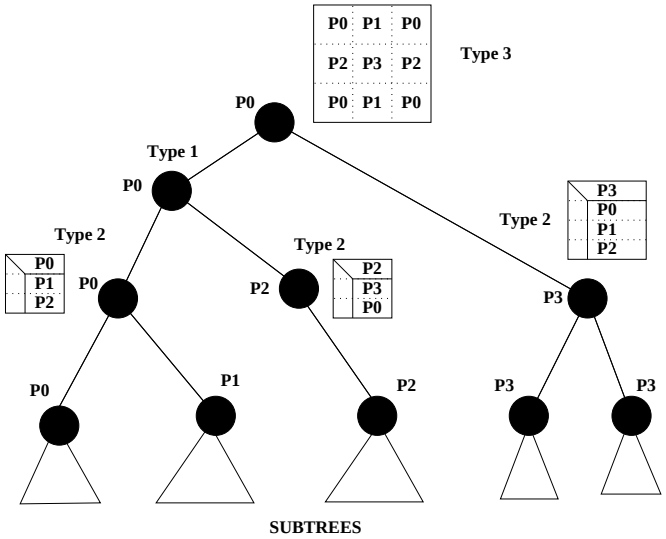


Figure 5.2.3: Different types of parallelism in the assembly tree.

5.2.1 Partial task mapping during the symbolic factorization phase

The selection of slaves for type 2 nodes during the factorization phase is an attempt to detect and adjust a possible imbalance of the workload between the processes at run time. However, it is necessary to carefully control the freedom given to dynamic scheduling (see [11] for a detailed analysis). Our sparse solver, MUMPS, addresses these issues by using the concept of *candidate processes*. Each type 2 node is associated, during the symbolic factorization phase, with a limited set of processes from which the slaves can be selected during numerical factorization. The candidate concept can be thought of as an intermediate step between full static and full dynamic scheduling. While we leave some freedom for dynamic decisions at run time, this is directed by static decisions on the candidate assignment.

The assignment and the choice of the candidate processes is guided by a relaxed proportional mapping (see Pothen and Sun [91]), slightly modified in [11] (shown here as step (2) of Algorithm 5.2.1). It consists of a recursive assignment of processes to subtrees according to their associated computational work. The assembly tree is processed from top down, starting with the root node. For each child of the root node, we calculate the work associated with the factorization of all nodes in its subtree, and the available processes are distributed cyclically among the children of the root node according to their weight. Each node thus gets its set of so called *preferential* processes which guide the selection of the candidate processes in a second bottom-up step (step (3) in Algorithm 5.2.1). The bottom-up mapping approach takes account of concurrency in the factorization and maps not only the master tasks of type 2 nodes but also chooses the candidates for slave tasks of type 2 nodes using the previously computed preferential processes. The main objective of this step is to balance the work between the processes.

Algorithm 5.2.1 *Task mapping during the symbolic factorization.*

(1) Given the assembly tree of a sparse matrix A
(2) **Top down** tree processing: Relaxed proportional mapping to select so called preferential processes
(3) **Bottom up** processing of the upper part of the tree: Decide type of parallelism, map the master node tasks, and choose the so called candidate processes for type 2 slave tasks.

5.2.2 Dynamic task scheduling during the factorization phase

For a better balance of the actual computational work at run time, both the number and the choice of the slaves of type 2 nodes are *dynamically* determined during factorization in the following way (see [8, 7] for further details). When the master of a type 2 node receives the symbolic information on the structure of the contribution blocks of the children, the slaves that will be involved in the factorization are selected based on their current work load, the least loaded processes being chosen from among the candidate processes of the node. The master then informs the processes handling the child nodes which slaves are participating in the factorization of the node so that they can send the entries in their contribution blocks directly to the appropriate slaves. The load of a process is defined here as the total number of operations ready to be performed on this process. Each process is in charge of updating its load information and informing the other processes of any "significant variation" of its load.

5.2.3 Limitation of the approach

Firstly, one should notice that the dynamic choice of the type 2 slaves is limited to the less-loaded processes from the list of candidates whose selection does not take into account the non-uniform cost of communication on networks of Symmetric Multi-Processors (which we will be referred to as SMP architectures).

Thus, improving the mapping of the type 2 slaves can influence the overall performance of the factorization. Secondly, in [11] it is shown that often most of the work and memory is spent on type 2 nodes and that when the number of processes and/or when the size of the matrix increases, the relative number of operations spent in type 2 nodes increases. For example, on 64 processes, more than 75% of the work is in general spent on type 2 slave tasks. This implies that modifying the mapping of these tasks is critical for the performance on large matrices. These two observations give both the scope and the motivation for modifying the scheduling of type 2 tasks on SMP architectures.

5.3 Experimental environment

5.3.1 Target machine

Our main target machine is the IBM SP3 from CINES. It is composed of 29 SMP nodes of 16 processors (Power3+, 375 Mhz). Each node shares 16 GBytes of memory and is interconnected to the others by a Colony switch (1 GBytes/s). When not explicitly mentioned, the results are obtained on this IBM SP3 configuration. We will also provide results on Regatta SMP nodes with 32 Power4/1.3 Ghz processors with 64 GBytes of

memory per SMP node and the same network as above at the end of Section 5.4.2.1. This machine will be referred to as the IBM SP4. We show, in Figure 5.3.1, the differences in the bandwidth and latency when using point-to-point communications based on MPI_ISEND and MPI_RECV on the IBM SP3.

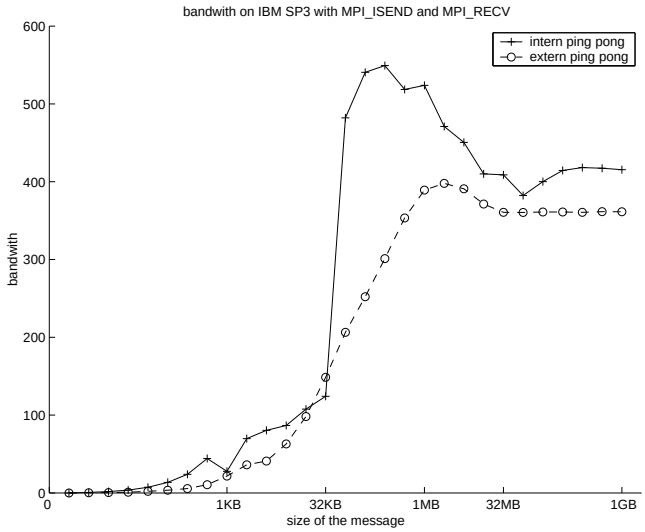


Figure 5.3.1: Influence of IBM SP3 architecture on point-to-point communication (ping-pong based on MPI_ISEND/RECV).

We observe a difference of 476 μs of latency time (64 μs of latency for intra node communications and 540 μs for inter node) and a variation of bandwidth of 288 MBytes/s in the worst case (depending on the message size). Furthermore, we note that these estimations can be even worse when the network is congested during communication. For instance, if 16 processes on the same node communicate simultaneously with 16 processes on another node (worst case on our IBM SP3 target machine), the effective bandwidth will be very poor.

For our sparse solver, the communication pattern is very irregular as it is driven by dynamic decisions and often involves one-to-many communications. Moreover, messages from a master of a type 2 node to all its slave processes are of a size between 64 KBytes and few MBytes. Therefore, from Figure 5.3.1, we are clearly in the area where there is quite a difference in the bandwidth between internal and external node communications.

5.3.2 Test problems

The matrices described in this section all arise from industrial applications and include test matrices from the PARASOL Project [2], the Rutherford-Boeing Collection [39] and the University of Florida sparse matrix collection [25].

In Table 5.3.1, we describe the characteristics of these test matrices arising from real life problems.

We also consider, as in [9], a set of test matrices obtained from an 11-point discretization of the Laplacian on 3D grids of either cubic or rectangular shape. The grid sizes

Matrix name	Order	No. of entries	Origin
PRE2	659033	5959282	Rutherford-Boeing (circuit sim)
G7JAC	59310	837936	Jacobian from CEPII's
XENON2	157464	3866688	ronis@onsager.chem.mcgill.ca (crystals)
SHIP_003	121728	8086034	Ship structure from production (PARASOL)
BMWCRA_1	148770	5396386	Automotive crankshaft (PARASOL)
AUDIKW_1	943695	39297771	Automotive crankshaft (PARASOL)
INLINE_1	503712	18660027	Inline skater (PARASOL)

Table 5.3.1: Test matrices.

are given in Table 5.3.2. The set of problems is designed so that when the number of processes increases, the number of operations per process in the LU factorization stays approximately constant when employing a nested dissection ordering [53]. In the remainder of this chapter, and when not explicitly mentioned, the number of processes used for a rectangular/cubic matrix will then characterize the matrix according to Table 5.3.2.

Procs	rectangular						cubic			
	Name	NX	NY	NZ	order	nnz	Name	N	order	nnz
1	RECT120	120	30	15	54000	573360	CUB36	36	46656	495216
8	RECT168	168	42	21	148176	1589448	CUB51	51	132651	1422951
16	RECT184	184	46	23	194672	2092816	CUB57	57	185193	1991865
32	RECT208	208	52	26	281216	3031288	CUB64	64	262144	2826496
64	RECT224	224	56	28	351232	3791536	CUB72	72	373248	4033440
128	RECT248	248	62	31	476656	5154928	CUB80	80	512000	5542720

Table 5.3.2: 3D grid problems.

As we are concerned with understanding the behaviour of our algorithms on a large variety of different dependency graphs, we use a range of different matrix orderings for our experiments.

- `AMD` (Approximate Minimum Degree [4]) has the advantage of being fast and generating a good ordering for rectangular grids and irregular problems (in terms of minimizing the number of operations). On cubic grid problems, it involves larger frontal matrices than other orderings such as `SCOTCH`.
- `SCOTCH` ([89, 90]) is a hybrid ordering. It has the advantage of giving a tree with good parallelism. It computes permutations which result in fewer operations than `AMD` in all cases of the test set. However, it is more costly to compute than the `AMD` ordering.
- `AMF` (Approximate Minimum Fill [85, 92]) minimizes fill-in at each pivot selection based on local heuristics (similar to `AMD`). It often results in fewer operations than the other local heuristics but leads to trees with less parallelism.

Note that our default (and most efficient) orderings for the solver are `SCOTCH` and `AMD` and we will focus more on these since we clearly are more interested in obtaining gains on graphs resulting from these orderings.

5.4 Task scheduling on heterogeneous architectures

An efficient scheduling has to take into account the balancing of memory, floating-point operations, and communication. This last point can be very critical when working on heterogeneous network architectures (for instance clusters of Linux PCs). But common strategies of scheduling often do not take into account this heterogeneity.

In Figure 5.4.1, we show two groups of processes linked by a slow network. The proportional mapping of the tree shown on the right does not take this into account and is inefficient. If the processes 3,4,5 and 6 were assigned to the bigger subtree, the cost of communication would decrease, and we would have better scheduling.

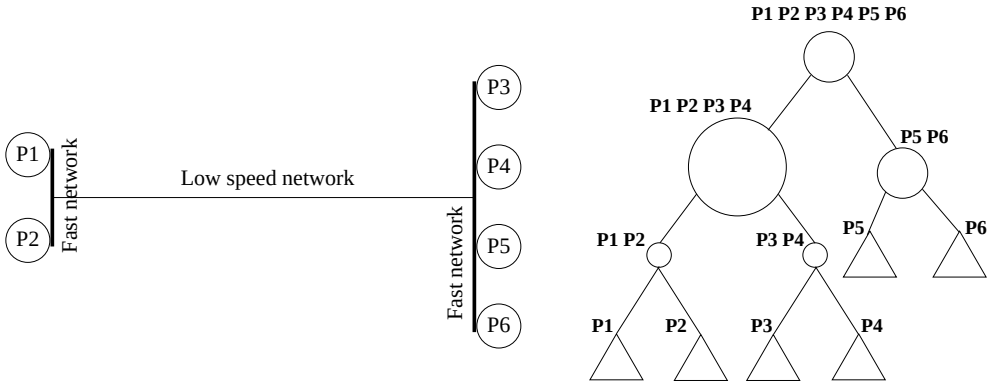


Figure 5.4.1: Example of a two-level architecture and of an ineffi cient mapping of the nodes of the tree onto the processors. (We assume here that the size of the nodes is proportional to the work in the subtree).

The main computer architecture that we target can be defined as a so called two-level architecture. Each level is composed of a set of identical processors sharing a common memory (that is, is an SMP node). This type of architecture includes quite a large set of commonly used computers such as the IBM SP3, HP-COMPAQ, and clusters of PCs. Two processors of the same SMP node are thus linked with a faster network than two processors of two different SMP nodes.

In this section, we will take into account architecture considerations to modify the dynamic and the static scheduling algorithms of our solver. We first describe architecture considerations in the fully dynamic version `v4.1` of the `MUMPS` solver (the master of a type 2 node has complete freedom to choose its slaves among all the other processes). Then, we will consider the impact of architecture considerations also on the choice of the candidate processes; this code will be referred to as `Cand`.

We have compared the factorization time on different configurations (4 processes on the same SMP node, 4 processes on 2 SMP nodes, 4 processes on 4 SMP nodes...). The largest variation was obtained when comparing 16 processes on the same node with 8 processes on two SMP nodes. Those results are reported in Table 5.4.1 in order to illustrate the potential of an algorithm which takes into account of the architecture of the machine, We notice variations of about 25 % in the factorization time. Indeed, in the rightmost column, we have the effect of network congestion in addition to the extra cost of external node communication

Matrix Name	Factorization Time (sec) 1 SMP node	2 SMP nodes
CUB51	73.7	91.3
CUB57	105.1	130.7
CUB64	226.8	282.4
RECT168	43.2	53.6
RECT208	132.5	166.2
PRE2	70.4	83.0

Table 5.4.1: Influence of architecture on the performance. LU factorization with 1×16 and 2×8 processes (`AMD` ordering and `v4.1` code).

5.4.1 Improving the fully dynamic solver (`v4.1`)

During the factorization, when a process becomes the master of a type 2 node, it examines the load (measured in flops) of the other processes. Based on this information, it decides both the number of slave tasks to create and the slave processes to which work will be assigned (see Section 5.2.2).

5.4.1.1 Modelling

We assume that each process has information on the load of the other processes. To take into account the distribution of the processes, we decided to penalize the load of the processes which are not on the SMP node of the master. If process j is the master of a type 2 task, it computes the penalized load, referred to as $load_{real}$ (in number of flops), of process number i. The dynamic scheduling then uses the penalized load function when determining the least-loaded processes.

Algorithm 5.4.2 *Computation of* $load_{real}(i)$ *by master* j.

if i is on my SMP node then
  if $load(i) < load(j)$ then $load_{real}(i) \leftarrow load(i) - load(j)$
  else $load_{real}(i) \leftarrow load(i)$
else $load_{real}(i) \leftarrow penalty(load(i))$

In all cases, we consider that a process which is on the same node and less loaded than the master is a **good potential slave** (its $load_{real}$ is strictly negative). Indeed, when the master sends to it the rows of the contribution block, it will be ready to treat them. On the other hand, since we ensure that $penalty(load(i))$ is positive, processes which are not on the SMP node of the master will be selected after the good potential slaves. In our experiments, we apply three basic rules for the penalty function, where Q is the number of bytes of information that has to be transmitted between the master and its slaves and λ, α and β are parameters that will be further discussed:

$$penalty(load(i)) = \lambda \times load(i) \ , \tag{5.4.1}$$

$$penalty(load(i)) = load(i) + \alpha \times Q + \beta \ , \tag{5.4.2}$$

$$penalty(load(i)) = \infty + load(i) \ . \tag{5.4.3}$$

Rule (5.4.1) penalizes the processes which are far away and loaded. It corresponds to a simple scaling of the loads. It prevents the selection of slaves which will not be ready to make a reception in the near future and thus reduces congestion of network and buffers. λ is a dimensionless parameter and should be linked to the size of the SMP nodes. Indeed, if we work with large SMP nodes, there is more danger of congestion in the network and λ has to be relatively large to prevent the selection of too many far away slaves.

Rule (5.4.2) corresponds to a modelling of the network. A good knowledge of the network is required to choose the values of α and β and to understand their physical meanings. The master of a type 2 node wants to select the slaves which will compute the fastest answer. We assume that each process has the same speed (v_{proc} in Mflops/s) on all nodes, that the bandwidth between processes on the same node is v_1 and between two different nodes is v_2 , that the latency time on the same node is $tlat_1$ and $tlat_2$ between different nodes, that communications and work do not overlap, and that the values of load are correct. Let k be a process on the SMP node of the master and l a process on another SMP node. Let T_1 be the time needed by k and T_2 the time needed by l to do its work if they are chosen as slaves. With these assumptions, we have:

$$T_1 = \frac{load(k)}{v_{proc}} + tlat_1 + \frac{Q}{v_1} + \frac{work\ as\ slave}{v_{proc}}$$

and

$$T_2 = \frac{load(l)}{v_{proc}} + tlat_2 + \frac{Q}{v_2} + \frac{work\ as\ slave}{v_{proc}}.$$

To make a good decision we have to compare these two values, which is equivalent to comparing $load_{real}(k)$ and $load_{real}(l)$ where:

$$load_{real}(k) = load(k)\ ,\ \text{and}\ \ load_{real}(l) = load(l) + \alpha Q + \beta\ ,$$

with

$$\alpha = v_{proc} \times (\frac{1}{v_2} - \frac{1}{v_1})\ ,\ \text{and}\ \ \beta = v_{proc} \times (tlat_2 - tlat_1)\ .$$

Rule (5.4.3) will never (if there is no constraint on the minimum number of processes that must be created on a given node) take, as a slave process, a process located on a different node from the master. When a minimum number of slaves is imposed (often because of memory/buffer constraints) then a sorting based on an initial load is still used (we have used the notation $\infty + load(i)$ for this purpose).

5.4.1.2 First experiments and tuning of the algorithms

Different values for α, β and λ in the previous formulae have been tested. The best results on the IBM SP3 were obtained for $\lambda = 4$, $\alpha = 1.5$ and $\beta = 50000$. Rule (5.4.3) provided the best results on matrices involving nodes with relatively large contribution blocks such as in the cubic grid problems. For this reason, we introduce a $threshold$ (of 3MBytes) in the strategies below. Rule (5.4.2) seemed to be more stable on irregular problems. Based on a first set of experiments, not reported here, we retained the following two strategies:

Strategy SCAL (scaling):
if $Q < threshold$ then
$penalty(load(i)) = \lambda \times load(i)$
else $penalty(load(i)) = 2 \times \lambda \times load(i)$

Strategy NETW (network modelling):
if $Q < threshold$ then
$penalty(load(i)) = load(i) + \alpha \times Q + \beta$
else $penalty(load(i)) = 2 \times (load(i) + \alpha \times Q + \beta)$

We first show, in Table 5.4.2, results obtained on 16 processes with our largest problems.

Matrices	SCOTCH				AMD			
	16x1	8x2			16x1	8x2		
	v4.1	v4.1	SCAL	NETW	v4.1	v4.1	SCAL	NETW
CUB64	92.8	121.7	124.2	116.5	226.8	282.4	286.9	294.6
RECT208	72.8	81.0	76.8	80.4	132.5	166.2	135.3	136.7

Table 5.4.2: Results with different distributions of 16 processors on MUMPS unsymmetric code. 16x1: 16 MPI processes on 1 SMP node. 8x2: 16 MPI processes distributed on 2 SMP nodes.

On RECT208 we succeeded in eliminating more than half of the penalty of the architectural distribution of the processes (see Table 5.4.2). This comes from the fact that there is enough tree parallelism, which enables us to decrease the number of slaves without affecting parallelism. We almost obtain the same factorization time on 16 processes on 1 SMP node and on 16 processes distributed on 2 SMP nodes.

We do not obtain gains on 2 SMP nodes and 16 processes (8 processes per nodes) for cubic grids (see Table 5.4.2). An explanation could be the physical limitation of locality (number of processes per SMP node). This limitation has more effect on cubic-grid problems and on small size SMP nodes. For instance, using AMD we obtain an assembly tree with large type 2 nodes and the master cannot keep the data on its SMP node because of constraints on the granularity of the task for slaves of type 2 nodes due to both memory and parallelism issues. We will see, however, in the following that, on more processors, there is still scope for interesting gains even on cubic matrices.

Matrices	SCOTCH			AMD		
	Ops($\times 10^9$)	v4.1	Arch	Ops($\times 10^9$)	v4.1	Arch
CUB64	744	107.8	81.0	1722	244.0	174.9
RECT208	511	81.0	62.3	657	142.7	123.7

Table 5.4.3: Influence of architectural decision on 32 processors (16 processes on each SMP node). LU factorization time in seconds.

Moreover, we want our algorithms to be as independent as possible from the low-level machine architecture characteristics. The NETW strategy aims to model the network. It turns out that it strongly depends on the target machine characteristics and potentially on the load on the machine. Because of this, in the remainder of this chapter, we will only consider the SCAL strategy. This will be referred to as the Arch code in the remainder of this chapter.

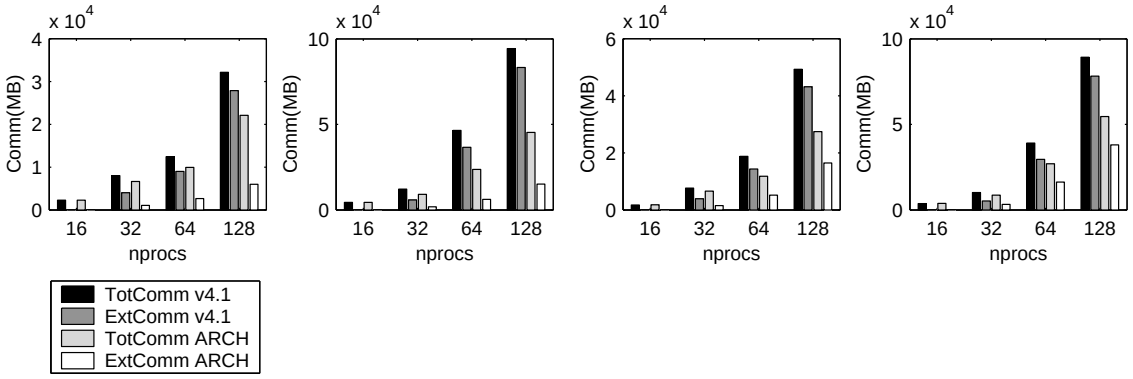


Figure 5.4.2: Influence on communication when using SCOTCH. From left to right: unsymmetric code on rectangular grids, unsymmetric code on cubic grids, symmetric code on rectangular grids, symmetric code on cubic grids. Problem sizes vary with number of processes as given in Table 5.3.2.

We obtain good speedups (between v4.1 and Arch codes) on two SMP nodes with 32 processes (see Table 5.4.3). This illustrates our argument about issues of congestion (as we deal with the worst possible case of 16 processes communicating with 16 processes of another SMP node). On our target architecture, we obtain gains of between $10\,\%$ and $40\,\%$ with an average gain of $25\,\%$. In Figure 5.4.2, we show the decrease in the volume of total communication (TotComm) and in the volume of external communication (ExtComm). Comparing Figures 5.4.2 and 5.4.3, we see that there is a link between the volume of external communication and factorization time for the LU factorization. Indeed, the architecture-aware algorithm prevents the masters from doing expensive communications, and hence the factorization time correspondingly decreases. In the case of the LDL^T factorization, the relative variation of the total amount of communication is not negligible compared to the relative variation of the external communication. Thus, the factorization time with the Arch version of the symmetric code might be influenced more equally by the combination of these two variations. Analysing the behaviour of v4.1 for an LDL^T factorization shows that it does not take much advantage of tree parallelism. For each type 2 task, the master tends to select most of the other processes as slaves. In the Arch version, the master will select fewer processes that are not on its SMP node. In this case, MUMPS will thus use the tree structure more to provide parallelism. That is why the volume of total communication decreases in the same way as the external communication.

Since our strategies penalize some processes, taking into account the SMP architecture during dynamic scheduling will lead to a reduction in the average number of slaves (see Figure 5.4.4). This corresponds to a decrease in the amount of tasks created for type 2 parallelism and is obviously most critical near the root where tree parallelism is limited. Our algorithms have, however, been designed to automatically maintain a minimum level of parallelism. Furthermore, memory constraints might also force the masters to work with slaves on other SMP nodes.

The threshold of 15 in Figure 5.4.4 corresponds, on the IBM SP3, to the maximum number of slaves that can be chosen on the SMP node of a master. The comparison of results obtained with the symmetric and the unsymmetric codes on cubic grids (see Figures 5.4.3

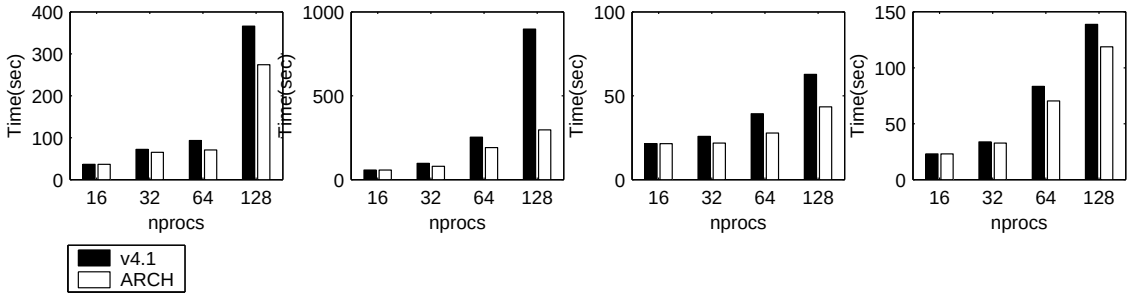


Figure 5.4.3: Factorization time when using SCOTCH. From left to right: unsymmetric code on rectangular grids, unsymmetric code on cubic grids, symmetric code on rectangular grids, symmetric code on cubic grids. Problem sizes vary with the number of processes as given in Table 5.3.2.

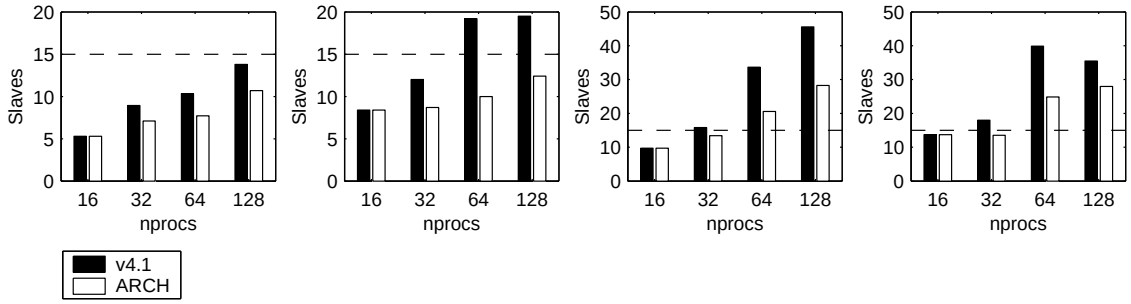


Figure 5.4.4: Influence on the average number of slaves of the type 2 node when using SCOTCH. From left to right: unsymmetric code on rectangular grids, unsymmetric code on cubic grids, symmetric code on rectangular grids, symmetric code on cubic grids. Problem sizes vary with the number of processes as given in Table 5.3.2.

and 5.4.4) illustrates the effect of different algorithm and memory constraints. On 128 processes with the symmetric code, our constraints prevent us from reducing the average number of slaves enough. We then do not obtain an improvement in factorization time comparable to that obtained for the unsymmetric code for which the parallel management of type 2 nodes (see [8]) introduces less constraints.

We have thus observed two limitations of our strategy of taking into account the SMP architecture:

- the physical configuration (number of processes per SMP node) prevents us from keeping the data on the same SMP node,
- too strong a penalty of processes which do not belong to the SMP node of the master of a type 2 task decreases type 2 parallelism too much and can affect the efficiency of the code.

In conclusion, we have thus succeeded in accelerating the factorization step of v4.1 by architectural considerations in our dynamic scheduling algorithm. In order to obtain further gains, it seems necessary to combine our modification of the dynamic scheduling with improvements to the candidate based algorithm with relaxed proportional partial mapping. On machines like the IBM SP3, we thus also guide the static mapping using architectural knowledge.

5.4.2 Influence of the architecture on candidate based algorithms

In this section, we describe how the architecture of the computer can be taken into account to influence the decision taken by the MUMPS version based on candidate processes (referred to as Cand in the following). In Section 5.4.1.2, we have described how the dynamic scheduling can be influenced by our algorithm and that work naturally extends to the candidate concept. We can limit the computation of $load_{real}$ of Algorithm 5.4.2 to the list of the candidate processes.

To facilitate the description of our algorithm we introduce the followings definitions and notations.

DEFINITION 4.1. *The SMP node of a process i will be noted $SMP(i)$. The size of an SMP node is the number of processes on it. For a process i, the size of its SMP node will be denoted by $|SMP(i)|$.*

DEFINITION 4.2. *The distance between two processes i and j is equal to 0 if i and j are on the same SMP node. We will denote this by $d_{SMP}(i,j)$. For multilevel architectures, we define $d_{SMP}(i,j)$ as a function of the average time to communicate between i and j. In our target architectures, we set $d_{SMP}(i,j) = 1$ when i and j are not on the same SMP node.*

DEFINITION 4.3. *A process i is said to be* **SMP predominant** *among a $list$ of processes if it minimizes $\sum_{j \in list} d_{SMP}(i,j)$.*

During the symbolic factorization, a partial mapping of the tasks is performed and we thus want our approach to merge information from the proportional mapping and from the architecture. During the static mapping a master and a list of so called candidate processes are associated with each type 2 node. We thus add two new controls:

- the candidates are chosen according to an architecture criterion.
- a master of a type 2 node must have enough candidates on its SMP node in order to keep communication as local as possible.

During the top-down phase of the task mapping (Algorithm 5.2.1), the subtrees are sorted layer-wise in decreasing order of size. Then the preferential processes are recursively assigned cyclically (according to the MPI process numbering in the communicator) to the subtrees. To take into account the architecture, we assign the preferential processes cyclically but, instead of using MPI process numbering, we use the positions of the processes in reordered lists. We only build the reordered list of processes for the root layer so that the distance between two neighbours is minimized and the processes are sorted in decreasing order of the size of their SMP node (see step (2') of Algorithm 5.4.3). In a more formal way, L is a list which satisfies the two following conditions:

- L minimizes $\sum_{i=1}^{nprocs} d_{SMP}(L(i), L(i+1))$,
- for any two processes i and j, if $|SMP(i)| > |SMP(j)|$, then i is sorted before j.

Algorithm 5.4.3 *Task mapping with architecture consideration during the symbolic factorization.*

(1) Given the assembly tree of a sparse matrix A
(2') **Top down**
-Build the reordered list L of processes to minimize the distance between neighbouring processes.
-Tree processing: relaxed proportional mapping to select preferential processes with cyclic assignment on L
(3') **Bottom up** processing of the upper part of the tree: decide type of parallelism, map the master node tasks using the *architecture criterion*, and choose candidate processes for type 2 slave tasks.

In the recursive implementation of the proportional mapping, we can expect more or less to keep these properties throughout the tree. Thus, we increase the probability of having lists of preferential processes with fewer SMP nodes represented than in the `Cand` version. We also expect to map the largest tasks onto a smaller number of SMP nodes, because the largest SMP nodes are sorted at the beginning of the list and for each layer, large tasks are mapped first (see [11]). For instance, on the example of Figure 5.4.1, we will have $L = [\ P_3\ P_4\ P_5\ P_6\ P_1\ P_2\ ]$. Then the strict proportional mapping, influenced by this reordered list, will affect the preferential list $L_1 = [\ P_3\ P_4\ P_5\ P_6\ ]$ for the biggest subtree and the list $L_2 = [\ P_1\ P_2\ ]$ for the other subtree, and so forth. For this example, we get a mapping that minimizes the communication costs.

During the bottom-up phase we guide the choice of the master processes using the preferential list. We add an **architecture criterion** in order to minimize the communication cost from the master to its slaves: a preferential process can be selected as a master of a type 2 node if and only if it is SMP predominant within the preferential list.

In the remainder of this chapter, we refer to the approach combining the use of Algorithm 5.4.2 and Algorithm 5.4.3 as `v4.2` since it is available in the new release 4.2 of the `MUMPS` package.

This thus gives us four versions of the code that we can summarize as follows.

- `v4.1` Original code that assumes all processes can be slaves of type 2 nodes.
- `Cand v4.1` but with choice for slaves restricted to set of candidate processors.
- `Arch` In the dynamic choice of slaves during factorization account is taken of locality (using modified load values).
- `v4.2 Arch` plus a modification of the candidate selection in analysis to encourage communication within SMP nodes.

5.4.2.1 Results

We have run `MUMPS` with `SCOTCH` and `AMF` orderings. In this section, we focus on results with the `SCOTCH` ordering since it is the most general purpose ordering that provides, in general, the best performance in terms of flop counts and timings on a large class of matrices. In Tables 5.4.4 and 5.4.5, we show results obtained on 64 processes (4 SMP nodes) and 128 processes (8 SMP nodes), respectively. We focus on the rectangular and

Matrices	Code	Ops	v4.1	Arch	Cand	v4.2
Rectangular	SYM	3.9D+11	39.3	27.7	27.4	27.1
	UNS	7.9D+11	93.2	70.5	76.6	68.6
Cubic	SYM	1.1D+12	83.2	70.4	72.6	73.8
	UNS	2.3D+12	252.7	191.0	229.8	204.2
G7JAC	UNS	1.6D+11	35.2	31.2	33.3	26.8
XENON2	UNS	1.1D+11	11.7	10.4	10.2	8.3
PRE2	UNS	1.7D+11	26.2	21.0	23.5	19.4
BMWCRA_1	SYM	6.4D+10	9.0	5.8	6.8	4.8
INLINE_1	SYM	1.3D+11	10.0	8.2	8.8	8.3
AUDIKW_1	SYM	5.4D+12	300.9	249.7	289.7	275.2
SHIP_003	SYM	9.2D+10	11.0	6.5	8.3	7.4

Table 5.4.4: Influence of architecture on 64 processors using the SCOTCH ordering. SYM: LDL^T factorization. UNS: LU factorization. Time in seconds.

Matrices	Code	Ops	v4.1	Arch	Cand	v4.2
Rectangular	SYM	7.1D+11	62.7	43.4	37.1	35.5
	UNS	1.4D+12	365.9	273.9	136.1	111.9
Cubic	SYM	2.3D+12	138.7	118.7	93.9	98.7
	UNS	4.7D+12	896.8	296.5	294.6	272.9
AUDIKW_1	SYM	5.4D+12	211.1	192.1	192.4	184.5

Table 5.4.5: Influence of architecture on 128 processors using SCOTCH ordering. SYM: LDL^T factorization. UNS: LU factorization. Time in seconds.

cubic grids and a large symmetric irregular problem, AUDIKW_1, because we want to keep reasonable granularity.

Our previous remarks on communications, architecture constraints and decrease of type 2 parallelism also apply to this set of results. For instance, during the factorization of PRE2 on 64 processors, the total amount of communication varies from 5.2 GB (v4.1) to 5.4 GB (v4.2) while the external communication decreases from 4.3 GB to 2.3 GB and the average number of slaves decreases from 14 to 10.5. It is important to note that the MPI numbering on the IBM SPs is the same as the architecture numbering. This property is, however, not defined in the MPI standard and will be much less likely to be satisfied on clusters of Linux based PCs. Even if, because of this property, our algorithm has little impact on our target machines during the top-down phase, it is important to apply it in a more general context, as was shown when running on the fully dynamic version of the code, not driven by a "good" candidate list of processes. This explains why our algorithms are less effective on the Cand code (see Tables 5.4.4 and 5.4.5).

Let t_j denote the time to execute a given job involving a total of ops_j floating-point operations on j parallel processes. Then, we define the **scaled speedup**, S_j, by

$$S_j = \frac{t_1/ops_1}{t_j/ops_j}.$$

In Figure 5.4.5, we show the scaled speedup for the regular matrices with SCOTCH

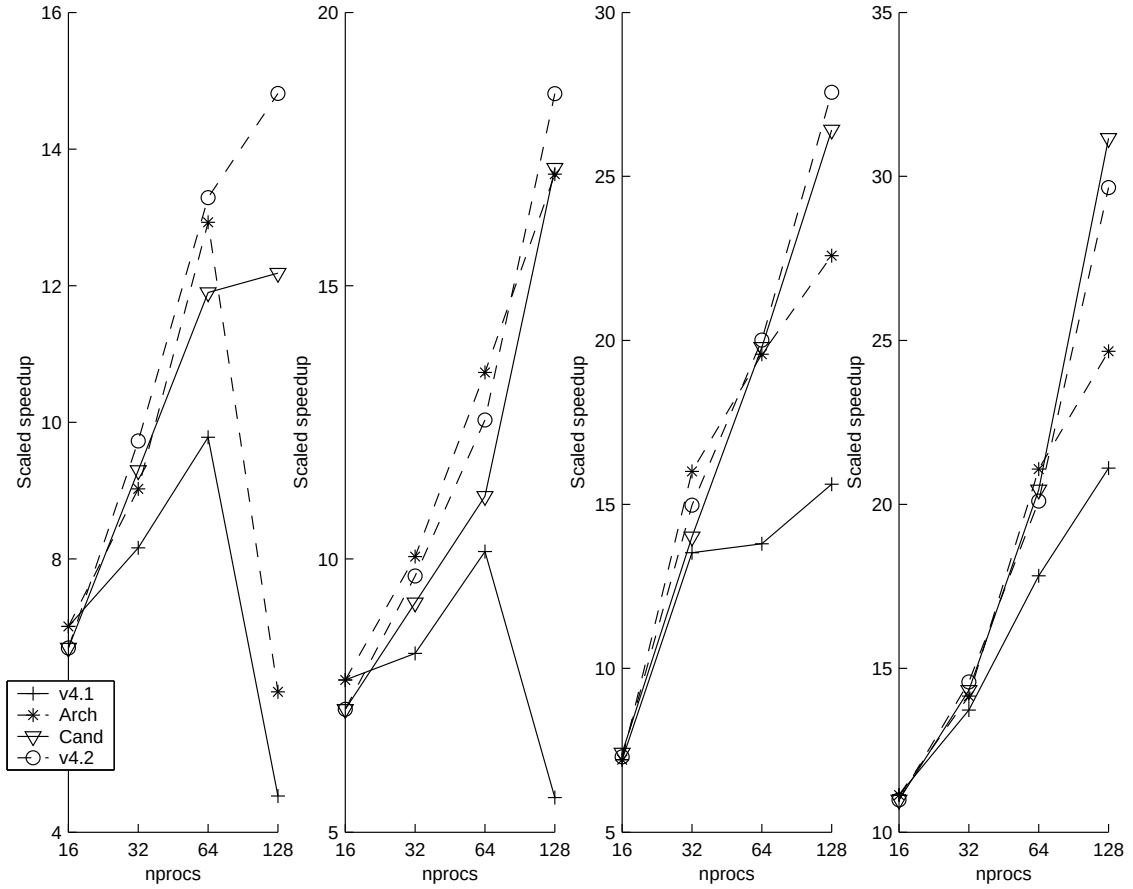


Figure 5.4.5: MUMPS scaled speedup with SCOTCH ordering. From left to right: unsymmetric code on rectangular grids, unsymmetric code on cubic grids, symmetric code on rectangular grids, symmetric code on cubic grids.

ordering. It shows that we obtain good improvements over `v4.1`. Moreover, the relative speedups between 16 and 32 processes, 32 and 64, 64 and 128 are also quite good. The average relative scaled speedup is nearly 1.5 when doubling the number of processes (between 16 and 32 processes, 32 and 64 etc ...). `v4.2` is clearly the most robust even in our case where the "natural" process numbering of MPI tasks in the communicator is already "good". We also gain in the stability of the performance of the code when including architecture considerations. Indeed when the architecture consideration is activated in the code, `MUMPS` becomes less sensitive to the partial static mapping. If we compare the gap $Area_1$ between the graphs for `v4.1` and of `Cand`, with the gap $Area_2$ between the graphs for `Arch` and `v4.2`, $Area_2$ is smaller than $Area_1$. In other words, the improvement shown between the graphs for `Arch` and `v4.2` is less than the improvement between `v4.1` and `Cand`.

Matrices	Code	v4.1	Arch	Cand	v4.2
Rectangular	SYM	46.8	18.5	27.1	21.0
	UNS	65.4	36.2	40.2	27.3
Cubic	SYM	131.5	56.0	63.7	52.8
	UNS	213.0	97.5	96.3	71.4
G7JAC	UNS	17.5	14.7	17.6	10.7
XENON2	UNS	8.1	5.7	9.2	5.5
PRE2	UNS	18.0	12.2	26.5	15.6
BMWCRA_1	SYM	12.9	5.6	4.1	2.8
INLINE_1	SYM	15.2	8.5	6.9	5.8
AUDIKW_1	SYM	359.6	180.1	214.0	209.0
SHIP_003	SYM	15.4	6.1	12.5	6.8

Table 5.4.6: Factorization time on IBM SP4 on 64 processors (2 SMP nodes) using `SCOTCH` ordering. SYM: LDL^T factorization. UNS: LU factorization.

Table 5.4.6 shows the `MUMPS` factorization time on 2 SP4 SMP nodes. We observe the same behaviour as on the IBM SP3. The gains between the different versions of `MUMPS` are better on this architecture because the congestion of the network is even more critical: 32 processors communicate with 32 processors on another node.

5.5 Hybrid parallelization

Another interesting possibility with SMP architectures is to use hybrid parallelization. Indeed on the same node, we can use a shared memory paradigm by means of the multithreaded BLAS library ([1], [30], [31]). Our first experiments on the IBM SP3 show that, on our application, the time to factorize on 2 MPI processes is comparable with that obtained for one MPI process with two OpenMP threads. Thus we can expect gains due to a decrease of the communication volume on a larger number of processes by halving the number of the MPI processes and taking two OpenMP threads for each MPI process. We first tried to evaluate which rate of OpenMP threads per MPI process would be best. As shown in Table 5.5.1 the performance of our solver can be improved by using 2 or 4 OpenMP threads per MPI process.

On a large number of processors, we see in Table 5.5.2 that significant performance gains

MPI process	OpenMP threads per MPI process	Factorization time (sec) AMD
16	1	37.6
8	2	29.8
4	4	31.1
2	8	41.1
1	16	52.2
16	2	77.5
8	4	70.9

Table 5.5.1: Hybrid parallelization on 1 node (16 processors) on a cubic grid (46x46x46) (`v4.1` used).

can be obtained by mixing OpenMP and MPI. One reason is that a 1D block partitioning is less scalable than a 2D block partitioning (see [94]). When we use OpenMP, each task (master or slave task) will be subdivided leading to a pseudo 2D block partitioning among OpenMP threads. Results with the AMF ordering illustrate this point well. When running the AMF ordering, we obtain a tree with very limited parallelism which can be approximated by a chain. Thus we often have a sequence of type 2 nodes for which the gain from this implicit 2D partitioning is even more critical. We develop this argument in the next paragraph.

Procs	MPI Tasks	OpenMP Threads / MPI	SCOTCH RECT Ops	SCOTCH RECT Facto Time	SCOTCH CUB Ops	SCOTCH CUB Facto Time	AMF RECT Ops	AMF RECT Facto Time	AMF CUB Ops	AMF CUB Facto Time
16	16	_	2.2	36.2	4.1	56.7	1.7	34.0	8.0	131.4
	8	2		**25.7**		**44.6**		40.4		101.7
	4	4		33.6		47.8		46.9		114.9
32	32	_	5.1	60.5	7.4	82.4	4.6	71.6	16.8	280.8
	16	2		46.5		**72.4**		68.9		205.2
	8	4		**46.4**		75.3		72.0		190.1
64	64	_	7.9	68.5	23.8	204.2	8.5	163.9	41.1	626.2
	32	2		61.8		173.7		130.2		422.5
	16	4		**52.0**		**145.4**		117.1		316.2

Table 5.5.2: Unsymmetric code (`v4.2`). RECT : Rectangular grid. CUB : Cubic grid. Ops : Total number of operations ($\times 10^{11}$). Time in seconds. Problem sizes as given in Table 5.3.2.

Let C be a chain of n nodes of type 2 ($C(1)$, ..., $C(n)$), and T_{m_i} be the time needed to compute the fully summed rows of $C(i)$ on 1 process (that is, the time taken by the master of $C(i)$ to do its work when we are not using OpenMP threads). Let T_{s_i} denote the time needed to compute the partially summed rows of $C(i)$ on 1 process (it is the time needed to compute the slave part of $C(i)$ if the master of $C(i)$ chooses only one slave and if we are not using OpenMP threads). Let N_{mpi} be the number of MPI processes, N_{omp} be the number of OpenMP threads per MPI process and $N_{procs} = N_{mpi} \times N_{omp}$ be the total number of processes calling BLAS routines. Finally, let T_{tot} denote the total time needed to treat the chain. If we assume perfect scalability and no communication

overheads, we then have

$$T_{tot} = \sum_{i=1}^{n} \max(\frac{T_{m_i}}{N_{omp}}, \frac{T_{s_i}}{N_{procs} - N_{omp}}).$$

If N_{omp} is small, T_{tot} tends to be determined by the terms T_{m_i}/N_{omp}, otherwise it tends to be determined by the terms $T_{s_i}/(N_{procs} - N_{omp})$. Thus, for a given chain there exists an optimum depending on N_{omp} (see Table 5.5.2). For example, we halve the factorization time on a cubic grid on 64 processors with 4 OpenMP threads per MPI process, whereas one OpenMP thread is best on rectangular grids with 16 processes. A safe compromise can be obtained with 2 OpenMP threads per MPI process.

To summarize the gains obtained and to give an idea of the parallelism obtained by our final code, we show, in Table 5.5.3, the scaled speedup with and without hybrid parallelism in Version 4.2 of our code. We first see that the gains due to OpenMP on a single SMP node propagate but do not extend to more SMP nodes. On the other hand, we also see that there is additional speedup from 32 to 64 processors, once we have paid the cost for using more than only one SMP node.

Grid type	OpenMP (on/off)	Procs 16	32	64
Rectangular	off	6.7	9.7	13.3
	on	9.9	12.7	17.5
Cubic	off	7.2	9.7	12.5
	on	9.9	11.0	17.6

Table 5.5.3: Comparison of scaled speedup between pure MPI (off) and hybrid parallelization (on) (`v4.2`, `SCOTCH` and LU factorization). For the hybrid version we report the best scaled speedup (2 or 4 OpenMP threads per MPI process).

5.6 Conclusion

Our study of the distributed memory general sparse solver `MUMPS` has shown that its performance with respect to computation time can be improved on networks of SMPs. In this chapter, we have presented scheduling algorithms designed to address the problem of heterogeneous architectures. Our approach consists of including architectural considerations during both the partial static mapping of the tasks onto the processes and during the dynamic scheduling.

We have illustrated key properties of our approach by detailed case studies on a wide range of test problems. The comparison of versions with and without architecture considerations has illustrated the main benefit of our approach, an improved scalability when running on a two-level architecture represented by an IBM SP3 with up to 8 SMP nodes and 128 processors. Even greater benefits are shown by our new approaches on an IBM SP4. Finally, we have discussed the mixing of OpenMP and MPI in order to further improve the scalability of the code.

We want to mention that the algorithms presented in this chapter, although only experimented on such target computers, will naturally adapt to a more general framework

of computers with heterogeneous processors and multilevel architectures. Most of the additional effort that has still to be done is in the development of middleware tools that will enable us to have a good estimation (with probably dynamic readjustments) of the parameters characterizing such a multilevel architecture. This will clearly require further significant work that is outside the scope of this chapter but that will benefit from the algorithms and results presented here.

Chapter 6

Hybrid scheduling for the parallel solution of unsymmetric matrices

In this chapter, we consider the problem of designing a dynamic scheduling strategy that takes into account both workload and memory information in the context of the parallel multifrontal LU factorization. The originality of our approach is that we base our estimations (work and memory) on a static optimistic scenario during the analysis phase. We then use it during the factorization phase to constraint the dynamic decisions. The task scheduler has been redesigned and improved to take into account these new features. Performance analyses show that the memory estimation becomes very close to the memory used and that even in a constrained memory environment we decrease the factorization time with respect to the initial approach.

6.1 Introduction

The work presented in [64] has shown how to use a memory based dynamic scheduling to improve the memory management of a parallel multifrontal approach. However, the authors also noticed that they can significantly improve the memory behaviour but at the cost of an increase in the factorization time. Another important issue concerns the overestimation of the memory needed for parallel factorization. Indeed, even if in [11] the authors have shown that with the concept of candidate processors the memory estimates can be significantly reduced, there is still an important and unpredictable gap between real and estimated memory. Hence another target will be to decrease the memory estimates of the analysis and to respect them during the factorization.

In this chapter, we propose a scheduling approach that uses both memory and workload information in order to obtain a better behaviour in terms of factorization time, estimated memory and memory used in the context of the parallel LU factorization. We will explain in the conclusions why the symmetric case is in fact more complicated. The main principle of our approach is to use an optimistic scenario during the analysis that is then relaxed to offer flexibility for the factorization phase.

To conclude this introduction section, we first describe in 6.1.1 and 6.1.2 two functionalities which have been added to MUMPS since the work described in the previous chapter. In Section 6.1.3, we then describe the constraints and objectives of our work. This chapter is then organized as follows. Section 6.2 introduces the quantities that will influence the dynamic decisions or our dynamic scheduling algorithm described in Section 6.3. In Section 6.4 we then present experimental results on large unsymmetric

matrices. We advise the reader to refer to Section 5.2 or to [8] if he is not familiar with the types of parallelism involved in MUMPS.

6.1.1 The four zones

Initially, the assembly tree was separated into two zones, the upper part of the tree (above layer 0) and the bottom part of the tree (below layer 0) where each subtree is mapped onto a unique process.

We decided to separate the tree into 4 zones instead of 2 (see Figure 6.1.1). Zone 4 corresponds to the bottom of the tree. It was suggested in the conclusion of [11] that the mapping of the upper part of the assembly tree could be separated into two zones. The first zone (zone 1) would correspond to a relaxed proportional mapping whereas the second zone (zone 2) would correspond to a stricter proportional mapping. Hence the flexibility offered at the top of the tree would enable the master processes to correct the mistakes or the unbalances due to the variations of the load of the machine. Guided by this remark, zones 1 and 2 have been implemented.

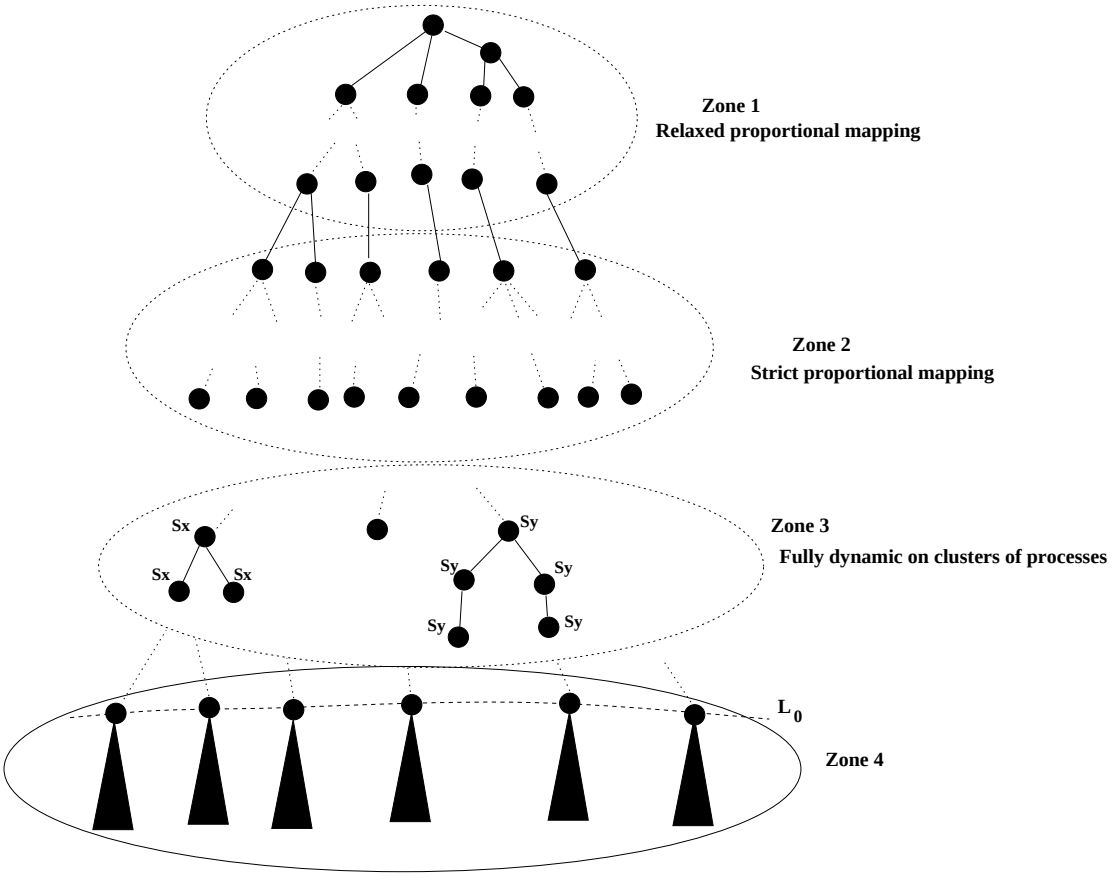


Figure 6.1.1: The four zones of the assembly tree. S_x and S_y are sets of preferential processes.

Moreover we decided to add another zone (zone 3) in which each son inherits the

preferential processes from its parent. This choice was motivated by the following experimental observations:

- on a small number of processors the fully dynamic code is very competitive,
- increasing the number of candidate processes in the first layers decreases the size of the buffers (if a node with few candidates and a large contribution block appears then the size of the communication buffers and the memory estimates increase),
- with more candidates above zone 4 the proportional mapping is better respected on layer 0,
- on clusters of SMPs, this will naturally take into account the memory locality.

The limit of zone 3 depends on a parameter $proc_{max}$ which corresponds to a number of processes. During the top-down approach of the proportional mapping, if the number of preferential processes of a node x is smaller than or equal to $proc_{max}$ then x and all its descendants (above zone 4) belong to zone 3 and have the same set of preferential processes (see sets S_x and S_y in Figure 6.1.1).

The extra freedom given in zone 1 does not perturb too much the memory estimation of MUMPS version 4.3 which is based on an average worst case. It is not the case for zone 3. In this zone, the memory estimates behave like the fully dynamic code which has shown that it severely overestimates the required space. That is why the four zones approach will not be included for the standard version of the experimental section (Section 6.4). We will give more details about this issue in Section 6.2.1.

6.1.2 Irregular tasks scheduling

In [64], the authors developed an approach with irregular partitions to decrease the memory usage. We will also use this capability (actually we need it) to offer more flexibility to our new scheduling strategy.

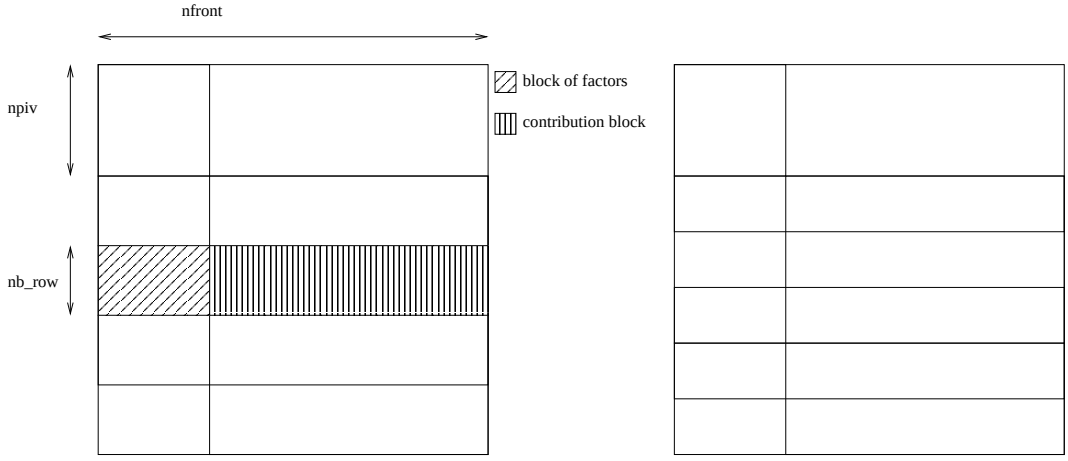
6.1.3 Our constraints and objectives

When the target is time reduction, the master process of each node determines a partition in order to "balance as much as possible the workload" between the processes. In our context, we have the same objective, but also have to respect additional constraints. In this section, we first present the memory constraints taken into account during scheduling. Then, we give a generic formulation of this new constrained problem.

We recall that a frontal node is composed of fully summed rows and partially summed rows. The partially summed rows are given to slave processes which store factors and contributions blocks (see Figure 6.1.2.a).

The different criteria which are used to estimate the memory constraints are presented below:

- **Amount of memory available.** It corresponds to the remaining memory which can be used to store the contribution blocks and the factors. It varies during the



(6.1.2.a) Regular partition of a type 2 node on 4 slaves.

(6.1.2.b) Regular partition of a type 2 node on 5 slaves.

Figure 6.1.2: Examples of partitions of a type 2 node.

factorization. For each process i, mem_i will refer to this quantity (see Section 6.2.1 for more details).

- **Maximum factor size.** It corresponds to the maximum size of the factors that a master can give to a slave process i. It will be denoted by $fact_i$. This quantity is related to both the static scenario used to estimate the memory during analysis and to dynamic information obtained during factorization (see Section 6.2.2).
- **Maximum buffer size.** Since we want to be able to send and receive messages corresponding to a slave task, we take the most restrictive size between the send and receive buffers. We give more details about the size of the send and receive buffers in Section 6.2.3. For each process i, buf_i will denote this quantity.

We now define the notation used to describe our algorithms. Let us consider a node of the assembly tree of order $nfront$ with $npiv$ variables to eliminate. We recall that the expressions below correspond to the unsymmetric case (see Section 6.5 for the symmetric case). Let nb_row be a function such that for each slave process i and for a buffer size buf_i,

$$nb\_row(buf_i) = \frac{buf_i}{nfront},$$

for an available memory mem_i,

$$nb\_row(mem_i) = \frac{mem_i}{nfront},$$

for a maximum factor size $fact_i$,

$$nb\_row(fact_i) = \frac{fact_i}{npiv},$$

for a maximum number of operations $load_i$,

$$nb\_row(load_i) = \frac{load_i}{npiv(2 \times nfront - npiv)}.$$

If the master of a node knows the above quantities, it knows the maximum number of rows that it can assign to each candidate. During the factorization, each master has to "balance as much as possible the workload" between its s_1, s_2, ..., s_k candidates giving them n_1, n_2, ..., n_k rows respectively such that for each slave i, $n_i \leq \min\{nb\_row(buf_i), nb\_row(mem_i), nb\_row(fact_i)\}$.

Even if the problem of finding the best workload balance which respects the memory constraints is easy to solve theoretically, in practice, communication schemes, granularity and the topology of the assembly tree need to be considered to answer this question. That is why we do not give, in this section, more details about the meaning of "balance as much as possible the workload". Note also that the above problem may not have a solution because the memory constraints are too restrictive. These points will be examined in Section 6.3.

6.2 Static and dynamic estimates

This section describes the measures that are used by our scheduler. Some of these quantities are computed during the analysis (the memory estimations, the size of the buffers, the size of the area reserved for factors), some are also adjusted during the factorization (the memory available to store contribution blocks, the memory available to store factors).

6.2.1 Memory estimates and available memory

We first explain how we decrease the total memory allocated compared to the standard version of MUMPS.

After mapping the candidates, the memory estimates are performed during a bottom-up processing of the assembly tree. For each process involved in the computation of a node, its memory estimate is decreased when a contribution block is removed and is increased when assemblies, activation of tasks and storage of factors occur. Note that it is only during factorization that for a type 2 node we decide how many and which processes among the candidate processes will be used to help the master process in its task. For type 2 nodes our new estimates, computed during the analysis phase, are based on an average optimistic scenario instead of the worst case as in [11]. In both cases the estimation assumes regular partitions of the contribution block. In the worst case, we first compute the minimum number of slaves, min_needed, to perform all the work for the slaves (this number depends on internal parameters and algorithmic aspects that fix the maximum granularity). Then, for our simulation each candidate receives a block of size $nfront(nfront - npiv)/min\_needed$ where $nfront$ and $npiv$ are defined earlier. In the optimistic case, we assume that work can be given to all available slaves and so each candidate receives a block of size $nfront(nfront - npiv)/ncand$ where $ncand$ is the number of candidates of the type 2 node. As $nfront(nfront - npiv)/min\_needed > nfront(nfront - npiv)/ncand$ then the estimates will be smaller in the optimistic scenario. For example, let us consider the type 2 node of Figure 6.1.2.a. Let us suppose that it has 5 slaves, that the block which has to be distributed to the slaves has a size of

200 MBytes and that at least 4 slaves are needed. Then the worst case (Figure 6.1.2.a) will give 50 MBytes to each candidate process whereas the optimistic scenario (Figure 6.1.2.b) will give 40 MBytes to each slave. So we save 10 MBytes per process.

These estimates are then relaxed by a percentage (by default it is equals to 20%) to offer more flexibility to the scheduling. This supplementary memory enables us to take into account extra fill-in due to numerical pivoting and to offer more freedom to the dynamic decisions. It will also reduce the amount of data compressions involved in a parallel environment because of the irregular access to the contribution blocks. The memory available on each processor is then dynamically updated as proposed in [63].

6.2.2 Maximum size of factors

The maximum size of factors is used by the scheduler to determine the largest portion of the factors that can be given to a candidate process. It is composed of two terms.

For each node J of the assembly tree, the first term $fact\_anal_i(J)$ is estimated during the analysis. It corresponds to the size of the factors given to the process i with the optimistic scenario (with the convention that if a process i is neither the master nor a candidate of node J, $fact\_anal_i(J) = 0$). Thus, according to the analysis scenario, a process i will store the quantity $fact\_anal_i = \sum_J fact\_anal_i(J)$ of factors.

Algorithm 6.2.1 Update of the supplementary memory for the factors.

`Initialization on process` p_i `:`
  Let Δ_i be the initial supplementary memory for the factors.
  Include (p_i, Δ_i) in a message msg_sup_mem.
  Asynchronous send of msg_sup_mem to the other processes.

`After selection of a set` $\mathcal{S}$ `of slaves by process` p_i `for the node` J `:`
  for all candidate process p_k **do**
    if $p_k \in \mathcal{S}$ **then** /* p_k has been selected */
      $\delta_k =$ Estimated size of the factors for a slave task $-$ Real size of the factor given to p_k
    else
      $\delta_k =$ Estimated size of the factors for a slave task
    end if
    Include (p_k, δ_k) in a message msg_sup_mem.
  end for
  Asynchronous send of msg_sup_mem to the other processes.

`After the reception of a message` msg_sup_mem `:`
  for all (p_i, δ_i) **do**
    $\Delta_i = \Delta_i + \delta_i$
  end for

The second term, the flexibility Δ_i, is initialized to the supplementary memory given to store the factors. It enables the scheduling to deviate from the optimistic analysis scenario. Using Algorithm 6.2.1 Δ_i is adjusted dynamically during the factorization phase. Hence, after having updated information about workload and memory during the selection of the slaves of a node J, the master knows that it cannot give more than $fact_i(J) = fact\_anal_i(J) + \Delta_i$ factors to the candidate i. Obviously, if there are no

numerical problems and if $\Delta_i = 0$ for each process i, then the factorization will respect the partition of the factors predicted by the analysis phase.

6.2.3 Buffer size

During the bottom-up phase of the memory estimation, the size of the send and receive buffers is computed. For each node it is based on the optimistic scenario. For each candidate process, the size of the buffer is adjusted to the maximum size of the messages that it may have to send or receive. As for the memory estimation, a relaxation parameter is then used.

6.2.4 Workload and anticipation

The decision of a master of a type 2 task is guided by its view of the workload of its slaves referred to as $load_i$. For each process p_i, $load_i$ and its variations are taken into account thanks to an asynchronous communication scheme. Experiments in [63] have shown the positive effects of anticipating the memory variations. Algorithm 6.2.2 describes this mechanism for the workload. The workload of a master of a node is updated as soon as all of its children have been activated.

Algorithm 6.2.2 Anticipation of the tasks.

`Initialization on process` p_i
  for all nodes J for which p_i is the master **do**
    Set $nb\_children(J)$ to the number of children of J.
  end for

`Sending of a message` $child\_OK$ `when task` J `starts on the master` p_i `:`
  Let K be the parent of node J.
  Let p_k be the master in charge of task K.
  Include K in a message $child\_OK$.
  Asynchronous send of $child\_OK$ to the process p_k.

`After receiving a message` $child\_OK$ `on process` p_i `:`
  Extract task J from the message $child\_OK$.
  $nb\_children(J) = nb\_children(J) - 1$.
  if $nb\_children(J) = 0$ **then**
    Let W_J be the work associated with the task of the master of node J.
    Include (p_i, W_J) in a message msg_load_update.
    Asynchronous send of msg_load_update to the other processes.
  end if

`After receiving a message` msg_load_update `:`
  for all (p_i, W) **do** /* Update the workload of processor p_i */
    $load_i = load_i + W$
  end for

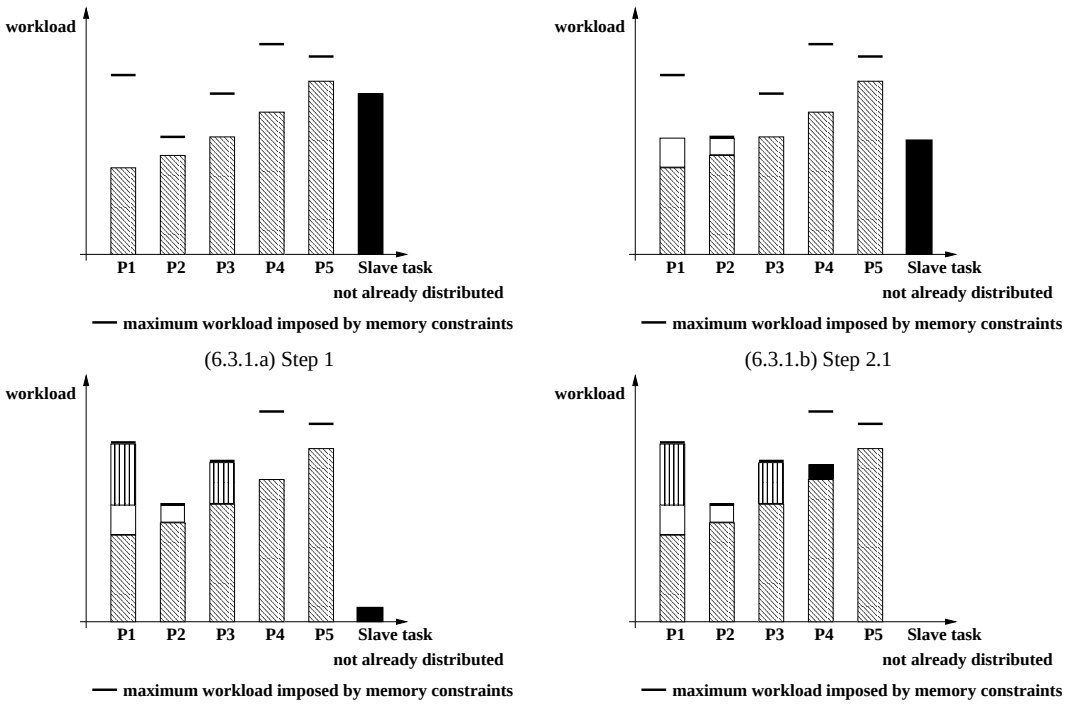
6.3 Hybrid dynamic scheduling

In this section, we describe the algorithms used to balance the workload while taking into account the memory constraints.

Algorithm 6.3.3 Main steps of hybrid scheduling

Receive information related to workload and memory.
1 Define a reference number of processes $nref$ and an advised maximum number of slaves $nslaves_{lim}$ ($nref \leq nslaves_{lim}$).
2 Try to balance the work on a number of processes between $nref$ and $nslaves_{lim}$ taking into account memory constraints.
3 If step 2 did not succeed then increase $nref$ until a balanced partition that satisfies memory constraints has been found or the number of candidates has been reached.
4 If steps 2 and 3 did not succeed then for each candidate relax the constraint of analysis prediction and try to balance the work on the candidates taking into account these new constraints.
5 If steps 2, 3 and 4 did not succeed then try to balance the work among the candidates taking into account only the buffer constraints.

Algorithm 6.3.3 presents the main scheme of our hybrid dynamic scheduling. Note that violating a constraint due to the buffers would automatically lead to a failure. That is why we never relax them during steps 4 and 5.



(6.3.1.a) Step 1 (6.3.1.b) Step 2.1

(6.3.1.c) Step 2.3 (6.3.1.d) Step 3

Figure 6.3.1: Example of hybrid scheduling on 5 processes.

Before describing in full details our algorithms we illustrate in Figure 6.3.1 the behaviour of first steps of Algorithm 6.3.3. Let us assume that at step 1 both the reference number of processors $nref$ and the advised number of processors $nslaves_{lim}$ are set to 3. Work is

then assigned at step 2 to these three processes (Figures 6.3.1.b and 6.3.1.c). Note that the numbering of the steps 2.1 and 2.3 is used to have a coherent description with the detailed Algorithm 6.3.6 (on our example step 2.2 is inactive because $nref = nslaves_{lim}$). Finally the remaining amount of work can be assigned after having included P4 in the set of slaves (Figure 6.3.1.d).

We now give a more precise and formal description of our hybrid scheduling algorithm. Let us define first notations used in our algorithms. For each node J we have :

- $ncand$ the number of candidates,
- $\{p_1, \ldots, p_{ncand}\}$ the set of candidates sorted by increasing workload,
- W_{master} the workload of the master task,
- W_{slave} the workload associated with the sum of all the slave tasks,
- for each candidate p_i :
  - mem_i the size of available memory,
  - buf_i its buffer size,
  - $fact_i$ the maximum factor size that can be given to this process,
  - $load_i$ its workload and
  - $MAXload_i$ ($\geq load_i$) the workload limit that it cannot exceed to respect the previous memory limitations.

Algorithms 6.3.4 and 6.3.5 are the elementary pieces used during our dynamic scheduling algorithm. Algorithm 6.3.4, referred to as `ASSIGN`, tries to equilibrate, in a one pass algorithm, the distribution of a number of rows, R_{in}, over a set $\mathcal{S}$ of k processes. Memory constraints per process are also considered.

`ASSIGN` will be used in three contexts:

- *Context 1.* We want to adjust the workload of the processes to a specific work limit per process W_{max}. There is not additional constraints on the number of row and R_{max} is set to ∞. The algorithm then tries, while respecting memory constraints, to assign $W - load_i$ work to each process i. For example, let us consider the processes of Figure 6.3.1.a. If `ASSIGN` is called with $R_{max} = \infty$ and W_{max} equals to the workload of process P3, then process P1 becomes as loaded as P3 and P2 is saturated as shown in Figure 6.3.1.b.
- *Context 2.* We want to give at most R_{max} rows per process. For example, let us suppose that a master has to distribute 500 rows to 5 candidates $p_1, \ldots, p_5$ and that none of them is loaded ($load_i = 0$) and that $MAXload_i$ is set to ∞. If `ASSIGN` were called in Context 1 with W_{max} large enough, the system would be unbalanced after the distribution. If `ASSIGN` is called with $R_{max} = 100$ and $W_{max} = \infty$, then each candidate receives 100 rows and the system remains balanced.
- *Context 3.* We want to mix the two previous cases. We try to give almost the same number of rows to each process but we also want to respect a workload limit (in this case, $W_{max} \neq \infty$ and $R_{max} \neq \infty$).

Algorithm 6.3.4 Assignment of a number of rows to a set of processes which respect memory and workload constraints with one pass.

`ASSIGN (` $\mathcal{S}$ `,` W_{max} `,` R_{max} `,` R_{in} `,` R_{out} `,` $nsat$ `):`
`INPUT:`
$\mathcal{S} = \{p_1, \ldots, p_k\}$: set of processes sorted by increasing workload.
W_{max} : workload limit.
R_{max} : maximum number of rows which can be assigned per process.
R_{in} : the number of rows to assign.
`OUTPUT:`
R_{out} : number of remaining rows ($R_{in}-$ the number of assigned rows).
$nsat$: number of saturated processes in $\mathcal{S}$.

$nsat = 0$ and $R_{out} = R_{in}$.
for $j = 1$ to k **do**
if $load_j < W_{max}$ **then**
$n_j = \min\{nb\_row(buf_j), nb\_row(mem_j), nb\_row(fact_j)\}$.
$n_j = \min\{n_j, nb\_row(W_{max} - load_j), R_{max}, R_{out}\}$.
Assign n_j rows to p_j and update mem_j , $fact_j$, $load_j$ and buf_j .
if $n_j = 0$ or memory bound reached after update **then** $nsat = nsat + 1$.
$R_{out} = R_{out} - n_j$.
end if
end for
return R_{out} , $nsat$.

Algorithm 6.3.5, referred to as `ASSIGN_EQ`, is an iterative algorithm that tries, at each iteration, to give an equal number of rows per process. After one iteration of the algorithm, if some rows still have to be distributed then the fixed number of rows R_{max} is adjusted for the next pass with the objective of equilibrating the load of the slaves which have not already saturated their memory constraints.

Algorithm 6.3.5 Assignment of a number of rows to a set of processes which respect memory and workload constraints with several passes.

`ASSIGN_EQ (` $\mathcal{S}$ `,` W_{max} `,` R_{in} `,` R_{out} `):`
`INPUT:`
$\mathcal{S} = \{p_1, \ldots, p_k\}$: set of processes sorted by increasing workload.
W_{max} : workload limit.
R_{in} : the number of rows to assign.
`OUTPUT:`
R_{out} : number of remaining rows ($R_{in}-$ the number of assigned rows).

$R_{max} = Int(R_{in}/k) + 1$.
`ASSIGN (` $\mathcal{S}$ `,` W_{max} `,` R_{max} `,` R_{in} `,` R_{out} `,` $nsat$ `).`
while $R_{out} \neq 0$ and $R_{out} \neq R_{in}$ and $nsat < k$ **do** /* there are still unassigned rows and a call to `ASSIGN` can improve the partition */
$R_{max} = Int(R_{out}/(k - nsat)) + 1$.
$R_{in} = R_{out}$.
`ASSIGN (` $\mathcal{S}$ `,` W_{max} `,` R_{max} `,` R_{in} `,` R_{out} `,` $nsat$ `).`
end while
return R_{out} .

Algorithm 6.3.6 describes how our hybrid scheduling produces a balanced partition. We

use GET_NREF defined in Algorithm 6.3.7 to compute a reference number of processes and an advised maximum number of slaves. The reference number of processes will be used at step 2.1 of Algorithm 6.3.6 to balance the workload.

The second part of Algorithm 6.3.6 is divided into 3 stages (lines 2.1, 2.2 and 2.3). During the first call to ASSIGN (line 2.1), work is given to $nref$ processes to reach the workload limit $W_{max} = load_{nref}$ (*Context 1*). In stage 2.2, we increase the number of slaves until reaching the advised maximum number of processes or until all rows have been distributed. Note that at the end of this loop, the workload of the selected slaves is necessarily smaller than $load_{nslaves_{lim}}$ (the calls to ASSIGN done by ASSIGN_EQ corresponds to *Context 3*). In the third stage 2.3, if there are remaining rows we try to balance the workload without imposing a workload limit (in this case the calls to ASSIGN correspond to *Context 2*). The same approach is used during step 3 while increasing the number of processes.

Algorithm 6.3.6 Hybrid dynamic scheduling on a set $\{p_1, \ldots, p_{ncand}\}$ of candidates sorted by increasing workload.

Receive information related to workload and memory.
1 GET_NREF ($nref$, $nslaves_{lim}$).
$R_{in} = nb\_row(W_{slave})$.
2.1 ASSIGN ($\{p_1, \ldots, p_{nref}\}$, $W_{max} = load_{nref}$, $R_{max} = \infty$, R_{in} , R_{out} , $nsat$).
2.2 **while** $nref < nslaves_{lim}$ and $R_{out} > 0$ **do**
$nref = nref + 1$.
ASSIGN_EQ ($\{p_1, \ldots, p_{nref}\}$, $W_{max} = load_{nref}$, R_{out} , R_{out}).
end while
2.3 **if** $R_{out} > 0$ **then** ASSIGN_EQ ($\{p_1, \ldots, p_{nref}\}$, $W_{max} = \infty$, R_{out} , R_{out}).
3.1 **while** $nref < ncand$ and $R_{out} > 0$ **do**
$nref = nref + 1$.
ASSIGN_EQ ($\{p_1, \ldots, p_{nref}\}$, $W_{max} = load_{nref}$, R_{out} , R_{out}).
end while
3.2 **if** $R_{out} > 0$ **then**
$nref = ncand$.
ASSIGN_EQ ($\{p_1, \ldots, p_{nref}\}$, $W_{max} = \infty$, R_{out} , R_{out}).
end if
4 **if** $R_{out} > 0$ **then**
Set $fact_i = \infty$ for each candidate i.
ASSIGN_EQ ($\{p_1, \ldots, p_{nref}\}$, $W_{max} = \infty$, R_{out} , R_{out}).
end if
5 **if** $R_{out} > 0$ **then**
Set $fact_i = \infty$ and $mem_i = \infty$ for each candidate i.
ASSIGN_EQ ($\{p_1, \ldots, p_{nref}\}$, $W_{max} = \infty$, R_{out} , R_{out}).
end if
if $R_{out} > 0$ **then**
return failure because of buffer constraint.
else
return $nref$ and partition.
end if

Moreover, since in zone 3 the tree parallelism is sufficient, we want to allow the algorithm to create unbalanced partitions and large type 2 tasks. In this way we can expect to improve the performance and limit the volume of communications. That is why in Algorithm 6.3.7 we decide that all candidates can receive work during the first call to

ASSIGN ($nref = ncand$) and their workload is adjusted on the most loaded process ($W_{max} = load_{ncand}$). Hence larger tasks will be given to the selected candidates and the type 2 parallelism will be limited.

If the node is not in zone 3, we want to balance as much as possible the workload between the candidates and have tasks of reasonable granularity. We compute the following two quantities to take into account these two aspects:

- the minimum number of slaves which would ensure a balanced partition

$$\max\{i \text{ such that } \sum_{j=1}^{i} \min(load_i, MAXload_j) - load_j \leq W_{slaves}\},$$

- the maximum number of slaves $nslaves_{lim}$ which prevent us from creating tasks smaller than ρ times the work of the master task. In practice, we set $\rho = 70\%$.

To illustrate our discussion about $nslaves_{lim}$, let us consider that the work is well equilibrated among all candidates of a relatively small type 2 node. In this case $nref$ might be very large (probably equal to the number of candidates) which might not be appropriate for efficiency. Since $nslaves_{lim}$ is designed to maintain a minimum granularity, we make sure in Algorithm 6.3.7 that $nref$ is smaller that $nslaves_{lim}$.

Algorithm 6.3.7 Computation of a reference number of slaves.

GET_NREF ($nref$, $nslaves_{lim}$):
OUTPUT:
$nref$: reference number of slaves.
$nslaves_{lim}$: advised maximum number of slaves.

Let $\rho > 0$ be a granularity parameter.
if the current node is not in zone 3 **then**
$nslaves_{lim} = \min(ncand, \max(\frac{W_{slave}}{\rho\, W_{master}}, 1))$.
$nref = \max\{i \text{ such that } \sum_{j=1}^{i} \min(load_i, MAXload_j) - load_j \leq W_{slaves}\}$. /* candidate processes are sorted by increasing workload */
$nref = \min(nref, nslaves_{lim})$.
else /* the node is in zone 3 */
$nref = ncand$ and $nslaves_{lim} = ncand$.
end if
return $nref$ and $nslaves_{lim}$.

6.4 Experimental results

In this section we analyse the effects of our hybrid approach. We first describe our testing environment. In Section 6.4.2, we analyse the behaviour of our algorithms in terms of estimated memory and memory effectively used during factorization. The influence on the factorization time is discussed in Section 6.4.3.

6.4.1 Experimental environment

In this section, we focus on two very large unsymmetric test matrices described in Table 6.4.1:

- CONV3D64 has been provided by CEA-CESTA and was generated using AQUILON (`http ://www.enscpb.fr/master/aquilon`),
- ULTRASOUND80 comes from propagation of 3D ultrasound waves and has been provided by Masha Sosonkina.

We consider two orderings PORD [95] and METIS [73] to analyse the behaviour of our hybrid scheduling on different tree topologies.

			PORD		METIS	
Matrix	order	nnz	$nnz(L\|U)$ $\times 10^6$	Ops $\times 10^9$	$nnz(L\|U)$ $\times 10^6$	Ops $\times 10^9$
CONV3D64	836550	12548250	4699.9	48540	2693.9	23880
ULTRASOUND80	531441	33076161	1044.9	5225	981.4	3915

Table 6.4.1: Test set. nnz: number of nonzeros in the matrix. $nnz(L|U)$: number of nonzeros in the factors. Ops: total number of operations during factorization.

Our target machine is the IBM SP from IDRIS[1]. It is composed of 6 clusters with 16 SMP nodes of 4 processors (Power 4/1.7Ghz P655). Each node shares 8GBytes of memory and is interconnected to the others by a Federation switch. We will compare the following versions of MUMPS on 64 and 128 processors:

- The fully dynamic version will be referred to as `MUMPS_dyn`. It corresponds to the dynamic version of MUMPS used in [9]: when a master wants to select a slave, it can choose any process. In other words, for each node of the assembly tree, all processes are candidates.
- The standard version with proportional mapping and candidates will be referred to as `MUMPS_cand`. It corresponds to the version used in [11] except that the mechanism of Algorithm 6.2.2 for anticipating the workload has also been included. A master selects its slaves among candidates and balances the workload using regular partitions.
- The hybrid version will be referred to as `MUMPS_hyb`. It corresponds to a candidate version implementing all the algorithms described in this section. In particular, the separation of the tree in four zones (see Section 6.1.1), the estimation of the supplementary available memory for the factors (see Algorithm 6.2.1) and the hybrid scheduling (see Section 6.3) are included. The estimation of the maximum number of slaves returned by `GET_NREF` is done with $\rho = 70\%$.

[1] Institut du Développement et des Ressources en Informatique Scientifique

6.4.2 Estimated and used memory

In this section, we analyse the memory behaviour of the three versions of our solver. We look at both the predicted memory peak and the actually used memory. We are interested in both the average memory per process and the peak between processes.

Matrix		MUMPS_dyn Estim	MUMPS_cand Estim	MUMPS_cand Real	MUMPS_hyb Estim	MUMPS_hyb Real
CONV3D64	Max	–	–	–	–	–
PORD	Avg	–	–	–	–	–
CONV3D64	Max	137.7	98.44	84.53	84.99	76.78
METIS	Avg	118.3	68.98	60.84	61.30	60.93
ULTRASOUND80	Max	103.8	45.31	32.85	33.23	34.66
PORD	Avg	94.3	30.03	24.82	25.40	24.73
ULTRASOUND80	Max	110.1	36.08	37.42	28.75	26.75
METIS	Avg	104.0	26.24	22.79	23.02	22.37

Table 6.4.2: Estimated and real memory for the factorization on 64 processors. Max: maximum amount of memory. Avg: average memory per process. Memory in millions of reals. Memory allocated is 20% more than estimated. – means that the execution did not succeed because memory limits has been attained.

We recall that MUMPS_dyn and MUMPS_cand versions estimate the memory using a worst case scenario whereas the MUMPS_hyb version uses an optimistic scenario. Note for all cases we relax the memory estimated by 20% to run the factorization (except for MUMPS_cand version on CONV3D64 with 64 processors and the METIS ordering where this percentage is reduced because of memory limitations on the machine).

Matrix	Ordering	MUMPS_cand Estim	MUMPS_cand Real	MUMPS_hyb Estim	MUMPS_hyb Real
CONV3D64	Max	93.17	–	89.71	68.06
PORD	Avg	64.91	–	53.61	49.26
CONV3D64	Max	72.98	46.28	50.85	46.78
METIS	Avg	39.55	31.41	34.96	32.09
ULTRASOUND80	Max	63.18	26.56	24.47	20.99
PORD	Avg	23.19	14.57	14.64	13.19
ULTRASOUND80	Max	32.69	17.45	18.99	18.22
METIS	Avg	18.16	12.43	13.15	11.91

Table 6.4.3: Estimated and real memory for the factorization on 128 processors. Max: maximum amount of memory. Avg: average memory per process. Memory in millions of reals. Memory allocated is 20% more than estimated. – means that the execution did not succeed because memory limits has been attained.

Tables 6.4.2 and 6.4.3 show the memory estimated and the memory used on 64 and 128 processors respectively. We first see in Table 6.4.2, already reported in [9] and [11], that the fully dynamic version severely overestimates the size of the factors. We also notice that the hybrid version significantly reduces the estimated memory (both average and peak) and does nearly an exact prediction. This behaviour is emphasized on 128 processors. Moreover the new strategy can lead to a significant decrease in the memory peak (see for example ULTRASOUND80 with METIS on 64 processors).

6.4.3 Factorization time

In this section, we analyse the factorization time. We observe two different effects. On the one hand, irregular partitions of Section 6.1.2 offer the flexibility to balance the workload better, which should improve the factorization time. On the other hand, memory constraints prevent the master from making an optimal decision in terms of balancing the workload, which should moderate the effect of mapping irregular tasks.

		64 processors		128 processors	
Matrix	Ordering	MUMPS_cand	MUMPS_hyb	MUMPS_cand	MUMPS_hyb
CONV3D64	PORD	–	–	–	617.34
CONV3D64	METIS	293.11	228.64	236.59	181.76
ULTRASOUND80	PORD	62.21	55.68	57.32	61.93
ULTRASOUND80	METIS	57.66	48.83	45.62	46.65

Table 6.4.4: Factorization time in seconds with 64 and 128 processors.

Table 6.4.4 shows the impact of our new strategy on the factorization time. Note that hybrid scheduling enables the solution of CONV3D64 with the PORD ordering on 128 processors. On most of the cases, the hybrid approach improves the factorization time on both 64 and 128 processors. In other words the memory constraints do not prevent the scheduling from balancing the workload well. We can observe a small increase in the factorization time with ULTRASOUND80 on 128 processors. This increase is not due to wrong dynamic decisions but to an increase in the amount of data compressions when memory estimates are very tight (MUMPS_cand and MUMPS_dyn perform respectively 17 and 157 compressions). Note also that for this matrix the huge gain in terms of estimated memory (a reduction by a factor 2.5) makes the small speed-down of 7% affordable. Furthermore, if we relax the memory estimated by 100%, the new strategy becomes 5% faster than the MUMPS_cand version while still allocating 38% less memory.

Table 6.4.4 also shows that the factorization time does not scale well between 64 and 128 processors. Note that it is worse for the hybrid version because it is significantly faster than the candidate version on 64 processes. We will come back to this point in our conclusions.

Figure 6.4.1 shows the influence of the memory relaxation parameter on the factorization time and memory used to factorize the CONV3D64 matrix on 128 processors using METIS. If the relaxation is too small ($< 20\%$), MUMPS does not have enough flexibility to balance the workload and performs too many memory compressions (there are 636, 541, 394, 321 and 238 memory compressions for a relaxation of 5%, 10%, 20%, 30% and 40% respectively) which severely penalizes the factorization time. Note that for a relaxation greater than 20% the factorization time does not significantly change. Note also that the total memory peak and the peak of the stack are less sensitive to the variation of the relaxation. Finally one should point out that the candidate version computes the factors in 236 seconds and does 293 compressions whereas the factorization with our hybrid strategy takes 181 seconds and performs 394 compressions. Thus, the improvement of the performance is not due to a decrease in the number of compressions, but to an improvement of the decisions taken by our new scheduler.

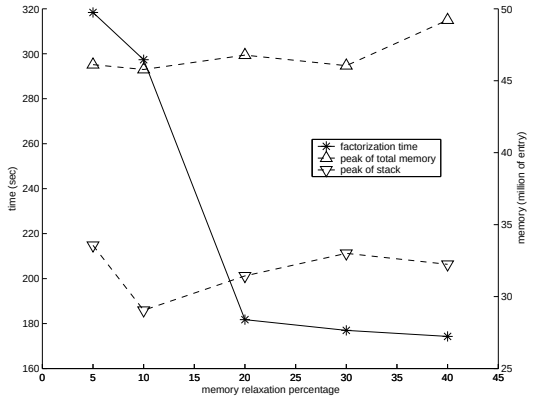


Figure 6.4.1: Influence of the memory relaxation on factorization time and memory with the CONV3D64 matrix on 128 processors. METIS ordering is used.

6.5 Concluding remarks and future work

We presented in this chapter a hybrid approach to dynamic scheduling which takes into account information about workload and memory in the context of parallel LU factorization. We proposed an approach with modifications of the static mapping and the dynamic execution. We have shown the benefits of our approach on two large real test cases with two different orderings on 64 and 128 processors. Our intention for future work is to improve and develop the three axes reported below to address symmetric matrices and improve the scalability of the solver.

Firstly, we want to adapt the hybrid scheduling to the symmetric case. This is slightly more complex, since in the symmetric case, the relations between the memory, the workload (in number of operations) and the number of rows is not straightforward and depends on the previous selection of slaves (see the partition of a type 2 node in the symmetric case in [8]). Moreover, in the unsymmetric cases all communications come from the master whereas in the symmetric case slaves have to communicate between each other. This communication scheme has to be taken into account for an optimal decision.

Secondly, the candidate version of `MUMPS` has been adapted to clusters of SMPs in [10]. We want to develop hybrid approaches which also exploit this feature of the computer architecture. In our context, it seems already that the size of the SMP nodes will give a quite natural criteria to define the new zone 3.

Finally, we saw in our experiments that even if the absolute behaviour in terms of memory and factorization time has been improved, the time did not scale well between 64 and 128 processors. We think that the memory peaks and the workload balance between processes could be enhanced with a deeper layer 0. Our tighter memory estimation makes this possible in a simpler way than with the candidate version. In this context, it should also become useful to apply the proportional mapping on layer 0 (it is easier than in the candidate version because of the new zone 3).

Conclusions

Symmetric indefinite matrices

The first part of this thesis has dealt with symmetric indefinite matrices. In the first chapter, we have shown how the use of an appropriate scaling can solve many computational difficulties. We have shown how a bipartite maximum weighted matching can be used on such matrices to effect a symmetric scaling and identify potential 2×2 pivots. Another benefit of our work is that the analysis phase gives a better indication of the work and storage required by the subsequent factorization, since numerical problems are anticipated.

We have developed new classes of greedy orderings that we called "(relaxed) constrained orderings" and that succeed in decreasing the factorization time. Our intention is to decrease even more the fill-in while keeping a stable factorization by extending the concept of constrained ordering in the context of graph partitioning.

In Chapter 2, we have adapted the analysis phase of a solver that manages the diagonal zeros of augmented systems and we have improved the behaviour of its factorization. We have stressed some advantages such as the reduction of the number of operations, of the memory used, and of the size of the factors. However, multifrontal methods designed for general symmetric indefinite matrices remain better on a large range of augmented systems.

We have presented in Chapter 3 different alternatives for static pivoting. We have developed a first static pivoting approach that applies 1×1 and 2×2 perturbations automatically whenever there is a numerical issue and a second approach which tries to eliminate as many stable pivots as possible before considering static pivoting. This last approach only performs 1×1 perturbations. Our approaches address a large range of symmetric indefinite problems and are significantly faster than approaches only based on numerical pivoting.

Although our results are promising, the precision of the solution and the number of steps of iterative refinement can still be improved. In our future work, we would like to include 2×2 perturbations of the first approach in the second one. Hence, it might be useful to discover dynamically patterns of 2×2 pivots during the factorization and to have a criterion to decide between 1×1 or 2×2 perturbations.

Furthermore we have noticed that the second approach was not robust if the threshold using for numerical pivoting is too far from the one used in static pivoting. To overcome this problem we want to develop pivoting strategies that smooth the switching from numerical to static pivoting.

Structurally unsymmetric matrices

The second part of this thesis has dealt with ordering for structurally unsymmetric matrices. We have designed an ordering approach whose originality is to compute simultaneously a row and a column permutation with the following goals in mind: to reduce the fill-in in the factors, and to preselect numerically good pivots for the factorization. We have shown that it generates factors that are sparser than existing approaches and that it speeds-up the factorization. We have proposed different parameterizations both in the context of a solver that performs numerical pivoting and in the context of a solver that performs static pivoting.

We have designed a preprocessing based on a doubly stochastic scaling. Nevertheless, we have identified that it is less robust because of the low convergence rate of the iterative process involved. That is why one of our targets is to accelerate the convergence toward doubly stochastic matrices.

We also plan to include functionalities such as detection of quasi-dense variables which would accelerate our ordering and/or enable us to work with larger sets of off-diagonal entries. Note that we have mentioned the huge effect of these improvements on our ordering compared to the classical orderings.

Furthermore, since the constraint matrix contains information that can be seen as an incomplete factorization, it is our intention to use it as a preconditioner and to compare its quality and cost with those of existing incomplete LU factorizations.

Scheduling

In the last part of this thesis we have studied dynamic scheduling strategies adapted to a general distributed memory sparse solver, `MUMPS`. In chapter 5, we have proposed scheduling algorithms designed to address the problem of heterogeneous architectures and shown that the performance with respect to computation time can be improved on networks of SMPs.

Chapter 6 has presented a hybrid approach to dynamic scheduling which takes into account information about workload and memory in the context of the parallel LU factorization. We have shown significant improvements in the memory estimation of the analysis, in the memory usage during the factorization and also in the execution time.

We have mentioned that the hybrid scheduling has to be adapted to the symmetric case and to clusters of SMPs. Thanks to our hybrid approach some issues (in particular the memory estimation) which prevented the static part of the scheduling from creating parallelism do no more exist. That is why some algorithmic constraints can now be more relaxed and we expect to improve the scalability.

Appendix

Appendix A

CMLS: data structure and implementation

In this appendix, we describe our choices with respect to the implementation of the bipartite graph $\mathbf{C}$. They are strongly connected to the functionalities that need to be provided :

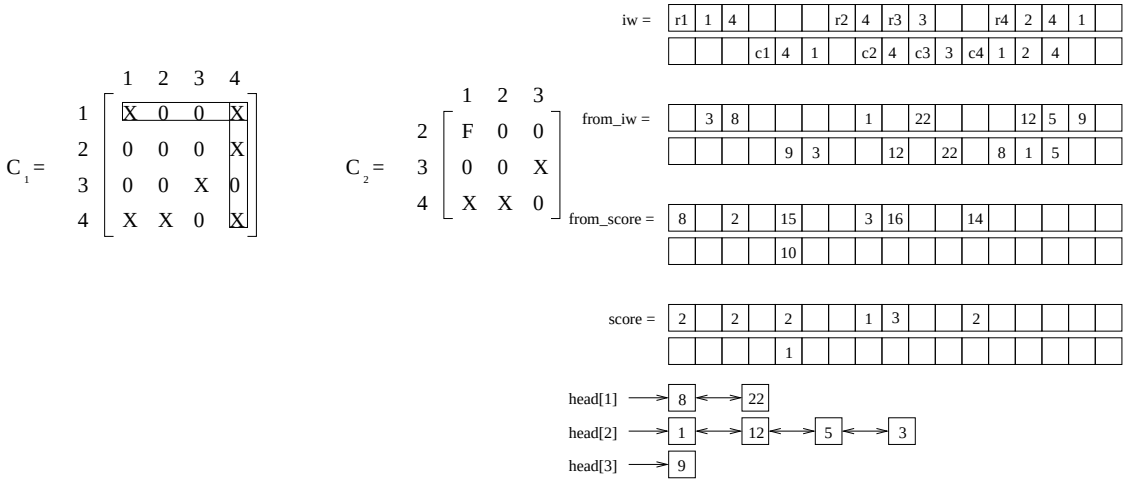
- knowledge of the exact structure of $\mathbf{C}$ and of the metric value of each nonzero entry,
- selection of a pivot in $\mathbf{C}$,
- update of the structure of each row and column and update of the metric of each entry, and finally
- compression of free space.

We will illustrate our implementation on the small example of Figure A.1.1 in which $\mathbf{C}_1$ represents the constraint matrix before the elimination of pivot $(1, 4)$ and $\mathbf{C}_2$ the constraint matrix after elimination. Note that here, and in the remainder of this section, we will use also the matrix representation of the bipartite graph of $\mathbf{C}$ (as in Figure A.1.1.a) and refer to it as the $\mathbf{C}$ matrix. When the p^{th} pivot $(rowp, colp)$ is eliminated, U_p, L_p, $\overline{U_p}$ and $\overline{L_p}$ will represent R^p_{rowp}, C^p_{colp}, $R^p_{rowp} \setminus \{colp\}$ and $C^p_{colp} \setminus \{rowp\}$ respectively.

A.1 Bipartite graph and metric

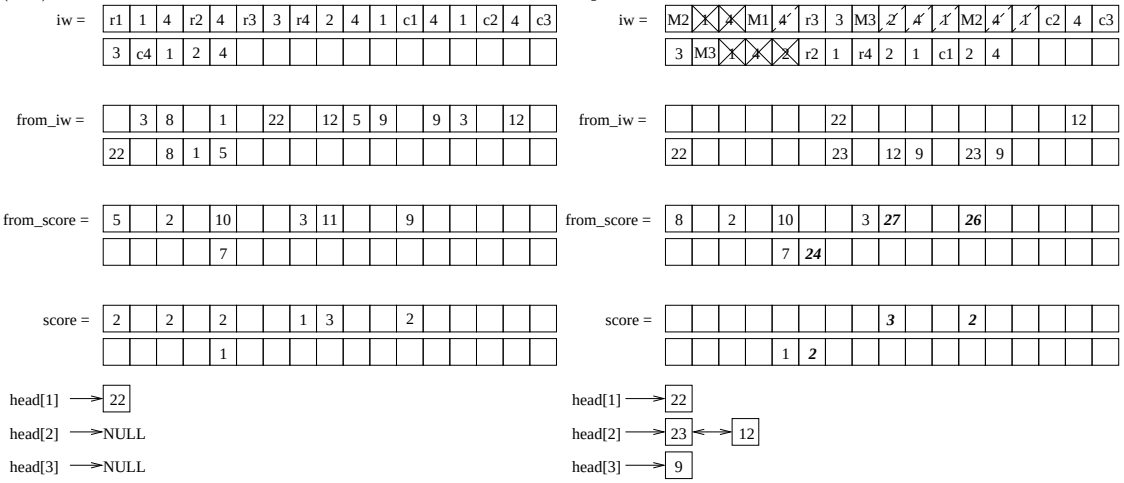
A standard bipartite graph representation will allow us to represent the exact structure of $\mathbf{C}$, and to access the row and column structures of $\mathbf{C}$.

We now discuss why we want to store the metric of each entry in $\mathbf{C}$. If we were only interested in a Markowitz cost then we could have stored the row degree and the column degree separately and then computed the metric when it is needed. The framework of our implementation is more general and can address more complicated metrics such as approximate Markowitz cost, approximate minimum fill-in, and hybrid metrics involving both structural and numerical criteria. Note that, even in the case of our approximate Markowitz cost, we have shown that the approximation strongly depends on the local



(A.1.1.a) $\mathbf{C}$ before ($\mathbf{C}_1$) and after ($\mathbf{C}_2$) elimination of pivot $(1,4)$.

(A.1.1.b) Example of $\mathbf{C}_1$ weighted bipartite graph implementation.



(A.1.1.c) After compression: $\mathbf{C}_1$ weighted bipartite graph implementation.

(A.1.1.d) After pivot elimination: $\mathbf{C}_2$ weighted bipartite graph implementation.

Figure A.1.1: Example of implementation of $\mathbf{C}$: data structure and functionalities.

symmetrization. In other words (see Algorithm 4.6.6), we cannot use the approximation of the row degree of an entry (i, j_1) to compute the real approximate Markowitz cost of an entry (i, j_2), $j_2 \neq j_1$. For example , if an element, $e_1 \notin \mathcal{R}_i$, belongs to $\mathcal{C}_{j_1}$, because of local symmetrization U_{e_1} must be considered in the approximate degree of row i (local symmetrization will occur if (i, j_1) is selected as pivot). But if $e_1 \notin \mathcal{C}_{j_2}$ then U_{e_1} need not be considered to compute the approximate degree of row i.

Three arrays $colptr$, $rowptr$ and iw are used to represent the structure of $\mathbf{C}$. The array $rowptr$ gives the position of the beginning of each row adjacency in the array iw. The array $colptr$ gives the position of the beginning of each column adjacency in the array iw. Furthermore we add flags at the beginning of each rows and columns (see array iw in Figure A.1.2). We will show in this section how these flags are used to accelerate the pivot selection and the compression of the iw array.

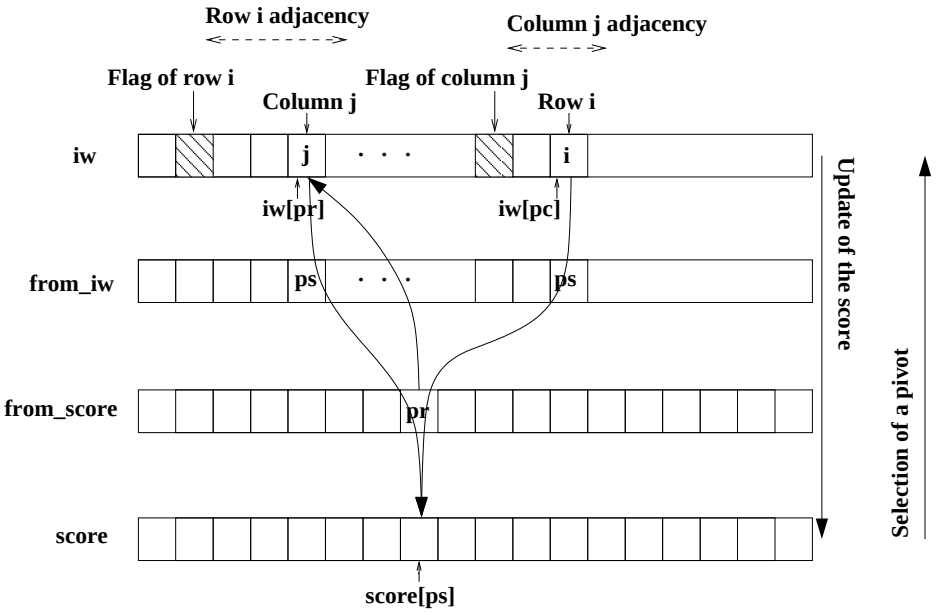


Figure A.1.2: $\mathbf{C}$: storage and functions.

This is illustrated in Figure A.1.1.b where we show how $\mathbf{C}_1$ entries are stored in the array iw with $rowptr = [2, 8, 10, 14]$ and $colptr = [22, 26, 28, 30]$. Moreover, we also see that the beginning of each row i (resp. column j) is marked with a flag r_i (resp. c_j) that indicates that it is the beginning of the adjacency list of row i (resp. column j).

The metric of each entry in $\mathbf{C}$ can be accessed using the information stored in an array $score$ (see Figures A.1.1 and A.1.2). Note that, depending on the strategies used, the array $score$ can be of more than one dimension. For example, with a hybrid strategy with one threshold we have $score[i] = (score_1[i], score_2[i])$ where $score_1[i]$ represents a structural metric, and $score_2[i]$ represents a numerical metric. In order to accelerate the pivot selection process, a doubly linked list is used to store the indices with the same first coordinate (structural metric) in the $score$ array. An array of pointers called $head$ stores the address of the beginning of each list.

Moreover, each entry (i, j) in $\mathbf{C}$ is associated with two entries in the iw array (one in

position p_r in the structure of row i and one position p_c in the structure of column j) and one in position p_s in the $score$ array. We keep the link between these positions thanks to two arrays called $from\_iw$ and $from\_score$ which satisfy the following equalities:

$$from\_iw[p_r] = from\_iw[p_c] = p_s \quad \text{(A.1.1)}$$

$$\text{and } from\_score[p_s] = p_r. \quad \text{(A.1.2)}$$

We then define two functions $\mathbf{f_1}$ and $\mathbf{f_2} = \mathbf{f_1}^{-1}$ such that $\mathbf{f_1}(p_s) = (i, j)$. Given a position p_s in the score array, $\mathbf{f_1}$ computes its associated entry (i, j). Given an entry (i, j), $\mathbf{f_2}$ returns its associated position p_s in the $score$ array. In practice, the array $from\_score$ is used to implement $\mathbf{f_1}$ and the array $from\_iw$ is used to implement $\mathbf{f_2}$. Details on the computation of these functions will be given in the following sections.

A.2 Selection of a pivot

Given the position p_s of a "good" pivot (for example $= head[mincost]$ with a pure structural strategy) in $score$, we need to compute the indices of the entry (i, j) in $\mathbf{C}$ such that $(i, j) = \mathbf{f_1}(p_s)$. In our implementation, we first compute the position $p_r = from\_score[p_s]$ from which we deduce j ($j = iw[p_r]$). The first flag on the left of p_r in iw is then used to determine i. This process is illustrated in Figure A.1.2 with a bottom-up reading.

For example, $head[1]$ of Figure A.1.1.b points to the doubly linked list $8 \leftrightarrow 22$. It means that the entries whose score values are stored in positions 8 and 22 of the $score$ array have a structural metric equal to 1. Let us consider the entry stored in position $p_s = 8$ of the $score$ array which minimizes the structural metric. $from\_score[8] = 3 = p_r$ gives the position in the iw array of the corresponding $\mathbf{C}_1$ entry. Then $iw[3] = 4$ gives the column index of this entry. To find the row of this entry, we follow iw in decreasing order until we visit a row flag. Hence we know that the entry $(1, 4)$ of $\mathbf{C}_1$ is stored in position 8 of the $score$ array.

A.2.1 Structural selection

The selection of a pivot with a structural strategy is straightforward: the first visited pivot of minimum structural metric is chosen. Thanks to the doubly linked lists and the array $head$, the access to this entry is direct using the address of $head[mincost]$ where $mincost$ is the minimum value in the $score$ array. The corresponding entry represents a minimum of the m_0 metric.

A.2.2 Hybrid pivot selection

In the hybrid case, the selection of a pivot consists in finding the best pivot according to a hybrid metric within a set of pivots that are good with respect to a structural metric.

Algorithm 1.2.1 Pivot selection($th, MS, NCOL$), hybrid strategy with one threshold.

Let $p = (rowp, colp)$ be an entry of minimum structural metric m_0 in $\mathbf{C}$.
Set $valbest \leftarrow |c_{rowp,colp}|/||c_{.colp}||_\infty$.
if $valbest \geq th$ **then**
  Accept pivot p and **return**
end if
$i = 2$
/* **Loop 1:** **examine the MS first best pivots with respect to m_0 and stop as soon as a numerically acceptable pivot is found.** */
while $i \leq MS$ **do**
  Let $p = (row_i, col_i)$ the i^{th} entry of minimum structural metric in $\mathbf{C}$.
  Set $val \leftarrow |c_{row_i,col_i}|/||c_{.col_i}||_\infty$.
  if $val \geq th$ **then**
    $p \leftarrow (row_i, col_i)$, $valbest \leftarrow val$
    Accept pivot p and **return**
  end if
  $i \leftarrow i + 1$
end while
/* **Loop 2:** **find the best numerically acceptable structural pivot.** */
$best\_metric = +\infty$, $i = 1$.
while $i \leq min(MS, NCOL)$ **do**
  Let $p = (row_i, col_i)$ be the i^{th} entry of minimum structural metric m_0.
  if col_i never visited in Loop 2 **then**
    for each entry (l, col_i) in $\mathbf{C}$ **do**
      if $|c_{l,col_i}|/||c_{.col_i}||_\infty \geq th$ **then**
        if $m_0(l, col_i) < best\_metric$ **then**
          $p \leftarrow (l, col_i)$, $best\_metric \leftarrow m_0(l, col_i)$
        end if
      end if
    end for
  end if
  $i = i + 1$
end while
Accept pivot p and **return**

As explained in Section 4.6.3, to bound the complexity of the hybrid strategies, the search is limited to the subset S of the $\mathbf{C}^k$ entries at step k. Two threshold parameters MS and $NCOL$ are introduced to define S and the ordered relation $<_{th}$ is used to select the best entry p in S (see Section 4.6.3). We recall here equation (4.6.1) that characterizes the hybrid strategy with one threshold.

$$p = arg \quad \min_{\leq_{th}}(i, j, c_{ij}/||c_{.j}||_\infty) \\ (i, j) \in S \tag{A.2.1}$$

Algorithm 1.2.1 exploits Remark 6.1 (see Section 4.6.3), to stop the search as soon as a numerically acceptable pivot is found in the first MS entries of S (Loop 1 of the algorithm).

The infinite norm of the columns is held in an array of size n and updated as explained in Section A.4.2. The choice of a relative threshold based on column norms (instead of the row norms) is only motivated by our choice of data structures.

It may happen that all the MS pivots are not acceptable. In this case, we visit a fixed number of columns (maximum $NCOL \geq 0$) of the entries with a small structural metric. We visit the column because we have a direct access to the column number thanks to the $from\_score$ values. If $NCOL > 0$, the best structural pivot which satisfies the numerical constraint among the entries of these columns is accepted. Such a pivot exists because the set of acceptable pivots for a column is non-empty (see Remark 6.2). Indeed, at least the entry which corresponds to the maximum in absolute value verifies the numerical criterion. If $NCOL = 0$ and no acceptable pivots have been found, the pivot of minimum structural metric is selected (it is a minimum with respect to equation (A.2.1), see Remark 6.3) even if it does not satisfy the numerical criterion.

For the *hybrid selection with two thresholds*, the selection is based on the relation $\leq_{th}^{absth}$ applied to $(i, j, c_{ij}/||c_{.j}||_\infty, c_{ij})$ for each entry in S. The implementation of the hybrid selection with two thresholds differs in the first loop because now a pivot needs to satisfy both $c_{ij}/||c_{.j}|| \geq th$ and $|c_{ij}| > absth$ to be numerically acceptable. During this first loop, we also store the first entry $p_{rel} = (i, j)$ such that $|c_{ij}| \geq th \times ||c_{.j}||_\infty$ since it might not be visited in the second loop and could correspond to a minimum when none of the entries visited in the second loop satisfies both numerical criteria (see Remark 6.5). When none of the entries in the MS-set is numerically acceptable, the second loop computes the best entry with respect to $\leq_{th}^{absth}$ and compares it to entry p_{rel} to find the minimum on the complete set S (see Remark 6.4).

A.3 Compression

Compression of the bipartite graph related arrays (iw and $from\_iw$) needs to be done when there is not enough place to perform the updates. It is difficult to change the doubly linked lists during the compression of the data in an efficient way. Therefore, we want to preserve the linked lists during the compression and to keep the position of the metric of an entry in the $score$ array. Hence during the compression, indices in iw are moved to the left and we have to keep the link between a new position and its position in $score$ which does not change. The two arrays $from\_iw$ and $from\_score$ are used for this purpose.

During compression both arrays iw and $from\_iw$ are compressed in the same way and only the values in $from\_score$ change. This can be done in-place and in linear time. Figure A.3.1 shows the effect of a compression on the different arrays representing the structure of $\mathbf{C}$. The memory needed for Area 1 is reduced and the size of contiguous free memory increases (Area 2).

Note that, during the update of $\mathbf{C}$, memory flags are inserted (see explanation in Section A.4). These flags enable us to skip entries in iw and to accelerate the compression. For example, we can skip in Figure A.1.1.d the i barred entries ($\times$) after each memory flag M_i.

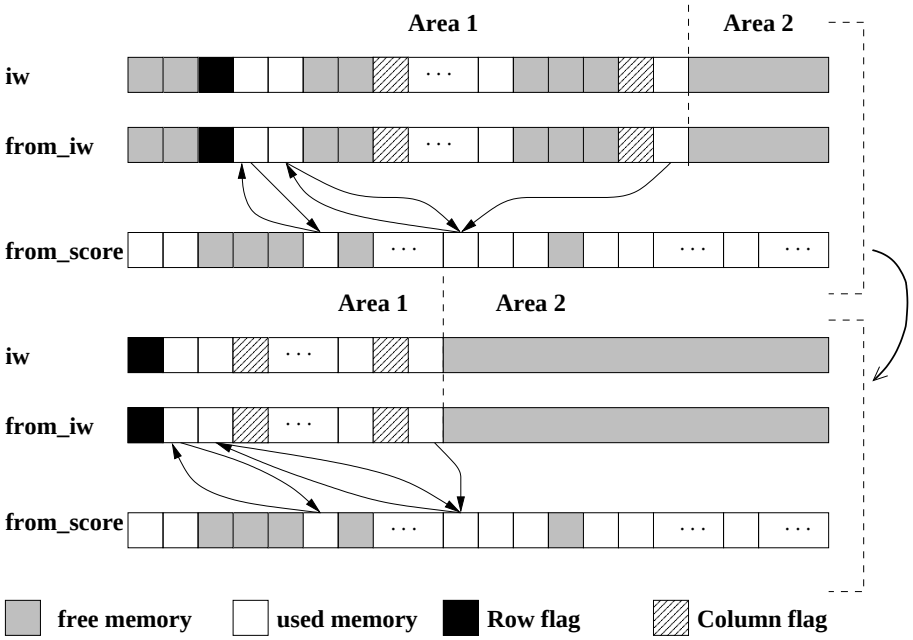


Figure A.3.1: $\mathbf{C}$: Compression and areas.

A.4 Updating the bipartite graph $\mathbf{C}$

In Algorithm 4.6.3, we underlined the fact that two passes are required to update the bipartite graph $\mathbf{C}$. Firstly new entries are added to $\mathbf{C}$ and, in the context of a hybrid metric algorithm, numerical values are updated. Secondly the structural metrics are updated. How $\mathbf{C}$ is updated depends on the choice of the structural strategy and on the memory available. In Algorithm 1.4.2, we describe the implementation of Algorithm 4.6.4 and explain how we estimate the memory needed. If the memory needed is available in Area 2 of Figure A.3.1, or if it is available after one compression, then TOTALUPDATE can be performed. Otherwise MATCHUPDATE must be done.

When hybrid metrics are used, updates (with TOTALUPDATE strategy) could be avoided when the magnitude of the update is small. Furthermore, entries in $\mathbf{C}$ can also be dropped when they become too small.

To compute the updates in one pass, we use a buffer memory of fixed size n. This buffer must be greater than $|U_p \setminus colp| \times |L_p \setminus rowp| = (|U_p| - 1) \times (|L_p| - 1)$ to complete a TOTALUPDATE (see Section A.4.2 for details on the use of this buffer).

When an update begins, it must be completed, since if it fails the cost of backtracking is expensive. The choice of type of update depends on the available memory, which can be compared to the following easy-to-compute upper bound of the space needed:

$$2\,|U_p \setminus colp||L_p \setminus rowp| + \sum_{k \in U_p \setminus colp} |C_k \setminus rowp| + \sum_{k \in L_p \setminus rowp} |R_k \setminus colp|. \qquad \text{(A.4.1)}$$

The first term of this estimation corresponds to the Schur complement. The second term corresponds to the entries of the shaded columns of Figure A.4.1. The third term corresponds to the entries of the shaded rows of Figure A.4.1.

Algorithm 1.4.2 Update of $\mathbf{C}$ (`UpdateStrategy`, `NumericalStrategy`) (Implementation of Algorithm 4.6.4)

Let $(rowp, colp)$ be the pivot selected at step k, $U_p = R_{rowp}$ and $L_p = C_{colp}$
`EnoughMemory` = true
if $(|U_p| - 1) \times (|L_p| - 1) >$ *Buffer Size* **then**
  `EnoughMemory` = false /* not enough space in the buffer */
end if
Let space_needed be an upper bound of the real space needed to do all the update with at most one compression (see for example Equation A.4.1).
if memory available $<$ space_needed **then**
  `EnoughMemory` = false /* not enough space in the bipartite graph structure */
end if
if ((`UpdateStrategy` == MATCHUPDATE) **or** (`Not EnoughMemory`)) **then**
  perform MATCHUPDATE and set $\mathcal{S} = \{(i,j) \in \mathbf{C}^{k+1} \cap (L_p \times U_p)\}$.
else
  Compress if necessary.
  perform TOTALUPDATE and set $\mathcal{S} = L_p \times U_p$.
end if
if (`NumericalStrategy`==*hybrid strategy*) **then**
  for all $(i,j) \in \mathcal{S}$ **do**
    $c_{ij}^{(k+1)} = c_{ij}^{(k)} - \frac{c_{i\ colp}^{(k)} c_{rowp\ j}^{(k)}}{c_{rowp\ colp}^{(k)}}$
  end for
end if

This bound corresponds to the worst case: the block corresponding to the Schur complement was a zero block before the elimination of pivot p. If this upper bound is respected then the update can be done with only one compression of the data structure (see Algorithm 1.4.4). The complexity of this phase clearly depends on the data structure. This will be discussed in Section A.4.3.

During the second pass of Algorithm 4.6.3, the structural metric of $\mathbf{C}$ entries which are either in $\mathcal{U}_p$ columns or in $\mathcal{L}_p$ rows is updated. Once the structural metric of (i,j) is computed we store it at position $p_s = \mathbf{f_2}(i,j)$ of the array $score$. If $i \in \mathcal{L}_p$ and $j \notin \mathcal{U}_p$ then the new value of $m_0(i,j)$ will be stored in the array $score$ while accessing the rows of $\mathbf{C}$ which belong to $\mathcal{L}_p$. If $i \notin \mathcal{L}_p$ and $j \in \mathcal{U}_p$ then the new value of $m_0(i,j)$ will be stored in the array $score$ while accessing the columns of $\mathbf{C}$ which belong to $\mathcal{U}_p$. This explains why we want to maintain the relation $from\_iw[p_r] = from\_iw[p_c] = p_s$. This process is illustrated in Figure A.1.2 by a top-down reading. More precisely, as illustrated in Figure A.1.1, let us suppose that $\mathcal{U}_p = \{1\}$ and $\mathcal{L}_p = \{2,4\}$ and that the pivot entry $(1,4)$ of $\mathbf{C}_1$ is eliminated. We also assume that the $\mathbf{C}$ related arrays iw, $from\_iw$ and $from\_score$ have been updated. We first access row 2 and find entry $(2,1)$ in position 24 of iw. $from\_iw[24] = 23$ and we store the metric of $(2,1)$ in position 23 of $score$ ($score[23] \leftarrow 2$) and we add 23 to the doubly linked list which begins at $head[2]$.

A.4.1 MATCHUPDATE implementation

Algorithm 1.4.3 describes how the coherence between the row and the column descriptions is maintained during MATCHUPDATE.

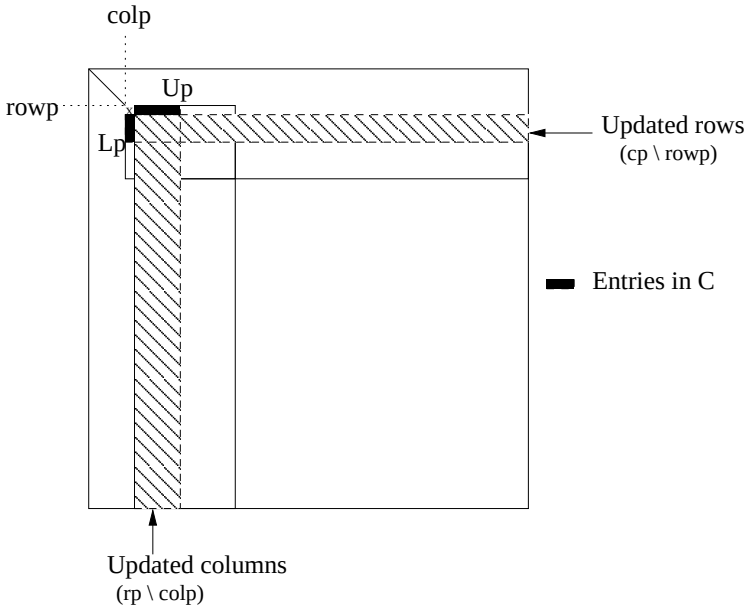


Figure A.4.1: $\mathbf{C}$: Structure update.

If an entry is added (because of Property 4.3.1), we arbitrarily take the free $score$ position corresponding to the old matching entry in the row of the pivot (we could have chosen the column of the pivot). Note that when a pivot is eliminated, all the $score$ positions associated with the entries in its row and column are freed. This data manipulation is included in the line "Suppress all entries in row $rowp$ or in column $colp$" of Algorithm 1.4.3. These keys may be required later because of the growth of $\mathbf{C}$. Moreover, memory flags are used to indicate the size of the new free blocks in the array iw. As mentioned before, this enables us to accelerate the compression process and to accelerate the search of free $score$ positions during TOTALUPDATE (more free keys are available).

Algorithm 1.4.3 Data management of $\mathbf{C}$ related data during a MATCHUPDATE.

Suppress the information related to all entries in row $rowp$ or in column $colp$ from $from\_iw$, $from\_score$, $score$ and iw (optionally do numerical dropping in hybrid strategies) and add memory flags.
Let $(rowp, j_1)$ and $(i_1, colp) \in \mathcal{M}$.
if $j_1 \neq colp$ **then**
/* $(rowp, colp) \notin \mathcal{M}$ and $(i_1, j_1) \notin \mathcal{M}$ */
$p_s \leftarrow \mathbf{f_2}(rowp, j_1)$. /* it is an unused entry in score array because entry $(rowp, j_1)$ has been removed */
Add entry (i_1, j_1) in $\mathbf{C}$ (compute its value if necessary) : set p_r and p_c.
$from\_iw[p_r] = from\_iw[p_c] = p_s$.
$from\_score[p_s] = p_r$.
end if

A.4.2 TOTALUPDATE implementation

During a TOTALUPDATE the rows and the columns which are modified are copied at the beginning of Area 2 in Figure A.3.1. Algorithm 1.4.4 shows that we first update the row structure and then the column structure.

Algorithm 1.4.4 Data management during $\mathbf{C}$ TOTALUPDATE.

$posbuf \leftarrow 1$ /* Initialize the writing position in the buffer */
for each row $k \in \overline{L_p}$ **do** /* **Loop on rows** */
  for each column $l \in R_k \setminus U_p$ **do** /* **Part 1** */
    Let p_r^{old} be the old position in iw of (k,l).
    Set p_r^{new} to the new position in iw of (k,l).
    $p_s = from\_iw[p_r^{old}]$; $from\_iw[p_r^{new}] \leftarrow p_s$; $from\_score[p_s] \leftarrow p_r^{new}$.
  end for
  for each column $l \in \overline{U_p}$ **do** /* **Part 2** */
    Set p_r^{new} to the new position in iw of (k,l).
    if (k,l) is a new entry in $\mathbf{C}$ **then**
      Find an unused position p_s in score array.
    else
      Let p_r^{old} be the old position in iw of (k,l).
      $p_s \leftarrow from\_iw[p_r^{old}]$.
    end if
    $from\_iw[p_r^{new}] \leftarrow p_s$; $from\_score[p_s] \leftarrow p_r^{new}$
    $buffer[posbuf] \leftarrow p_s$; $posbuf \leftarrow posbuf + 1$
    With the hybrid strategies: update numerical value and optionally do dropping or avoid updates, update global numerical information relative to column l and/or $\mathbf{C}$ matrix.
  end for
end for
for each column $k \in \overline{U_p}$ **do** /* **Loop on columns** */
  for each row $l \in C_k \setminus L_p$ **do** /* **Part 1** */
    Let p_c^{old} be the old position in iw of (l,k).
    Set p_c^{new} to the new position in iw of (l,k).
    $p_s = from\_iw[p_c^{old}]$; $from\_iw[p_c^{new}] \leftarrow p_s$.
  end for
  With the hybrid strategies: update global numerical information relative to column k of $\mathbf{C}$ matrix.
  for each row $l \in \overline{L_p}$ **do** /* **Part 2** */
    Set p_c^{new} to the new position in iw of (l,k).
    Find the value p_s in position $poskey$ of buffer (see Equation A.4.2).
    $from\_iw[p_c^{new}] \leftarrow p_s$.
  end for
end for

Each loop of the algorithm is split into two parts. In the first part, the columns that are not in U_p (respectively the rows that are not in L_p) are copied. In the second part, the columns that are in U_p (respectively the rows that are in L_p) are copied. During the second part of the first loop, values of p_s are computed for new entries (p_s corresponds to an unused place in the arrays $score$ and $from\_score$) and $from\_iw[p_r] = p_s$ is set. These values are temporarily saved in a buffer. During the second part of the second loop, the values of p_s are read this buffer and $from\_iw[p_c] = p_s$ is set. Since the rows and columns of L_p and U_p respectively are visited in the same order during both loops (on rows and columns), there is a relation between the place of an entry in the buffer and when

it is visited. Thus, during the column adjacency update, the values of the $score$ positions of an entry in the Schur complement of $\mathbf{C}$ can be found easily. If the entry corresponding to the i^{th} row of $\overline{L_p}$ and j^{th} column of $\overline{U_p}$ is visited, its $score$ position is stored in position

$$poskey = i \times (|L_p \setminus rowp| - 1) + j \tag{A.4.2}$$

in the buffer.

In theory a convenient size for the arrays $score$ and $from\_score$ is $sizeof(iw)/2$ (one entry in the $score$ array corresponds to one row index and one column index in the iw array). In practice, each of the arrays of Figure A.1.2 have the same length. The implementation of the structure of $\mathbf{C}$ and the initialization of the functions $\mathbf{f_1}$ and $\mathbf{f_2}$ are easier. Moreover it accelerates the search of unused places in the $score$ array for new entries: at least half of the entries in the score array are not used. The probability for finding an unused place at the first attempt is 0.5 in the worst case (when the array iw is full, half of the array $score$ is used). The average length for a search of an unused place in the worst case is $\sum_{k=1}^{\infty} k \cdot 0.5^k = 2$.

To illustrate our discussion with an example, we describe how to proceed from the compressed structure of $\mathbf{C}_1$ in Figure A.1.1.c to the structure of $\mathbf{C}_2$ in Figure A.1.1.d. Firstly row 1 and column 4 are removed (barred position of iw). In practice we only insert memory flags at the beginning of the row and column description in iw. As mentioned before the length of the row (resp. the column) appears in these memory flags and will be used to accelerate a future compression. Row 2 is first updated and a free position in the $score$ array is found (here 23) to store the metric of the fill-in corresponding to entry $(2,1)$. This value is saved: $buffer[1] \leftarrow 23$. Then we process row 4. Firstly entry $(4,2)$ is added during Part 1 of the algorithm and then entry $(4,1)$ which belongs to the bi-clique is copied during Part 2 of the row loop and $buffer[2]$ is set to 9. Finally column 1 is updated: first for entry $(2,1)$, $iw[29] \leftarrow 2$ and $from\_iw[29] \leftarrow buffer[1]$, then for entry $(4,1)$, $iw[30] \leftarrow 4$ and $from\_iw[30] \leftarrow buffer[2]$.

A.4.3 Complexity

In the worst case in terms of complexity, we perform a TOTALUPDATE at each step. The upper bound of the space needed to perform a TOTALUPDATE (quantity of equation (A.4.1)) can be computed in $\mathcal{O}(|U_p| + |L_p|)$. The first part of the loop on $\overline{L_p}$ rows ($R_k \leftarrow R_k \setminus U_p$) does at most $|R_k|$ copies. The second part performs $|\overline{U_p}|$ copies and searches at most $|\overline{U_p}|$ free positions. We approximate the cost of searching for a free position by 2 (see the worst case computation in Section A.4) The complexity of the first loop is thus in $\mathcal{O}(|L_p|(|U_p| + k^p))$ where k^p was defined as the densest row/column in Section 4.5. For the second loop, the complexity is $\mathcal{O}(|U_p|(|L_p| + k^p))$. Furthermore $|L_p||U_p|$ is bounded by the size of the buffer which is set to $\mathcal{O}(n)$. The worst cost of a TOTALUPDATE is thus $\mathcal{O}(nk^p)$. The total complexity of $\mathbf{C}$ update in `CMLS` is then $\mathcal{O}(n \sum_{k=1}^{n} k^p)$. In practice, because the size of $\mathbf{C}$ is in $\mathcal{O}(n)$ and because of memory constraints (the maximum number of nonzeros in rows and columns is bounded by a constant) the total complexity of our algorithm to update $\mathbf{C}$ is $\mathcal{O}(n^2)$. In the above analysis we neglected the compression complexity. In practice the compression does not occur at every step and does not dominate the update phase complexity.

Moreover the number of compressions can be artificially bounded by involving more often MATCHUPDATE when the working space is very limited.

Appendix B

CMLS: experimental results

This appendix presents exhaustive results. The matrices have been described in Tables 4.7.1 and in Section 4.7.1. To avoid effects of tie-breaking, we systematically apply a row and a column random permutation. We ran each problem 5 times and present in this appendix the execution whose ordering returns the median fill-in in the factors. The average gains correspond to either the average gain on Set 1 or to the average gain on Set 2. When not explicitly mentioned, the `CMLS` default parameters of Section 4.7.1.3 are used.

`Diag` refers to the version that initializes $\mathbf{C}$ with the entries of the `MC64` matching.

`MAT` refers to the strategy that performs only MATCHUPDATE.

`TOT` refers to the strategy that performs TOTALUPDATE if we are sure that it requires less than one compression (see the computation of the $spaceneeded$ quantity in equation (A.4.1)).

`LIM` refers to the strategy that performs `TOT` strategy until a certain number of entries have been added to $\mathbf{C}$ and then performs MATCHUPDATE.

B.1 Structural strategies

	CMLS							
$mindrop =$	—	0.1			0.9		0.99	
Matrix	Diag	MAT	LIM	TOT	MAT	LIM	LIM	DMLS
Set 1								
av41092	8454	8171	8033	8056	8108	8095	8101	9448
g7jac140sc	22380	18284	16080	17065	18998	18872	18231	24138
g7jac120sc	18433	14606	14246	14547	14521	15336	14898	19271
jan99jac120sc	3660	3623	3535	3520	3587	3462	3469	3849
jan99jac100sc	3080	3038	2810	2852	2915	2860	2831	3134
bayer10	425	395	365	361	383	358	366	413
bayer04	706	547	514	516	535	527	534	627
lhr34c	6374	7628	8296	7775	7610	7879	7784	6979
lhr71c	13451	16791	16663	16506	15661	17601	18194	15450
mark3jac120sc	12428	13975	14109	13968	13971	14531	13744	13674
mark3jac140sc	14776	16125	16793	17142	16146	16458	16844	16372
sinc18	32454	29842	31581	31118	31112	31529	34300	41445
sinc15	15218	14676	14860	14664	16154	15400	16645	17446
Zhao2	13008	11892	11128	11102	11863	11233	10904	14334
fd18	1053	568	569	569	594	584	594	1082
fd15	648	349	349	346	357	354	366	655
mult_dcop_03	502	1018	991	1021	1059	1062	1059	895
mult_dcop_02	439	762	761	804	735	790	806	897
onetone1	3082	3496	3177	3310	3599	3109	2829	3215
poli_large	33	33	33	33	33	33	33	33
bbmat	44745	39785	39034	38548	41405	43277	42580	53596
Avg. gain	7.5%	19.7%	22.3%	22.1%	18.6%	19.2%	18.9%	
Set 2								
shermanACb	398	364	366	362	365	363	364	399
rim	5877	5993	5907	5708	6060	5947	6006	5957
onetone2	1399	1611	1269	1280	1547	1250	1281	1396
shyy161	3732	3746	3774	3737	3702	3727	3757	4145
circuit_3	60	59	59	59	59	60	60	60
epb2	2027	2053	1902	1929	2045	2073	2050	2129
epb3	6864	6655	5975	5986	6874	6748	6711	7112
circuit_4	450	441	441	441	441	441	442	464
e40r0100	1986	1895	1750	1751	1904	1738	1747	2113
ns3Da	17149	17146	16918	17040	16883	17082	17111	16998
ecl32	32665	36717	35643	36484	33568	33758	32964	33354
Zhao1	5877	6192	6214	6294	6249	6220	6144	5891
af23560	11326	11405	11637	11490	11388	11216	11316	10913
3D_28984	11919	14008	14017	13895	12330	12362	12292	11971
3D_51448	32491	39847	38694	38414	34002	33535	33482	32647
ibm_matr	32582	39588	38492	38369	34090	34015	33595	33202
2D_54019	7714	8431	8432	8524	7714	7688	7729	7614
2D_27628	3021	3500	3529	3545	3101	3080	3068	3043
sme3Da	3951	3913	3862	3921	3824	3906	3784	3848
Avg. gain	1.5%	−3.0%	0.2%	0.1%	0.7%	2.5%	2.6%	

Table B.1.1: Number of nonzeros in the factors predicted by the analysis (in thousands of reals).

	CMLS							
$mindrop =$	—	0.1			0.9		0.99	
Matrix	Diag	MAT	LIM	TOT	MAT	LIM	LIM	DMLS
Set 1								
av41092	8687	8417	8224	8299	8298	8302	8330	9654
g7jac140sc	22385	18413	16252	17540	19006	19039	18277	24146
g7jac120sc	18438	14701	14359	14894	14536	15527	14938	19277
jan99jac120sc	3668	3625	3539	3527	3593	3465	3472	3854
jan99jac100sc	3090	3046	2812	2854	2921	2863	2835	3139
bayer10	425	396	366	363	384	360	367	414
bayer04	707	549	520	523	535	528	536	627
lhr34c	6523	8066	8696	8155	8023	8299	8179	7122
lhr71c	13676	17680	17573	17404	16543	18706	19104	15824
mark3jac120sc	12449	14019	14184	14046	14034	14603	13818	13708
mark3jac140sc	14808	16208	16879	17228	16213	16525	16949	16410
sinc18	32598	30631	32334	31916	31330	31749	34466	41563
sinc15	15270	15011	15158	15052	16347	15588	16808	17573
Zhao2	13370	12167	11453	11423	12158	11580	11296	14674
fd18	1093	569	570	569	595	585	596	1119
fd15	673	349	349	346	357	354	368	673
mult_dcop_03	502	1018	991	1021	1059	1062	1059	844
mult_dcop_02	439	762	761	805	735	790	806	860
onetone1	3082	3496	3192	3321	3602	3109	2830	3215
poli_large	33	33	33	33	33	33	33	33
bbmat	44799	40361	39600	39541	41729	43595	42870	53641
Avg. gain	7.4%	19.4%	22.0%	21.4%	18.6%	19.1%	18.9%	
Set 2								
shermanACb	399	364	366	362	365	363	364	399
rim	5878	6010	5944	5763	6075	5972	6017	5959
onetone2	1399	1611	1270	1282	1548	1252	1281	1396
shyy161	3732	3746	3838	3809	3702	3727	3757	4145
circuit_3	60	59	59	59	59	60	60	60
epb2	2027	2053	1942	1952	2045	2073	2050	2129
epb3	6864	6655	6008	6013	6874	6748	6711	7112
circuit_4	450	442	442	442	442	442	442	464
e40r0100	1986	1896	1813	1814	1906	1802	1813	2114
ns3Da	17149	17146	16921	17040	16883	17082	17112	16998
ecl32	32665	36717	35655	36485	33568	33758	32964	33354
Zhao1	5877	6192	6214	6294	6249	6220	6144	5891
af23560	11326	11419	11660	11547	11391	11217	11316	10913
3D_28984	12049	14210	14265	14172	12456	12483	12415	12057
3D_51448	32738	40225	39136	38822	34286	33800	33859	32891
ibm_matr	32872	39950	38909	38883	34319	34243	33879	33491
2D_54019	7900	8647	8686	8751	7900	7869	7906	7836
2D_27628	3045	3558	3598	3618	3118	3094	3089	3057
sme3Da	3951	3913	3862	3922	3824	3906	3784	3848
Avg. gain	1.5%	−3.1%	−0.5%	−0.6%	0.6%	2.2%	2.4%	

Table B.1.2: Number of nonzeros in the MA41_UNS factors (in thousands of reals).

	CMLS							
$mindrop =$	—	0.1			0.9		0.99	
Matrix	Diag	MAT	LIM	TOT	MAT	LIM	LIM	DMLS
Set 1								
av41092	1.03	1.03	1.02	1.03	1.02	1.03	1.03	1.02
g7jac140sc	1.00	1.01	1.01	1.03	1.00	1.01	1.00	1.00
g7jac120sc	1.00	1.01	1.01	1.02	1.00	1.01	1.00	1.00
jan99jac120sc	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
jan99jac100sc	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
bayer10	1.00	1.00	1.00	1.01	1.00	1.01	1.00	1.00
bayer04	1.00	1.00	1.01	1.01	1.00	1.00	1.00	1.00
lhr34c	1.02	1.06	1.05	1.05	1.05	1.05	1.05	1.02
lhr71c	1.02	1.05	1.05	1.05	1.06	1.06	1.05	1.02
mark3jac120sc	1.00	1.00	1.01	1.01	1.00	1.00	1.01	1.00
mark3jac140sc	1.00	1.01	1.01	1.00	1.00	1.00	1.01	1.00
sinc18	1.00	1.03	1.02	1.03	1.01	1.01	1.00	1.00
sinc15	1.00	1.02	1.02	1.03	1.01	1.01	1.01	1.01
Zhao2	1.03	1.02	1.03	1.03	1.02	1.03	1.04	1.02
fd18	1.04	1.00	1.00	1.00	1.00	1.00	1.00	1.03
fd15	1.04	1.00	1.00	1.00	1.00	1.00	1.01	1.03
mult_dcop_03	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.94
mult_dcop_02	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.96
onetone1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
poli_large	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
bbmat	1.00	1.01	1.01	1.03	1.01	1.01	1.01	1.00
Set 2								
shermanACb	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
rim	1.00	1.00	1.01	1.01	1.00	1.00	1.00	1.00
onetone2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
shyy161	1.00	1.00	1.02	1.02	1.00	1.00	1.00	1.00
circuit_3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
epb2	1.00	1.00	1.02	1.01	1.00	1.00	1.00	1.00
epb3	1.00	1.00	1.01	1.00	1.00	1.00	1.00	1.00
circuit_4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
e40r0100	1.00	1.00	1.04	1.04	1.00	1.04	1.04	1.00
ns3Da	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
ecl32	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Zhao1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
af23560	1.00	1.00	1.00	1.01	1.00	1.00	1.00	1.00
3D_28984	1.01	1.01	1.02	1.02	1.01	1.01	1.01	1.01
3D_51448	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01
ibm_matr	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01
2D_54019	1.02	1.03	1.03	1.03	1.02	1.02	1.02	1.03
2D_27628	1.01	1.02	1.02	1.02	1.01	1.00	1.01	1.00
sme3Da	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table B.1.3: Ratio between the number of nonzeros in the MA41_UNS factors and the number of nonzeros predicted by the analysis.

	CMLS							
$mindrop =$	—	0.1			0.9		0.99	
Matrix	Diag	MAT	LIM	TOT	MAT	LIM	LIM	DMLS
Set 1								
av41092	7223	6840	6769	6679	6820	6795	6817	8098
g7jac140sc	14953	13363	12062	12134	13578	13054	12971	15204
g7jac120sc	13088	10489	10393	10142	10064	9863	9804	14486
jan99jac120sc	1902	1734	1675	1657	1765	1654	1665	1908
jan99jac100sc	1728	1435	1445	1377	1454	1422	1373	1576
bayer10	333	310	296	296	310	297	294	329
bayer04	527	449	475	461	432	482	487	538
lhr34c	4296	5162	5770	6018	5264	5803	6172	5787
lhr71c	9306	10419	11966	12611	10991	11790	11871	10847
mark3jac120sc	10169	11127	10941	11177	11180	10722	11324	10828
mark3jac140sc	12100	12822	13328	12531	13214	12906	12567	12867
sinc18	28792	25489	26162	27688	26478	26589	29100	35287
sinc15	13716	12334	12732	12659	12533	13863	13102	15284
Zhao2	11019	10512	9777	9845	10308	9932	9730	12226
fd18	931	523	531	520	529	515	542	935
fd15	558	316	315	311	316	328	336	564
mult_dcop_03	297	569	618	589	610	607	612	403
mult_dcop_02	292	391	387	392	393	388	393	403
onetone1	2484	2743	2640	2757	2944	2602	2459	2660
poli_large	33	33	33	33	33	33	33	33
bbmat	41040	40303	34813	35064	36984	36272	37426	41315
Avg. gain	7.1%	19.2%	19.7%	19.8%	18.7%	18.9%	18.0%	
Set 2								
shermanACb	375	346	349	348	343	341	344	373
rim	5420	5620	5332	5435	5486	5227	5614	5283
onetone2	1058	1297	1034	1082	1237	1036	1125	1032
shyy161	3092	3203	3198	3200	3103	3088	3080	3391
circuit_3	60	59	59	59	59	59	60	59
epb2	1976	1962	1875	1848	1991	2005	2008	2057
epb3	6325	6016	5367	5506	6372	6260	6271	6669
circuit_4	449	441	441	441	441	441	441	463
e40r0100	1970	1892	1738	1729	1890	1695	1738	2028
ns3Da	16700	16313	16475	16431	16708	16457	17033	16777
ecl32	30731	35265	35253	34633	32427	32348	32854	30174
Zhao1	5870	6088	6220	5979	6104	6049	6128	5865
af23560	11401	10923	10889	10901	11004	10798	11020	10961
3D_28984	12000	13506	13817	13889	12063	12261	12165	12025
3D_51448	32090	37807	38744	38554	32980	32925	32890	32448
ibm_matr	32185	37631	37476	38262	33474	33569	32551	32820
2D_54019	7623	8300	8315	8356	7587	7634	7632	7591
2D_27628	3021	3471	3445	3416	3057	3043	3043	2996
sme3Da	3721	3850	3806	3816	3820	3741	3744	3764
Avg. gain	0.9%	−3.2%	−0.7%	−0.9%	0.0%	2.1%	0.9%	

Table B.1.4: Number of nonzeros in the SuperLU_DIST factors (in thousands of reals).

	CMLS							
$mindrop =$	—	0.1			0.9		0.99	
Matrix	Diag	MAT	LIM	TOT	MAT	LIM	LIM	DMLS
Set 1								
av41092	2798	2650	2539	2652	2594	2725	2601	3664
g7jac140sc	26452	19656	13907	17015	21232	20110	18324	30254
g7jac120sc	21203	14959	12811	13997	14411	15856	13763	22405
jan99jac120sc	1285	1225	1187	1200	1242	1181	1165	1348
jan99jac100sc	1105	1090	918	954	975	963	932	1058
bayer10	19	17	14	14	15	14	14	18
bayer04	57	31	25	26	28	26	28	34
lhr34c	1486	2238	2546	2403	2259	2618	2377	1794
lhr71c	3376	6046	5503	5494	4841	6686	6888	5281
mark3jac120sc	6101	8368	8793	8480	8155	9328	7981	7534
mark3jac140sc	7429	9306	10618	11185	9482	10090	10836	9295
sinc18	62163	59757	64041	63273	61782	63588	69047	96970
sinc15	19195	20635	21370	20653	22586	20174	23845	24567
Zhao2	8372	7298	6471	6430	7511	6791	6376	10080
fd18	121	44	43	45	48	47	45	128
fd15	63	22	22	22	23	23	24	61
mult_dcop_03	25	170	150	161	184	189	191	106
mult_dcop_02	14	80	81	95	73	89	92	111
onetone1	1205	1673	1332	1489	1542	1185	979	1296
poli_large	0.051	0.051	0.051	0.051	0.051	0.051	0.051	0.051
bbmat	50887	40435	39553	40474	43281	47854	47213	77900
Avg. gain	15.7%	35.5%	45.0%	40.6%	35.0%	34.9%	37.3%	
Set 2								
shermanACb	25	21	21	20	20	20	21	26
rim	1144	1187	1208	1091	1233	1168	1201	1145
onetone2	240	367	234	243	333	224	237	243
shyy161	539	543	576	562	525	536	549	693
circuit_3	0.24	0.23	0.24	0.24	0.24	0.24	0.24	0.25
epb2	304	337	320	317	304	333	318	347
epb3	1111	1102	938	951	1135	1099	1071	1149
circuit_4	10	10	10	10	10	10	10	14
e40r0100	251	225	202	204	227	198	201	280
ns3Da	12916	12451	12554	12591	12392	12559	12841	12861
ecl32	35977	43505	40558	41759	37082	38127	36564	37816
Zhao1	2321	2500	2517	2584	2504	2530	2411	2267
af23560	4970	4819	5099	4982	5071	4721	4985	4576
3D_28984	5756	7394	7435	7482	6028	6041	5991	5657
3D_51448	28641	39756	38310	37417	31318	30079	30588	28911
ibm_matr	28388	39934	37495	38211	31020	31182	30453	30736
2D_54019	1430	1691	1722	1826	1433	1433	1436	1390
2D_27628	416	560	585	582	440	423	417	416
sme3Da	963	935	900	934	882	912	876	884
Avg. gain	5.4%	−2.5%	1.1%	1.2%	4.2%	7.2%	6.9%	

Table B.1.5: MA41_UNS: number of operations (in millions).

	CMLS							
$mindrop =$	—	0.1			0.9		0.99	
Matrix	Diag	MAT	LIM	TOT	MAT	LIM	LIM	DMLS
Set 1								
av41092	3.5	8.8	10.4	10.4	8.9	10.0	10.1	3.9
g7jac140sc	18.3	22.5	27.7	71.3	21.8	29.3	17.6	6.2
g7jac120sc	14.0	16.8	23.2	43.3	14.4	20.4	13.4	3.8
jan99jac120sc	5.1	6.4	7.4	7.2	6.4	6.7	7.2	2.8
jan99jac100sc	4.0	4.7	5.2	5.6	4.7	5.5	5.5	2.0
bayer10	0.3	0.3	0.5	0.5	0.4	0.4	0.4	0.1
bayer04	0.6	0.7	0.9	0.8	0.6	0.8	0.7	0.3
lhr34c	3.2	6.6	8.7	8.2	6.4	8.2	6.6	3.4
lhr71c	7.7	15.5	19.0	18.2	14.7	19.0	16.7	7.9
mark3jac120sc	7.2	9.1	10.7	10.3	9.1	10.9	10.2	3.7
mark3jac140sc	8.8	10.3	13.3	13.3	10.7	12.3	12.8	4.4
sinc18	18.1	29.6	27.3	35.3	28.5	28.2	27.6	19.7
sinc15	6.7	10.2	13.1	15.7	11.4	10.7	10.6	6.5
Zhao2	2.2	2.3	3.4	3.3	2.1	3.4	3.3	0.9
fd18	0.4	0.3	0.4	0.4	0.3	0.4	0.4	0.1
fd15	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.1
mult_dcop_03	0.5	0.6	0.7	0.6	0.6	0.6	0.6	0.4
mult_dcop_02	0.5	0.9	1.2	1.2	0.9	1.1	1.1	0.4
onetone1	1.1	1.5	2.3	2.2	1.5	1.8	1.6	0.5
poli_large	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
bbmat	46.0	51.6	57.5	115.3	48.8	49.9	43.7	17.5
Avg. gain	-38.7%	-55.0%	-62.9%	-65.6%	-54.4%	-60.8%	-57.9%	
Set 2								
shermanACb	0.2	0.3	0.3	0.3	0.3	0.3	0.2	0.1
rim	0.8	1.2	1.8	3.0	1.1	1.5	1.2	0.5
onetone2	0.6	0.7	0.8	0.8	0.8	0.8	0.7	0.3
shyy161	1.4	1.6	1.8	1.8	1.4	1.4	1.4	0.6
circuit_3	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
epb2	0.6	0.7	0.8	0.8	0.6	0.6	0.6	0.3
epb3	3.2	3.2	3.5	3.5	3.1	3.2	3.0	1.5
circuit_4	22.3	57.7	60.1	60.9	57.6	55.4	57.0	15.0
e40r0100	0.3	0.4	0.7	1.0	0.4	0.6	0.6	0.1
ns3Da	0.8	1.7	2.4	3.5	1.3	1.7	1.2	0.3
ecl32	4.6	7.0	8.4	8.4	5.1	5.1	5.3	1.3
Zhao1	1.1	1.2	1.4	1.4	1.2	1.4	1.4	0.4
af23560	2.1	4.0	8.3	20.6	2.2	2.5	2.2	0.8
3D_28984	1.4	2.3	4.6	9.8	1.9	2.1	2.3	0.3
3D_51448	2.7	5.2	7.8	8.1	4.5	4.9	4.7	0.6
ibm_matr	2.9	5.2	8.3	8.9	4.4	4.7	4.9	0.6
2D_54019	1.0	1.6	2.8	3.0	1.1	1.2	1.2	0.2
2D_27628	0.4	0.7	1.0	1.0	0.5	0.6	0.6	0.1
sme3Da	0.3	0.4	0.7	0.9	0.2	0.2	0.2	0.1
Avg. gain	-60.0%	-71.7%	-76.9%	-79.1%	-66.9%	-69.6%	-68.3%	

Table B.1.6: Ordering time (in seconds).

	CMLS							
$mindrop =$	—	0.1			0.9		0.99	
Matrix	Diag	MAT	LIM	TOT	MAT	LIM	LIM	DMLS
Set 1								
av41092	1.0e+00	9.9e-01	1.0e+00	1.0e+00	1.0e+00	9.9e-01	1.0e-00	1.0e+00
g7jac140sc	5.7e-06	1.0e+00	1.0e+00	1.0e+00	9.9e-01	1.0e+00	9.2e-01	6.4e-07
g7jac120sc	1.2e-06	9.9e-01	1.0e+00	1.0e+00	9.9e-01	1.0e+00	9.7e-01	3.5e-07
jan99jac120sc	6.0e-10	2.9e-07	2.1e-07	2.8e-03	2.6e-10	6.1e-07	9.5e-06	4.3e-09
jan99jac100sc	2.3e-09	1.1e-09	9.0e-08	9.8e-09	7.1e-08	6.0e-07	1.8e-06	4.5e-08
bayer10	1.4e-09	3.1e-06	1.7e-02	2.8e-01	9.7e-05	1.0e+00	4.7e-06	9.9e-10
bayer04	5.5e-11	9.0e-03	1.0e+00	1.0e-00	9.4e-01	9.9e-01	9.6e-01	1.5e-05
lhr34c	1.1e-02	7.5e-03	1.0e+00	9.8e-01	6.9e-04	5.2e-03	3.1e-02	8.3e-06
lhr71c	3.1e-04	1.6e-01	3.6e-01	9.9e-01	2.0e-03	1.0e+00	6.7e-01	6.2e-06
mark3jac120sc	1.8e-05	9.6e-01	8.6e-01	1.0e+00	1.0e+00	9.6e-01	9.0e-01	1.3e-03
mark3jac140sc	4.9e-04	7.7e-01	9.9e-01	9.9e-01	6.4e-04	1.0e+00	9.9e-01	2.9e-05
sinc18	1.3e-06	7.1e-01	7.9e-01	9.8e-01	3.6e-02	5.6e-01	5.2e-01	5.4e-01
sinc15	2.0e-07	1.0e-01	7.7e-01	8.5e-01	8.4e-01	9.0e-01	4.3e-02	2.4e-04
Zhao2	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00
fd18	1.2e-02	8.9e-12	1.1e-09	4.0e-02	7.5e-10	2.1e-05	1.0e-11	3.6e-08
fd15	1.5e-08	1.6e-12	5.6e-08	1.1e-05	3.0e-12	2.9e-12	2.0e-08	7.5e-07
mult_dcop_03	1.6e-14	1.6e-08	3.0e-08	4.3e-08	3.0e-11	2.6e-08	2.6e-08	2.0e-14
mult_dcop_02	5.3e-15	2.0e-08	7.7e-04	1.9e-03	1.7e-06	1.6e-05	1.6e-05	4.3e-14
onetone1	9.0e-14	4.5e-13	9.4e-08	6.2e-04	2.5e-13	4.2e-08	4.7e-13	2.8e-13
poli_large	3.2e-16	2.7e-16	3.8e-16	2.1e-16	1.9e-16	2.1e-16	3.5e-16	2.3e-16
bbmat	2.5e-01	2.7e-01	7.9e-03	9.8e-01	3.6e-04	1.2e-02	6.1e-05	1.3e-02
Set 2								
shermanACb	2.6e-13	2.3e-13	1.1e-12	3.4e-14	9.4e-15	1.5e-14	2.1e-14	1.5e-13
rim	9.0e-07	9.2e-01	9.8e-01	9.4e-01	9.2e-01	8.6e-01	5.9e-01	1.8e-05
onetone2	1.3e-13	8.4e-10	1.1e-07	2.8e-07	7.5e-10	1.2e-07	1.1e-09	8.4e-14
shyy161	5.1e-16	9.2e-11	1.0e+00	1.0e+00	5.4e-16	4.8e-16	5.5e-16	4.6e-16
circuit_3	1.3e-11	8.3e-13	5.4e-11	1.1e-12	2.6e-13	7.0e-14	1.1e-13	1.5e-11
epb2	1.8e-15	1.2e-14	1.5e-07	1.0e-07	1.3e-15	2.4e-15	1.1e-15	2.0e-15
epb3	2.0e-15	1.4e-14	4.1e-03	8.2e-04	1.7e-15	3.9e-15	1.3e-15	1.4e-15
circuit_4	1.3e-11	3.5e-12	3.3e-12	5.7e-12	3.8e-12	3.2e-12	3.1e-12	1.4e-11
e40r0100	1.4e-04	8.4e-01	9.9e-01	1.0e-00	9.7e-01	9.9e-01	9.9e-01	4.8e-12
ns3Da	2.3e-12	7.6e-11	1.4e-10	1.8e-09	7.8e-10	1.0e-08	2.1e-10	9.3e-12
ecl32	5.2e-14	9.9e-12	1.0e+00	4.6e-07	6.9e-13	5.0e-13	3.4e-13	3.1e-14
Zhao1	1.5e-15	1.6e-15	2.1e-15	2.5e-12	2.4e-15	5.2e-15	5.9e-14	1.5e-15
af23560	9.2e-13	7.9e-06	1.1e-02	8.0e-01	1.5e-08	1.0e-08	3.2e-09	4.6e-13
3D_28984	9.4e-01	1.0e+00	1.0e+00	1.0e+00	9.9e-01	1.0e-00	1.4e-01	1.0e+00
3D_51448	2.0e-15	4.2e-03	9.0e-01	1.0e+00	2.7e-06	1.2e-06	2.9e-04	1.9e-15
ibm_matr	2.3e-15	1.8e-04	9.0e-01	9.7e-01	1.3e-06	1.4e-04	7.9e-08	1.9e-15
2D_54019	7.8e-16	9.9e-01	8.7e-01	6.1e-01	1.0e+00	3.1e-02	1.6e-06	1.6e-05
2D_27628	8.1e-16	5.4e-01	9.8e-01	9.9e-01	1.1e-06	3.3e-01	8.8e-06	6.6e-16
sme3Da	3.8e-16	2.9e-13	7.3e-12	8.9e-11	4.0e-16	4.3e-16	5.4e-16	4.1e-16

Table B.1.7: SuperLU_DIST: precision without iterative refi nement.

	CMLS							
$mindrop =$	—		0.1		0.9		0.99	
Matrix	Diag	MAT	LIM	TOT	MAT	LIM	LIM	DMLS
Set 1								
av41092	1.0e+00	1.0e+00	1.0e+00	1.0e-00	1.0e+00	1.0e+00	1.0e+00	1.0e+00
g7jac140sc	5.1e-16	1.0e+00	1.0e+00	1.0e+00	9.8e-01	1.0e+00	8.4e-16	6.4e-16
g7jac120sc	5.7e-16	1.0e+00	1.0e+00	1.0e+00	9.8e-01	1.0e+00	1.9e-14	5.9e-16
jan99jac120sc	3.7e-16	4.8e-16	5.0e-16	6.5e-16	5.4e-16	4.9e-16	5.0e-16	4.6e-16
jan99jac100sc	3.3e-16	5.1e-16	3.5e-16	4.1e-16	3.8e-16	5.1e-16	3.9e-16	3.7e-16
bayer10	2.1e-16	2.1e-16	2.1e-16	4.4e-14	3.0e-16	1.5e-13	2.1e-16	2.7e-16
bayer04	2.2e-16	3.2e-15	1.0e+00	1.0e-00	3.2e-16	8.0e-01	1.0e-12	2.1e-16
lhr34c	1.4e-04	8.9e-04	9.9e-01	6.7e-01	5.1e-03	4.7e-03	3.8e-02	9.5e-14
lhr71c	4.1e-04	1.2e-01	2.1e-01	9.6e-01	3.5e-03	1.0e+00	7.3e-01	1.7e-07
mark3jac120sc	4.5e-16	1.0e+00	9.9e-16	1.0e+00	6.0e-01	9.0e-01	1.6e-15	4.3e-16
mark3jac140sc	3.9e-16	2.1e-15	6.8e-01	9.9e-01	4.1e-16	1.0e+00	1.0e+00	4.2e-16
sinc18	2.7e-16	5.5e-01	8.7e-01	8.8e-01	2.7e-14	4.7e-01	3.5e-13	7.9e-01
sinc15	2.8e-16	4.8e-15	9.7e-01	9.3e-01	6.7e-01	8.4e-01	5.2e-14	9.0e-15
Zhao2	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00
fd18	2.6e-15	2.4e-16	2.1e-16	2.5e-16	2.5e-16	2.1e-16	2.4e-16	2.1e-16
fd15	3.2e-16	2.1e-16	2.6e-16	2.7e-16	2.2e-16	2.3e-16	2.0e-16	2.6e-16
mult_dcop_03	1.8e-16	1.5e-16	1.5e-16	1.5e-16	2.3e-16	1.8e-16	1.9e-16	1.4e-16
mult_dcop_02	1.6e-16	1.9e-16	1.6e-16	2.1e-16	2.0e-16	1.5e-16	1.6e-16	2.0e-16
onetone1	4.9e-16	4.4e-16	4.6e-16	1.0e-14	5.2e-16	4.9e-16	5.1e-16	5.0e-16
poli_large	2.1e-16	2.1e-16	1.6e-16	2.1e-16	1.9e-16	2.1e-16	1.6e-16	1.0e-16
bbmat	4.7e-16	9.2e-01	6.9e-15	9.7e-01	4.7e-16	5.7e-16	5.4e-16	4.5e-16
Set 2								
shermanACb	4.5e-16	1.5e-13	4.3e-16	2.4e-16	2.3e-16	2.5e-16	1.5e-13	3.4e-16
rim	5.5e-12	1.8e-05	9.7e-01	9.9e-01	9.8e-01	8.1e-01	1.8e-05	9.0e-13
onetone2	3.7e-16	8.4e-14	4.0e-16	4.1e-16	3.0e-16	3.7e-16	8.4e-14	4.6e-16
shyy161	2.7e-16	4.6e-16	9.9e-01	1.0e+00	1.7e-15	2.7e-16	4.6e-16	2.0e-16
circuit_3	3.4e-16	1.5e-11	5.3e-16	4.1e-16	6.0e-16	6.6e-16	1.5e-11	2.5e-16
epb2	2.8e-16	2.0e-15	2.8e-16	3.4e-16	2.8e-16	2.8e-16	2.0e-15	2.6e-16
epb3	2.7e-16	1.4e-15	3.8e-16	3.4e-16	2.5e-16	2.4e-16	1.4e-15	2.8e-16
circuit_4	6.4e-15	1.4e-11	8.4e-14	1.7e-14	1.4e-14	4.4e-14	1.4e-11	1.4e-14
e40r0100	3.7e-16	4.8e-12	9.9e-01	1.0e-00	8.3e-01	9.9e-01	4.8e-12	4.1e-16
ns3Da	2.9e-16	9.3e-12	2.8e-16	3.6e-16	3.7e-16	3.8e-16	9.3e-12	3.4e-16
ecl32	3.9e-16	3.1e-14	4.7e-06	3.7e-16	3.3e-16	3.5e-16	3.1e-14	3.6e-16
Zhao1	1.9e-16	1.5e-15	2.0e-16	2.2e-16	2.2e-16	2.0e-16	1.5e-15	1.3e-16
af23560	3.3e-16	4.6e-13	3.9e-16	8.2e-01	3.1e-16	2.8e-16	4.6e-13	3.1e-16
3D_28984	3.3e-15	1.0e+00	1.0e+00	9.9e-01	2.5e-05	1.2e-07	1.0e+00	1.2e-14
3D_51448	3.1e-16	1.9e-15	9.9e-01	1.0e+00	3.3e-16	2.9e-16	1.9e-15	3.6e-16
ibm_matr	3.3e-16	1.9e-15	1.0e-00	6.1e-01	3.4e-16	3.0e-16	1.9e-15	2.7e-16
2D_54019	3.1e-16	1.6e-05	7.2e-01	1.9e-01	3.5e-16	2.7e-16	1.6e-05	3.6e-16
2D_27628	2.6e-16	6.6e-16	1.0e-00	9.9e-01	3.1e-16	2.1e-15	6.6e-16	2.9e-16
sme3Da	4.9e-16	4.1e-16	4.2e-16	4.6e-16	4.7e-16	4.1e-16	4.1e-16	6.6e-16

Table B.1.8: SuperLU_DIST: precision of the solution after iterative refi nement.

	CMLS							
$mindrop =$	—	0.1			0.9		0.99	
Matrix	Diag	MAT	LIM	TOT	MAT	LIM	LIM	DMLS
Set 1								
av41092	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}
g7jac140sc	3	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	6	3
g7jac120sc	3	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	6	3
jan99jac120sc	3	3	3	5	3	4	4	3
jan99jac100sc	3	3	3	3	3	4	4	3
bayer10	3	4	5	4	4	4	4	3
bayer04	3	6	2^{**}	2^{**}	5	2^{**}	6	4
lhr34c	4^{**}	4^{**}	2^{**}	2^{**}	2^{**}	2^{**}	3^{**}	9
lhr71c	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	3^{*}
mark3jac120sc	4	2^{**}	10	2^{**}	2^{**}	2^{**}	9	5
mark3jac140sc	4	7	3^{**}	2^{**}	6	2^{**}	2^{**}	4
sinc18	5	2^{**}	2^{**}	2^{**}	7	2^{**}	7	2^{**}
sinc15	5	8	2^{**}	2^{**}	2^{**}	2^{**}	7	6
Zhao2	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}
fd18	7	3	3	7	3	4	3	4
fd15	4	2	3	4	3	3	3	4
mult_dcop_03	2	3	3	3	3	3	2	3
mult_dcop_02	2	2	2	2	2	2	2	3
onetone1	3	3	4	6	3	4	3	3
poli_large	2	2	2	1	1	1	2	2
bbmat	11	2^{**}	9	2^{**}	6	9	5	8
Set 2								
shermanACb	3	4	4	3	3	3	4	4
rim	3	2^{*}	2^{**}	2^{**}	2^{**}	2^{**}	2	5
onetone2	3	4	4	4	4	4	3	3
shyy161	2	3	2^{**}	2^{**}	2	2	3	2
circuit_3	3	3	3	3	3	4	3	3
epb2	3	3	4	3	3	3	3	3
epb3	3	3	6	5	3	3	3	3
circuit_4	4	3	3	4	4	3	4	4
e40r0100	5	2	2^{**}	2^{**}	2^{**}	2^{**}	2	3
ns3Da	3	4	4	3	3	3	4	3
ecl32	3	3	3	4	3	3	3	3
Zhao1	2	3	3	3	3	2	3	2
af23560	3	4	8	2^{**}	4	4	3	3
3D_28984	3	2^{**}	2^{**}	2^{**}	3^{*}	3^{*}	4^{**}	3
3D_51448	3	5	2^{**}	2^{**}	4	4	4	3
ibm_matr	3	4	2^{**}	2^{**}	4	4	3	3
2D_54019	3	4^{*}	2^{**}	3^{**}	3	6	3^{*}	3
2D_27628	3	7	2^{**}	2^{**}	4	4	3	3
sme3Da	2	3	3	3	2	2	2	2

Table B.1.9: SuperLU_DIST: number of steps of iterative refi nement to get the precision of Table B.1.8. ** means that after iterative refi nement the component-wise backward error is greater than 10^{-4} and * means that after iterative refi nement the component-wise backward error is between 10^{-4} and 10^{-8}.

B.2 Hybrid strategies

	CMLS							
$mindrop =$	0.1		0.9		0.99		0.1	
Matrix/ $NCOL$ =	0	10	0	10	0	10	STR	DMLS
Set 1								
av41092	6815	6820	6797	6755	6762	6657	6769	8098
g7jac140sc	12216	13949	13207	13790	12990	13642	12062	15204
g7jac120sc	11102	11116	10203	10224	9693	9842	10393	14486
jan99jac120sc	1680	1713	1721	1686	1745	1669	1675	1908
jan99jac100sc	1377	1527	1374	1450	1376	1434	1445	1576
bayer10	314	304	293	300	308	303	296	329
bayer04	482	469	462	486	489	477	475	538
lhr34c	6238	5764	7208	5730	5621	5902	5770	5787
lhr71c	13617	12452	12188	12193	12752	12415	11966	10847
mark3jac120sc	11305	10511	11581	10903	11500	11643	10941	10828
mark3jac140sc	13656	13073	13174	12621	13245	13256	13328	12867
sinc18	28362	26518	26397	26813	28072	30447	26162	35287
sinc15	12711	12739	14002	13932	13792	13126	12732	15284
Zhao2	10403	11039	10497	10679	10453	11051	9777	12226
fd18	601	615	625	609	631	634	531	935
fd15	378	366	383	378	385	387	315	564
mult_dcop_03	648	632	626	570	621	601	618	403
mult_dcop_02	401	403	388	389	389	388	387	403
onetone1	3859	4020	2712	2709	2592	2620	2640	2660
poli_large	33	33	33	33	33	33	33	33
bbmat	35145	35991	37350	38658	34230	34855	34813	41315
Avg. gain	12.3%	12.4%	13.5%	14.2%	14.6%	13.6%	19.7%	
Set 2								
shermanACb	346	345	345	343	348	348	349	373
rim	5658	5431	5515	5565	5484	5499	5332	5283
onetone2	1276	1272	1313	1288	1322	1323	1034	1032
shyy161	3138	3171	3056	3067	3095	3050	3198	3391
circuit_3	59	59	59	59	60	60	59	59
epb2	1869	1840	1974	2001	2013	1984	1875	2057
epb3	5442	5537	6192	6259	6247	6294	5367	6669
circuit_4	444	444	441	441	441	441	441	463
e40r0100	1762	1780	1783	1765	1769	1780	1738	2028
ns3Da	16640	16697	16875	16505	16724	16845	16475	16777
ecl32	34521	34172	32882	32502	32559	31951	35253	30174
Zhao1	6170	6127	6021	6177	6096	6087	6220	5865
af23560	10844	10774	10967	10978	11054	11068	10889	10961
3D_28984	13761	13844	12270	12091	12107	12177	13817	12025
3D_51448	38417	37787	33608	33360	32953	33213	38744	32448
ibm_matr	37872	37621	33066	33382	33119	33048	37476	32820
2D_54019	8241	8276	7657	7627	7618	7610	8315	7591
2D_27628	3440	3443	3026	3026	3033	3044	3445	2996
sme3Da	3916	3811	3722	3868	3776	3777	3806	3764
Avg. gain	−2.1%	−1.7%	0.3%	0.2%	0.1%	0.1%	−0.7%	

Table B.2.1: Number of nonzeros in the SuperLU_DIST factors (in thousands of reals). STR: structural metric used to select the pivots. Otherwise we use a hybrid metric.

	CMLS							
$mindrop =$	0.1		0.9		0.99		0.1	
Matrix/ $NCOL =$	0	10	0	10	0	10	STR	DMLS
Set 1								
av41092	10.1	10.7	10.6	10.3	10.9	10.1	10.4	3.8
g7jac140sc	36.2	56.1	50.5	36.7	18.2	18.6	28.9	5.7
g7jac120sc	47.3	36.5	27.3	24.4	14.4	13.1	24.0	3.8
jan99jac120sc	7.4	7.3	7.3	7.6	7.6	7.6	7.5	2.6
jan99jac100sc	5.8	5.8	5.4	5.8	5.5	5.7	5.4	2.1
bayer10	0.5	0.5	0.5	0.5	0.5	0.4	0.4	0.1
bayer04	1.1	1.0	0.9	0.9	0.9	0.8	1.0	0.3
lhr34c	8.7	8.3	9.9	8.5	8.2	8.0	8.2	4.1
lhr71c	20.9	20.5	20.8	21.9	18.7	19.3	20.9	9.3
mark3jac120sc	10.4	10.2	10.5	10.8	11.0	10.6	10.4	3.4
mark3jac140sc	12.5	12.5	12.3	12.2	12.5	12.5	13.0	4.3
sinc18	45.2	42.5	34.3	32.1	30.6	28.2	38.0	14.6
sinc15	17.8	19.1	12.6	13.3	11.2	11.9	16.3	3.7
Zhao2	17.6	57.2	12.0	56.3	14.6	125.6	3.5	1.0
fd18	2.2	1.4	0.8	0.8	0.6	0.6	0.4	0.1
fd15	0.7	1.5	0.4	0.5	0.3	0.3	0.2	0.1
mult_dcop_03	0.7	0.8	0.7	0.7	0.7	0.6	0.7	0.4
mult_dcop_02	1.3	1.2	1.2	1.2	1.1	1.1	1.2	0.4
onetone1	15.0	15.4	2.2	2.2	1.9	2.2	2.2	0.5
poli_large	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
bbmat	62.5	57.3	39.7	45.2	38.6	40.5	55.4	13.2
Avg. gain	-71.5%	-71.8%	-67.6%	-68.0%	-64.1%	-63.9%	-64.6%	
Set 2								
shermanACb	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.1
rim	2.1	2.1	1.6	1.7	1.2	1.2	1.9	0.4
onetone2	1.0	1.1	1.0	1.0	0.9	1.0	0.8	0.3
shyy161	1.8	1.8	1.4	1.4	1.4	1.4	1.8	0.7
circuit_3	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
epb2	0.8	0.8	0.7	0.6	0.6	0.6	0.8	0.3
epb3	3.6	3.7	3.2	3.3	3.1	3.1	3.5	1.5
circuit_4	65.4	65.9	60.4	58.6	61.7	62.2	66.2	14.7
e40r0100	0.9	0.9	0.6	0.6	0.6	0.6	0.8	0.1
ns3Da	2.7	2.8	1.7	1.7	1.2	1.2	2.4	0.3
ecl32	8.6	8.4	5.5	5.9	5.2	5.5	8.5	1.4
Zhao1	1.4	1.4	1.4	1.5	1.5	1.4	1.5	0.4
af23560	8.4	8.1	2.9	2.6	2.2	2.2	7.3	0.7
3D_28984	4.7	4.8	2.4	2.3	2.2	2.4	4.8	0.3
3D_51448	7.8	7.3	5.0	4.8	5.0	5.1	7.3	0.6
ibm_matr	7.6	7.4	5.0	5.0	5.0	5.0	8.0	0.6
2D_54019	3.1	3.1	1.3	1.2	1.2	1.2	2.8	0.2
2D_27628	1.1	1.1	0.5	0.6	0.6	0.6	1.0	0.1
sme3Da	0.8	0.8	0.2	0.2	0.2	0.2	0.7	0.1
Avg. gain	-81.7%	-81.8%	-74.4%	-74.2%	-72.6%	-72.9%	-80.8%	

Table B.2.2: Ordering time (in seconds). STR: structural metric used to select the pivots. Otherwise we use a hybrid metric.

	CMLS							
$mindrop =$	0.1		0.9		0.99		0.1	
Matrix/ $NCOL =$	0	10	0	10	0	10	STR	DMLS
Set 1								
av41092	9.9e-01	1.0e+00	1.0e+00	1.0e-00	1.0e+00	1.0e+00	1.0e+00	1.0e+00
g7jac140sc	9.7e-01	1.0e+00	1.9e-03	3.2e-04	6.5e-05	2.1e-04	1.0e+00	6.4e-07
g7jac120sc	9.4e-01	1.0e+00	5.4e-02	1.2e-05	2.1e-05	6.4e-04	1.0e+00	3.5e-07
jan99jac120sc	1.4e-09	1.7e-06	3.7e-08	7.4e-09	1.3e-08	1.0e-07	2.1e-07	4.3e-09
jan99jac100sc	1.6e-07	4.3e-08	5.0e-09	1.2e-08	6.2e-08	7.8e-08	9.0e-08	4.5e-08
bayer10	1.6e-06	3.0e-07	3.6e-07	3.3e-07	1.2e-06	6.0e-09	1.7e-02	9.9e-10
bayer04	9.0e-08	1.0e-07	1.1e-06	8.9e-07	6.9e-08	1.7e-06	1.0e+00	1.5e-05
lhr34c	4.3e-06	4.3e-06	3.7e-05	6.0e-07	7.2e-07	4.2e-07	1.0e+00	8.3e-06
lhr71c	3.6e-04	2.8e-04	6.7e-01	5.4e-03	1.0e-01	3.6e-07	3.6e-01	6.2e-06
mark3jac120sc	1.0e+00	2.0e-01	8.7e-01	8.5e-04	8.2e-02	6.5e-02	8.6e-01	1.3e-03
mark3jac140sc	1.0e+00	9.1e-01	9.1e-01	3.5e-02	8.1e-05	9.4e-01	9.9e-01	2.9e-05
sinc18	1.1e-01	7.3e-01	1.5e-03	7.8e-05	2.8e-06	6.4e-04	7.9e-01	5.4e-01
sinc15	5.9e-01	1.5e-02	2.4e-01	1.4e-02	2.7e-07	4.5e-08	7.7e-01	2.4e-04
Zhao2	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00
fd18	9.5e-01	9.7e-01	5.5e-09	1.5e-08	3.9e-09	2.5e-11	1.1e-09	3.6e-08
fd15	8.5e-01	1.0e-02	1.9e-10	1.1e-11	2.1e-12	7.3e-11	5.6e-08	7.5e-07
mult_dcop_03	2.6e-08	3.2e-08	2.6e-08	2.6e-08	2.6e-08	2.6e-08	3.0e-08	2.0e-14
mult_dcop_02	3.3e-08	7.8e-08	4.3e-10	4.3e-10	9.3e-11	4.3e-10	7.7e-04	4.3e-14
onetone1	8.7e-10	3.0e-10	2.2e-13	7.3e-13	1.6e-13	1.9e-13	9.4e-08	2.8e-13
poli_large	3.5e-16	2.1e-16	2.8e-16	1.8e-16	1.9e-16	3.1e-16	3.8e-16	2.3e-16
bbmat	1.2e-03	1.8e-03	1.2e-07	2.0e-07	4.1e-08	5.2e-08	7.9e-03	1.3e-02
Set 2								
shermanACb	7.0e-13	7.6e-13	9.5e-15	1.9e-14	1.7e-14	1.8e-14	1.1e-12	1.5e-13
rim	9.8e-06	9.0e-07	6.9e-02	8.3e-07	8.8e-05	1.3e-07	9.8e-01	1.8e-05
onetone2	2.0e-13	2.9e-13	1.7e-13	1.6e-13	1.8e-13	1.3e-13	1.1e-07	8.4e-14
shyy161	3.7e-06	6.7e-05	5.2e-16	4.7e-16	5.9e-16	5.9e-16	1.0e+00	4.6e-16
circuit_3	1.2e-09	1.7e-13	7.6e-14	2.3e-14	2.3e-13	7.7e-14	5.4e-11	1.5e-11
epb2	1.4e-08	7.5e-08	2.6e-15	1.4e-15	2.3e-15	2.9e-15	1.5e-07	2.0e-15
epb3	3.0e-08	2.9e-07	5.1e-15	5.7e-15	1.4e-15	1.8e-15	4.1e-03	1.4e-15
circuit_4	7.3e-12	9.0e-12	3.3e-12	1.8e-12	1.8e-12	2.0e-12	3.3e-12	1.4e-11
e40r0100	9.9e-01	9.9e-01	5.0e-01	8.3e-01	9.7e-01	7.3e-01	9.9e-01	4.8e-12
ns3Da	1.0e-10	3.3e-10	5.1e-10	4.1e-10	1.5e-09	6.8e-10	1.4e-10	9.3e-12
ecl32	4.0e-11	1.0e-10	5.2e-13	4.1e-13	7.6e-13	6.2e-13	1.0e+00	3.1e-14
Zhao1	3.9e-15	1.5e-15	1.8e-15	1.8e-15	1.9e-15	1.5e-15	2.1e-15	1.5e-15
af23560	4.1e-08	1.3e-07	1.1e-07	2.0e-10	2.1e-11	2.2e-08	1.1e-02	4.6e-13
3D_28984	9.9e-01	9.6e-01	8.6e-01	5.2e-01	9.9e-01	9.3e-01	1.0e+00	1.0e+00
3D_51448	1.3e-03	1.8e-03	2.6e-07	7.5e-07	1.4e-06	1.1e-08	9.0e-01	1.9e-15
ibm_matr	1.6e-01	8.4e-02	1.6e-08	3.1e-08	2.0e-07	3.0e-08	9.0e-01	1.9e-15
2D_54019	1.0e+00	5.2e-01	1.4e-01	6.5e-02	3.3e-02	6.3e-01	8.7e-01	1.6e-05
2D_27628	1.0e-02	9.7e-01	6.4e-05	7.1e-07	1.1e-06	1.5e-06	9.8e-01	6.6e-16
sme3Da	2.3e-12	6.2e-12	4.4e-16	4.3e-16	3.6e-16	3.7e-16	7.3e-12	4.1e-16

Table B.2.3: SuperLU_DIST: precision without iterative refi nement. STR: structural metric used to select the pivots. Otherwise we use a hybrid metric.

	CMLS							
$mindrop =$	0.1		0.9		0.99		0.1	
Matrix/ $NCOL =$	0	10	0	10	0	10	STR	DMLS
Set 1								
av41092	1.0e+00	1.0e+00	1.0e+00	1.0e-00	1.0e+00	1.0e+00	1.0e+00	1.0e+00
g7jac140sc	7.7e-14	1.7e-14	7.3e-16	5.5e-16	5.4e-16	6.0e-16	1.0e+00	6.4e-16
g7jac120sc	1.5e-14	1.0e+00	5.8e-16	5.4e-16	4.7e-16	6.5e-16	1.0e+00	5.9e-16
jan99jac120sc	5.0e-16	4.8e-16	5.4e-16	5.1e-16	4.9e-16	3.7e-16	5.0e-16	4.6e-16
jan99jac100sc	5.5e-16	4.1e-16	5.1e-16	4.8e-16	3.8e-16	4.6e-16	3.5e-16	3.7e-16
bayer10	2.6e-16	2.5e-16	2.4e-16	2.1e-16	2.4e-16	2.4e-16	2.1e-16	2.7e-16
bayer04	2.3e-16	3.0e-16	2.5e-16	2.5e-16	2.6e-16	2.6e-16	1.0e+00	2.1e-16
lhr34c	1.6e-12	6.6e-14	1.3e-12	1.1e-13	1.2e-13	8.7e-16	9.9e-01	9.5e-14
lhr71c	2.0e-12	2.9e-11	3.0e-08	5.7e-06	3.3e-08	8.5e-16	2.1e-01	1.7e-07
mark3jac120sc	8.2e-01	4.7e-16	9.9e-01	3.5e-16	3.7e-16	4.7e-16	9.9e-16	4.3e-16
mark3jac140sc	1.0e+00	3.0e-13	3.9e-16	3.5e-16	3.3e-16	5.6e-16	6.8e-01	4.2e-16
sinc18	2.9e-01	7.3e-01	1.9e-15	3.2e-16	3.3e-16	1.2e-14	8.7e-01	7.9e-01
sinc15	3.3e-01	7.1e-15	6.7e-01	1.9e-14	3.0e-16	3.1e-16	9.7e-01	9.0e-15
Zhao2	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00	1.0e+00
fd18	8.6e-01	4.0e-11	2.3e-16	2.9e-16	2.3e-16	2.2e-16	2.1e-16	2.1e-16
fd15	8.0e-13	6.6e-15	2.2e-16	2.0e-16	2.2e-16	1.9e-16	2.6e-16	2.6e-16
mult_dcop_03	1.4e-16	1.5e-16	1.5e-16	1.4e-16	1.4e-16	1.4e-16	1.5e-16	1.4e-16
mult_dcop_02	1.6e-16	1.7e-16	1.7e-16	1.7e-16	1.7e-16	1.8e-16	1.6e-16	2.0e-16
onetone1	3.9e-16	3.9e-16	5.8e-16	4.9e-16	4.5e-16	4.8e-16	4.6e-16	5.0e-16
poli_large	1.1e-16	2.1e-16	1.4e-16	1.8e-16	1.9e-16	1.2e-16	1.6e-16	1.0e-16
bbmat	3.9e-16	7.7e-16	5.2e-16	4.6e-16	4.2e-16	3.9e-16	6.9e-15	4.5e-16
Set 2								
shermanACb	2.4e-16	3.1e-16	4.5e-16	8.9e-16	5.5e-16	4.2e-16	4.3e-16	3.4e-16
rim	6.8e-12	1.3e-12	2.4e-02	1.1e-11	1.8e-13	3.9e-13	9.7e-01	9.0e-13
onetone2	3.9e-16	3.2e-16	4.0e-16	3.6e-16	3.6e-16	3.6e-16	4.0e-16	4.6e-16
shyy161	2.0e-16	2.0e-16	3.1e-16	2.5e-16	2.2e-16	2.1e-16	9.9e-01	2.0e-16
circuit_3	6.2e-16	3.5e-16	3.1e-16	4.6e-16	6.1e-16	5.4e-16	5.3e-16	2.5e-16
epb2	3.3e-16	3.0e-16	2.6e-16	2.8e-16	2.7e-16	2.8e-16	2.8e-16	2.6e-16
epb3	3.5e-16	3.2e-16	2.5e-16	2.7e-16	2.5e-16	2.7e-16	3.8e-16	2.8e-16
circuit_4	3.4e-14	2.9e-14	4.3e-14	2.7e-14	3.3e-14	6.9e-14	8.4e-14	1.4e-14
e40r0100	9.9e-01	9.9e-01	1.4e-02	8.7e-01	9.5e-01	6.5e-01	9.9e-01	4.1e-16
ns3Da	4.3e-16	3.7e-16	3.9e-16	2.9e-16	3.5e-16	3.6e-16	2.8e-16	3.4e-16
ecl32	3.5e-16	3.4e-16	3.5e-16	3.2e-16	3.6e-16	3.8e-16	4.7e-06	3.6e-16
Zhao1	2.0e-16	2.0e-16	2.1e-16	2.0e-16	2.0e-16	2.1e-16	2.0e-16	1.3e-16
af23560	2.8e-16	2.9e-16	3.1e-16	3.2e-16	2.6e-16	3.8e-16	3.9e-16	3.1e-16
3D_28984	9.8e-01	6.4e-01	1.2e-12	1.3e-12	2.3e-13	2.0e-11	1.0e+00	1.2e-14
3D_51448	3.0e-16	2.9e-16	3.1e-16	3.0e-16	3.3e-16	3.0e-16	9.9e-01	3.6e-16
ibm_matr	1.7e-01	2.9e-16	3.4e-16	3.3e-16	2.8e-16	3.0e-16	1.0e-00	2.7e-16
2D_54019	2.0e-06	4.2e-01	3.6e-16	3.1e-16	3.6e-16	7.6e-15	7.2e-01	3.6e-16
2D_27628	2.6e-16	2.7e-11	2.6e-16	2.7e-16	3.1e-16	3.3e-16	1.0e-00	2.9e-16
sme3Da	4.5e-16	3.6e-16	3.8e-16	4.5e-16	5.2e-16	4.3e-16	4.2e-16	6.6e-16

Table B.2.4: SuperLU_DIST: precision of the solution after iterative refi nement. STR: structural metric used to select the pivots. Otherwise we use a hybrid metric.

	CMLS							
$mindrop =$	0.1		0.9		0.99		0.1	
Matrix/ $NCOL$ =	0	10	0	10	0	10	STR	DMLS
Set 1								
av41092	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}
g7jac140sc	9	7	5	4	4	4	2^{**}	3
g7jac120sc	8	2^{**}	6	4	3	4	2^{**}	3
jan99jac120sc	3	4	3	3	3	3	3	3
jan99jac100sc	3	3	3	3	3	3	3	3
bayer10	4	4	4	3	4	4	5	3
bayer04	3	3	3	3	3	4	2^{**}	4
lhr34c	5	6	10	6	5	6	2^{**}	9
lhr71c	6	7	6^{*}	5^{*}	9^{*}	5	2^{**}	3^{*}
mark3jac120sc	2^{**}	5	2^{**}	4	5	6	10	5
mark3jac140sc	2^{**}	11	8	6	5	9	3^{**}	4
sinc18	2^{**}	2^{**}	5	5	4	6	2^{**}	2^{**}
sinc15	2^{**}	7	2^{**}	6	4	3	2^{**}	6
Zhao2	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}	2^{**}
fd18	2	6	3	3	3	3	3	4
fd15	6	7	2	3	2	3	3	4
mult_dcop_03	4	3	3	3	2	3	3	3
mult_dcop_02	2	2	2	2	2	2	2	3
onetone1	3	3	3	3	3	3	4	3
poli_large	2	1	2	1	1	2	2	2
bbmat	6	5	3	3	3	3	9	8
Set 2								
shermanACb	4	3	3	3	3	4	4	4
rim	5	4	4^{**}	3	5	5	2^{**}	5
onetone2	3	3	3	3	3	3	4	3
shyy161	3	3	2	2	3	3	2^{**}	2
circuit_3	3	4	3	3	3	3	3	3
epb2	3	4	3	3	3	3	4	3
epb3	3	4	3	3	3	3	6	3
circuit_4	4	3	3	3	3	3	3	4
e40r0100	2^{**}	2^{**}	5^{**}	2^{**}	2^{**}	2^{**}	2^{**}	3
ns3Da	3	3	3	3	3	3	4	3
ecl32	3	3	3	3	3	3	3	3
Zhao1	2	3	2	2	2	2	3	2
af23560	3	3	4	3	3	3	8	3
3D_28984	2^{**}	2^{**}	4	3	3	3	2^{**}	3
3D_51448	4	4	3	3	3	3	2^{**}	3
ibm_matr	2^{**}	4	3	3	3	3	2^{**}	3
2D_54019	5^{*}	2^{**}	5	4	4	4	2^{**}	3
2D_27628	8	4	3	3	3	3	2^{**}	3
sme3Da	3	3	2	2	2	2	3	2

Table B.2.5: SuperLU_DIST: number of steps of iterative refi nement to get the precision of Table B.2.4. ** means that after iterative refi nement the component-wise backward error is greater than 10^{-4} and * means that after iterative refi nement the component-wise backward error is between 10^{-4} and 10^{-8}. STR: structural metric used to select the pivots. Otherwise we use a hybrid metric.

B.3 Preprocessing influence

Matrix	estimated size				real size			
	MC21_MC77		MC64		MC21_MC77		MC64	
	STR	HYB	STR	DMLS	STR	HYB	STR	DMLS
Set 1								
av41092	5497	13199	8033	9448	6083 (1.11)	13991 (1.06)	8224	9654
g7jac140sc	12810	12734	16080	24138	13443 (1.05)	13113 (1.03)	16252	24146
g7jac120sc	10644	10377	14246	19271	11144 (1.05)	10736 (1.03)	14359	19277
jan99jac120sc	2895	2996	3535	3849	3000 (1.04)	3058 (1.02)	3539	3854
jan99jac100sc	2401	2459	2810	3134	2460 (1.02)	2501 (1.02)	2812	3139
bayer10	358	383	365	413	383 (1.07)	396 (1.03)	366	414
bayer04	562	597	514	627	616 (1.10)	632 (1.06)	520	627
lhr34c	8574	7268	8296	6979	13629 (1.59)	8288 (1.14)	8696	7122
lhr71c	16832	13758	16663	15450	24634 (1.46)	15888 (1.15)	17573	15824
mark3jac120sc	13075	29359	14109	13674	13599 (1.04)	30068 (1.02)	14184	13708
mark3jac140sc	15037	34738	16793	16372	15587 (1.04)	35667 (1.03)	16879	16410
sinc18	22968	21680	31581	41445	23785 (1.04)	22084 (1.02)	32334	41563
sinc15	10273	9803	14860	17446	10498 (1.02)	9911 (1.01)	15158	17573
Zhao2	9024	10019	11128	14334	9633 (1.07)	10386 (1.04)	11453	14674
fd18	648	685	569	1082	703 (1.08)	702 (1.02)	570	1119
fd15	394	421	349	655	419 (1.06)	429 (1.02)	349	673
mult_dcop_03	971	1063	991	895	1293 (1.33)	1145 (1.08)	991	844
mult_dcop_02	890	754	761	897	1120 (1.26)	792 (1.05)	761	860
onetone1	2781	3498	3177	3215	3398 (1.22)	3844 (1.10)	3192	3215
poli_large	33	33	33	33	33 (1.00)	33 (1.00)	33	33
bbmat	42954	55530	39034	53596	48187 (1.12)	60927 (1.10)	39600	53641
Avg. gain	36.9%	22.7%	22.3%		27.4%	18.9%	22.0%	
Set 2								
shermanACb	376	382	366	399	382 (1.02)	387 (1.01)	366	399
rim	5909	6216	5907	5957	6894 (1.17)	7297 (1.17)	5944	5959
onetone2	1285	1200	1269	1396	1678 (1.31)	1409 (1.17)	1270	1396
shyy161	3815	3751	3774	4145	3883 (1.02)	3757 (1.00)	3838	4145
circuit_3	60	60	59	60	61 (1.02)	60 (1.00)	59	60
epb2	1848	1919	1902	2129	1926 (1.04)	1940 (1.01)	1942	2129
epb3	4812	5135	5975	7112	4963 (1.03)	5179 (1.01)	6008	7112
circuit_4	440	463	441	464	449 (1.02)	465 (1.00)	442	464
e40r0100	1926	1882	1750	2113	2004 (1.04)	1905 (1.01)	1813	2114
ns3Da	19605	19348	16918	16998	19632 (1.00)	19370 (1.00)	16921	16998
ecl32	38584	37918	35643	33354	39969 (1.04)	38256 (1.01)	35655	33354
Zhao1	8668	8709	6214	5891	8891 (1.03)	8959 (1.03)	6214	5891
af23560	13022	12532	11637	10913	15382 (1.18)	13478 (1.08)	11660	10913
3D_28984	14922	15417	14017	11971	16422 (1.10)	16640 (1.08)	14265	12057
3D_51448	45208	45503	38694	32647	48531 (1.07)	47555 (1.05)	39136	32891
ibm_matr	45017	45438	38492	33202	48183 (1.07)	47980 (1.06)	38909	33491
2D_54019	9689	9736	8432	7614	10724 (1.11)	10399 (1.07)	8686	7836
2D_27628	3682	3767	3529	3043	3892 (1.06)	3919 (1.04)	3598	3057
sme3Da	3884	4048	3862	3848	3884 (1.00)	4049 (1.00)	3862	3848
Avg. gain	−4.5%	−5.5%	0.2%		−10.0%	−8.8%	−0.5%	

Table B.3.1: MA41_UNS: number of entries in the factors with different preprocessings. Number of entries in thousands. STR: structural strategy. HYB: hybrid strategy. In parenthesis, the ratio between the real size and the estimated size of the factors.

Matrix	MC21_MC77		MC64	
	STR	HYB	STR	DMLS
Set 1				
av41092	1537	16321	2539	3664
g7jac140sc	11118	10708	13907	30254
g7jac120sc	8890	8490	12811	22405
jan99jac120sc	859	880	1187	1348
jan99jac100sc	655	710	918	1058
bayer10	16	18	14	18
bayer04	42	46	25	34
lhr34c	7673	3180	2546	1794
lhr71c	12877	6356	5503	5281
mark3jac120sc	8466	58071	8793	7534
mark3jac140sc	9429	68426	10618	9295
sinc18	40850	38391	64041	96970
sinc15	12086	11212	21370	24567
Zhao2	4884	5707	6471	10080
fd18	60	62	43	128
fd15	29	31	22	61
mult_dcop_03	271	209	150	106
mult_dcop_02	211	84	81	111
onetone1	1366	1998	1332	1296
poli_large	0.051	0.051	0.051	0.051
bbmat	56950	104520	39553	77900
Avg. gain	54.1%	31.7%	45.3%	
Set 2				
shermanACb	24	25	21	26
rim	1801	2169	1208	1145
onetone2	418	270	234	243
shyy161	585	534	576	693
circuit_3	0.26	0.258	0.24	0.25
epb2	306	315	320	347
epb3	603	703	938	1149
circuit_4	12	14	10	14
e40r0100	268	239	202	280
ns3Da	17303	16462	12554	12861
ecl32	49973	46706	40558	37816
Zhao1	5052	4989	2517	2267
af23560	9028	7343	5099	4576
3D_28984	9794	10680	7435	5657
3D_51448	54767	55798	38310	28911
ibm_matr	56374	56167	37495	30736
2D_54019	2936	2707	1722	1390
2D_27628	699	742	585	416
sme3Da	894	1004	900	884
Avg. gain	-16.5%	-16.5%	0.8%	

Table B.3.2: MA41_UNS: number of operations (in millions) with different preprocessings. STR: structural strategy. HYB: hybrid strategy.

Matrix	MC21_MC77		MC64	
	STR	HYB	STR	DMLS
Set 1				
av41092	8.2	12.0	10.4	3.9
g7jac140sc	20.2	23.1	27.7	6.2
g7jac120sc	16.5	17.6	23.2	3.8
jan99jac120sc	10.1	12.9	7.4	2.8
jan99jac100sc	7.7	9.5	5.2	2.0
bayer10	0.5	0.6	0.5	0.1
bayer04	1.1	1.6	0.9	0.3
lhr34c	10.8	12.3	8.7	3.4
lhr71c	27.4	23.9	19.0	7.9
mark3jac120sc	17.5	90.2	10.7	3.7
mark3jac140sc	20.3	75.8	13.3	4.4
sinc18	24.8	35.2	27.3	19.7
sinc15	11.0	13.8	13.1	6.5
Zhao2	3.3	4.6	3.4	0.9
fd18	0.7	0.9	0.4	0.1
fd15	0.4	0.5	0.2	0.1
mult_dcop_03	0.7	0.8	0.7	0.4
mult_dcop_02	0.6	1.0	1.2	0.4
onetone1	5.5	13.9	2.3	0.5
poli_large	0.1	0.1	0.1	0.1
bbmat	90.2	80.9	57.5	17.5
Avg. gain	-66.4%	-73.2%	-61.6%	
Set 2				
shermanACb	0.4	0.4	0.3	0.1
rim	2.2	2.5	1.8	0.5
onetone2	0.9	1.0	0.8	0.3
shyy161	1.8	1.8	1.8	0.6
circuit_3	0.1	0.1	0.1	0.1
epb2	1.2	1.5	0.8	0.3
epb3	3.8	7.1	3.5	1.5
circuit_4	25.2	25.8	60.1	15.0
e40r0100	0.8	0.9	0.7	0.1
ns3Da	2.5	2.8	2.4	0.3
ecl32	13.2	13.0	8.4	1.3
Zhao1	2.5	2.7	1.4	0.4
af23560	9.2	7.7	8.3	0.8
3D_28984	4.2	5.2	4.6	0.3
3D_51448	9.4	10.0	7.8	0.6
ibm_matr	9.4	10.4	8.3	0.6
2D_54019	3.2	3.7	2.8	0.2
2D_27628	1.4	1.6	1.0	0.1
sme3Da	0.6	1.0	0.7	0.1
Avg. gain	-76.5%	-78.7%	-75.2%	

Table B.3.3: Influence of preprocessing on CMLS execution time (in seconds). STR: structural strategy. HYB: hybrid strategy.

Matrix	Before iterative refinement		After iterative refinement	
	CMLS	DMLS	CMLS	DMLS
Set 1				
av41092	1.0e+00	1.0e+00	1.0e+00	1.0e+00
g7jac140sc	1.0e+00	6.4e-07	1.0e+00	6.4e-16
g7jac120sc	1.0e+00	3.5e-07	1.0e+00	5.9e-16
jan99jac120sc	2.3e-03	4.3e-09	3.6e-16	4.6e-16
jan99jac100sc	7.4e-01	4.5e-08	9.7e-14	3.7e-16
bayer10	1.0e+00	9.9e-10	1.0e+00	2.7e-16
bayer04	1.0e+00	1.5e-05	1.0e+00	2.1e-16
lhr34c	1.0e+00	8.3e-06	1.0e+00	9.5e-14
lhr71c	1.0e+00	6.2e-06	1.0e+00	1.7e-07
mark3jac120sc	1.0e+00	1.3e-03	1.0e+00	4.3e-16
mark3jac140sc	1.0e+00	2.9e-05	1.0e+00	4.2e-16
sinc18	8.5e-01	5.4e-01	6.1e-12	7.9e-01
sinc15	1.8e-04	2.4e-04	3.8e-16	9.0e-15
Zhao2	1.0e+00	1.0e+00	1.0e+00	1.0e+00
fd18	9.9e-01	3.6e-08	9.4e-01	2.1e-16
fd15	9.8e-01	7.5e-07	1.0e-12	2.6e-16
mult_dcop_03	5.8e-06	2.0e-14	1.8e-16	1.4e-16
mult_dcop_02	4.8e-10	4.3e-14	1.8e-16	2.0e-16
onetone1	1.0e+00	2.8e-13	1.0e+00	5.0e-16
poli_large	2.3e-15	2.3e-16	2.1e-16	1.0e-16
bbmat	1.0e+00	1.3e-02	1.0e+00	4.5e-16
Set 2				
shermanACb	4.6e-01	1.5e-13	2.8e-01	3.4e-16
rim	9.8e-01	1.8e-05	9.9e-01	9.0e-13
onetone2	1.0e+00	8.4e-14	1.0e+00	4.6e-16
shyy161	1.8e-03	4.6e-16	2.2e-16	2.0e-16
circuit_3	9.5e-03	1.5e-11	5.0e-16	2.5e-16
epb2	2.2e-03	2.0e-15	3.1e-16	2.6e-16
epb3	1.0e+00	1.4e-15	3.7e-16	2.8e-16
circuit_4	1.2e-03	1.4e-11	3.6e-13	1.4e-14
e40r0100	9.9e-01	4.8e-12	9.9e-01	4.1e-16
ns3Da	1.3e-09	9.3e-12	3.1e-16	3.4e-16
ecl32	1.0e+00	3.1e-14	1.0e+00	3.6e-16
Zhao1	1.0e+00	1.5e-15	1.0e+00	1.3e-16
af23560	1.0e+00	4.6e-13	1.0e+00	3.1e-16
3D_28984	1.0e+00	1.0e+00	1.0e+00	1.2e-14
3D_51448	1.0e+00	1.9e-15	1.0e+00	3.6e-16
ibm_matr	1.0e+00	1.9e-15	1.0e+00	2.7e-16
2D_54019	1.0e+00	1.6e-05	1.0e+00	3.6e-16
2D_27628	1.0e+00	6.6e-16	1.0e+00	2.9e-16
sme3Da	5.9e-11	4.1e-16	5.3e-16	6.6e-16

Table B.3.4: `SuperLU_DIST` component-wise backward error. `CMLS` is run with `MC77_MC21` preprocessing and a hybrid strategy with two thresholds is used.

Bibliography

[1] The OpenMP application program interface. http://www.openmp.org.

[2] PARASOL test data. URL: http://www.parallab.uib.no/parasol/data.html, 2002.

[3] P. R. Amestoy. Recent progress in parallel multifrontal solvers for unsymmetric sparse matrices. In *Proceedings of the 15th World Congress on Scientific Computation, Modelling and Applied Mathematics, IMACS 97, Berlin*, 1997.

[4] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17:886–905, 1996.

[5] P. R. Amestoy, T. A. Davis, and I. S. Duff. Algorithm 8xx: Amd, an approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software*, 2004.

[6] P. R. Amestoy and I. S. Duff. Vectorization of a multiprocessor multifrontal code. *Int. J. of Supercomputer Applics.*, 3:41–59, 1989.

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

[8] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods Appl. Mech. Eng.*, 184:501–520, 2000.

[9] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li. Analysis and comparison of two general sparse solvers for distributed memory computers. *ACM Transactions on Mathematical Software*, 27(4):388–421, 2001.

[10] P. R. Amestoy, I. S. Duff, S. Pralet, and C. Vömel. Adapting a parallel sparse direct solver to architectures with clusters of smps. *Parallel Computing*, 29(11-12):1645–1668, 2003.

[11] P. R. Amestoy, I. S. Duff, and C. Vömel. Task scheduling in an asynchronous distributed memory multifrontal solver. *SIAM Journal on Matrix Analysis and Applications*, 2004.

[12] P. R. Amestoy, X. S. Li, and E.G. Ng. Diagonal Markowitz scheme with local symmetrization. Technical Report RT/APO/03/5, ENSEEIHT-IRIT, October 2003. Also appeared as Lawrence Berkeley Lab report LBNL-53854.

[13] P. R. Amestoy and C. Puglisi. An unsymmetrized multifrontal LU factorization. *SIAM Journal on Matrix Analysis and Applications*, 24:553–569, 2002.

[14] M. Arioli, J. Demmel, and I. S. Duff. Solving sparse linear systems with sparse backward error. *SIAM Journal on Matrix Analysis and Applications*, 10:165–190, 1989.

[15] C. Ashcraft. Compressed graphs and the minimum degree algorithm. *SIAM J. Sci. Comput.*, 16(6):1404–1411, 1995.

[16] C. Ashcraft and R. G. Grimes. SPOOLES: An object oriented sparse matrix library. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio, Texas, March 22–24, 1999.

[17] C. Ashcraft, R. G. Grimes, and J. G. Lewis. Accurate symmetric indefinite linear equation solver. *SIAM Journal on Matrix Analysis and Applications*, 20:513–561, 1998.

[18] C. Ashcraft and J. W. H. Liu. Robust ordering of sparse matrices using multisection. *SIAM Journal on Matrix Analysis and Applications*, 19:816–832, 1998.

[19] J. R. Bunch. Analysis of the diagonal pivoting method. *SIAM J. Numer. Anal.*, 8:656–680, 1971.

[20] J. R. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation*, 31:162–179, 1977.

[21] J. R. Bunch and B. N. Parlett. Direct methods for solving symmetric indefinite systems of linear systems. *SIAM J. Numer. Anal.*, 8:639–655, 1971.

[22] S. H. Cheng and N. J. Higham. A modified Cholesky algorithm based on a symmetric indefinite factorization. *SIAM Journal on Matrix Analysis and Applications*, 19(4):1097–1112, 1998.

[23] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. ScaLAPACK: A portable linear algebra library for distributed memory computers - design issues and performance. *Computer Physics Communications*, 97:1–15, 1996. (also as LAPACK Working Note #95).

[24] A. R. Curtis and J. K. Reid. On the automatic scaling of matrices for Gaussian elimination. *J. Inst. Maths. Applics.*, 10:118–124, 1972.

[25] T. A. Davis. University of Florida sparse matrix collection, `http://www.cise.ufl.edu/research/sparse/matrices/`, 2002.

[26] T. A. Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. Technical Report TR-00-006, Computer and Information Sciences Department, University of Florida, Gainesville, FL, 2003. To appear in TOMS.

[27] T. A. Davis and I. S. Duff. An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM Journal on Matrix Analysis and Applications*, 18:140–158, 1997.

[28] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 20(3):720–755, 1999.

[29] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[30] J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling. Algorithm 679. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16:1–17, 1990.

[31] J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling. Algorithm 679. A set of Level 3 Basic Linear Algebra Subprograms: model implementation and test programs. *ACM Transactions on Mathematical Software*, 16:18–28, 1990.

[32] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. SIAM Press, Philadelphia, 1998.

[33] I . S. Duff and S. Pralet. Experiments in preprocessing and scaling symmetric problems for multifrontal solutions. Technical Report WN/PA/04/17, CERFACS, Toulouse, France, 2004.

[34] I. S. Duff and J. R. Gilbert . Maximum-weight matching and block pivoting for symmmetric indefinite systems. In *Abstract book of Householder Symposium XV, June 17-21*, 2002.

[35] I. S. Duff. Algorithm 575. Permutations for a zero-free diagonal. *ACM Transactions on Mathematical Software*, 7:387–390, 1981.

[36] I. S. Duff. On algorithms for obtaining a maximum transversal. *ACM Transactions on Mathematical Software*, 7:315–330, 1981.

[37] I. S. Duff. MA57 - a new code for the solution of sparse symmetric definite and indefinite systems. Technical report RAL-TR-2002-024, Rutherford Appleton Laboratory, 2002.

[38] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, London, 1986.

[39] I. S. Duff, R. G. Grimes, and J. G. Lewis. The Rutherford-Boeing Sparse Matrix Collection. Technical Report RAL-TR-97-031, Rutherford Appleton Laboratory, 1997. Also Technical Report ISSTECH-97-017 from Boeing Information & Support Services and Report TR/PA/97/36 from CERFACS, Toulouse.

[40] I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 20(4):889–901, 1999.

[41] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM Journal on Matrix Analysis and Applications*, 22(4):973–996, 2001.

[42] I. S. Duff and J. K. Reid. A comparison of sparsity orderings for obtaining a pivotal sequence in Gaussian elimination. *Journal of the Institute of Mathematics and its Applications*, 14:281–291, 1974.

[43] I. S. Duff and J. K. Reid. MA27—a set of Fortran subroutines for solving sparse symmetric sets of linear equations. Technical Report R.10533, AERE, Harwell, England, 1982.

[44] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Transactions on Mathematical Software*, 9:302–325, 1983.

[45] I. S. Duff and J. K. Reid. The multifrontal solution of unsymmetric sets of linear systems. *SIAM Journal on Scientific and Statistical Computing*, 5:633–641, 1984.

[46] I. S. Duff and J. K. Reid. Exploiting zeros on the diagonal in the direct solution of indefinite sparse symmetric linear systems. *ACM Transactions on Mathematical Software*, 22(2):227–257, 1996.

[47] I. S. Duff, N. I. M. Gould, J. K. Reid, J. A. Scott, and K. Turner. Factorization of sparse symmetric indefinite matrices. *IMA Journal of Numerical Analysis*, 11:181–204, 1991.

[48] I. S. Duff and J. K. Reid. MA47, a Fortran code for direct solution of indefinite sparse symmetric linear systems. Technical Report RAL 95-001, Rutherford Appleton Laboratory, 1995.

[49] J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards (B)*, 69:125–130, 1965.

[50] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[51] E. Eskow and R. B. Schnabel. Algorithm 695: Software for a new modified Cholesky factorization. *ACM Transactions on Mathematical Software*, 17:306–312, 1991.

[52] H. N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 434–443. Society for Industrial and Applied Mathematics, 1990.

[53] A. George and J. W. H. Liu. An automatic nested dissection algorithm for irregular finite element problems. *SIAM Journal on Numerical Analysis*, 15:1053–1069, 1978.

[54] A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ., 1981.

[55] A. George and D. R. McIntyre. On the application of the minimum degree algorithm to finite element systems. *SIAM Journal on Numerical Analysis*, 15:90–111, 1978.

[56] A. George and J. W. H. Liu. A fast implementation of the minimum degree algorithm using quotient graphs. *ACM Trans. Math. Softw.*, 6(3):337–358, 1980.

[57] J. R. Gilbert and J. W. H. Liu. Elimination structures for unsymmetric sparse LU factors. *SIAM Journal on Matrix Analysis and Applications*, 14:334–352, 1993.

[58] J. R. Gilbert and E. G. Ng. Predicting structure in nonsymmetric sparse matrix factorizations. In J.R. Gilbert A. George and J.W.H Liu, editors, *Graph Theory and Sparse Matrix Computations*, pages 107–140. Springer-Verlag NY, 1993.

[59] P. E. Gill and W. Murray. Newton-type methods for unconstrained and linearly constrained optimization. *Mathematical Programming*, 28:311–350, 1974.

[60] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr (and SifDec) a constrained and unconstrained testing environment testing environment revisited. Technical Report 2002-009, Rutherford Appleton Laboratory, 2002.

[61] N. I. M. Gould and J. A. Scott. A numerical evaluation of HSL packages for the direct solution of large sparse, symmetric linear systems of equations. Technical Report RAL-TR-2003-019, Rutherford Appleton Laboratory, 2003. To appear in *ACM Trans. Math. Softw.*

[62] N. I. M. Gould and J. A. Scott. A numerical evaluation of HSL packages for the direct solution of large sparse, symmetric linear systems of equations. *ACM Transactions on Mathematical Software*, 30(3), September 2004.

[63] A. Guermouche and J.-Y. L'Excellent. Coherent load information mechanisms for distributed dynamic scheduling. Technical Report RR2004-25, LIP, 2004. Also INRIA report RR5178.

[64] A. Guermouche and J.-Y. L'Excellent. Memory-based scheduling for a parallel multifrontal solver. In *18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, 2004.

[65] A. Gupta, F. Gustavson, M. Joshi, G. Karypis, and V. Kumar. Pspases: An efficient and scalable parallel sparse direct solver. Technical report, Department of Computer Science, University of Minnesota and IBM T.J. Watson Research center, 1999.

[66] A. Gupta, G. Karypis, and V. Kumar. Highly scalable parallel algorithms for sparse matrix factorization. *IEEE Trans. Parallel and Distributed Systems*, 8(5):502–520, 1997.

[67] A. Gupta. Recent advances in direct methods for solving unsymmetric sparse systems of linear equations. *ACM Transactions on Mathematical Software*, 28(3):301–324, 2002.

[68] P. Heggernes, S. Eisenstat, G. Kumfert, and A. Pothen. The computational complexity of the minimum degree algorithm. In *Proceedings of the Norwegian Conference on Computer Science NIK*, 2002.

[69] B. Hendrickson and R. Leland. The CHACO User's Guide. Version 2.0. Technical Report SAND94-2692, Sandia National Laboratories, Albuquerque, 1994.

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

[71] N. J. Higham. *Accuracy and Stabilty of Numerical Algorithms*. SIAM, Philadelphia, 2. edition, 2002.

[72] HSL. A collection of Fortran codes for large scale scientific computation, 2004.

[73] G. Karypis and V. Kumar. METIS – *A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices – Version 4.0*. University of Minnesota, September 1998.

[74] H.W Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[75] G. Kumfert and A. Pothen. Two improved algorithms for envelope and wavefront reduction. *BIT*, 37(3):559–590, September 1997.

[76] E. Lawler. *Combinatorial Optimization: Networks and Matroids*. Dover, 2001.

[77] X. S. Li and J. W. Demmel. Making sparse Gaussian elimination scalable by static pivoting. In *Proceedings of Supercomputing*, Orlando, Florida, November 1998.

[78] X. S. Li and J. W. Demmel. A scalable sparse direct solver using static pivoting. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio, Texas, March 22–24 1999.

[79] X. S. Li and J. W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Transactions on Mathematical Software*, 29(2), 2003.

[80] J. W. H. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software*, 11(2):141–153, 1985.

[81] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM Journal on Matrix Analysis and Applications*, 11:134–172, 1990.

[82] J. W. H. Liu. The multifrontal method for sparse matrix solution: Theory and Practice. *SIAM Review*, 34:82–109, 1992.

[83] J. W. H. Liu, E. G. Ng, and W. Peyton. On finding supernodes for sparse matrix computations. *SIAM Journal on Matrix Analysis and Applications*, 14:242–252, 1993.

[84] I. Maros and C. Meszaros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11–12:671–681, 1999.

[85] E. Ng and P. Raghavan. Performance of greedy heuristics for sparse Cholesky factorization. *SIAM Journal on Matrix Analysis and Applications*, 20:902–914, 1999.

[86] G. Pagallo and C. Maulino. A bipartite quotient graph model for unsymmetric matrices. In *Lecture Notes in Mathematics 1005, Numerical Method*, pages 227–239, Springer-Verlag, New York, 1983.

[87] B. N. Parlett and T. L. Landis. Methods for scaling to doubly stochastic form. *Linear Algebra and its Applications*, 48:53–79, 1982.

[88] F. Pellegrini. SCOTCH 3.4 User's guide. Technical Report RR 1264-01, LaBRI, Université Bordeaux I, November 2001.

[89] F. Pellegrini and J. Roman. Sparse matrix ordering with SCOTCH. In *Proceedings of HPCN'97, Vienna, LNCS 1225*, pages 370–378, April 1997.

[90] F. Pellegrini, J. Roman, and P. Amestoy. Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering. *Concurrency: Practice and Experience*, 12(2-3):69–84, 2000.

[91] A. Pothen and C. Sun. A Mapping Algorithm for Parallel Sparse Cholesky Factorization. *SIAM Journal on Scientific Computing*, 14(5):1253–1257, 1993.

[92] E. Rothberg and S. C. Eisenstat. Node selection strategies for bottom-up sparse matrix ordering. *SIAM Journal on Matrix Analysis and Applications*, 19(3):682–695, 1998.

[93] D. Ruiz. A scaling algorithm to equilibrate both rows and columns norms in matrices. Technical Report RT/APO/01/4, ENSEEIHT-IRIT, 2001. Also appeared as RAL report RAL-TR-2001-034. Submitted to Linear Algebra and its Applications.

[94] R. Schreiber. Scalability of sparse direct solvers. *Graph Theory and Sparse Matrix Computation*, pages 191–209, 1994.

[95] J. Schulze. Towards a tighter coupling of bottom-up and top-down sparse matrix ordering methods. *BIT*, 41(4):800–841, 2001.

[96] Z. Zlatev. On some pivotal strategies in gaussian elimination by sparse technique. *SIAM Journal on Numerical Analysis*, 17:18–30, 1980.

Résumé

Nous nous intéressons à la résolution de systèmes linéaires creux de grande taille par des solveurs creux directs opérant en trois phases qui sont l'analyse, la factorisation et la résolution.

L'analyse est le lieu de prétraitements et doit dans la mesure du possible assurer à la fois des facteurs aussi creux que possible et une factorisation numériquement stable. La factorisation doit exploiter l'indépendance des calculs pour être efficace dans un environnement parallèle distribué. Cette étude contribue à l'amélioration de ces comportements sur des classes de problèmes connues comme étant difficiles ou mal appréhendées par des stratégies classiques.

Dans une première partie, nous développons des techniques de prétraitements numériques et structurels pour les matrices symétriques indéfinies. Nous étudions aussi de manière plus prospective des approches de factorisation LDL^T avec pivotage statique et l'élaboration d'ordonnancements pour les systèmes augmentés.

Dans une deuxième partie, nous présentons des techniques d'ordonnancements pour les matrices très non symétriques visant à la fois à réduire le remplissage et à stabiliser la factorisation. Ces ordonnancements reposent sur des métriques hybrides prenant en compte des informations structurelles et numériques.

Dans une troisième partie, nous discutons des stratégies de séquencement des tâches dans un solveur multifrontal parallèle, MUMPS. Dans un premier temps, nous essayons de prendre en compte l'hétérogénéité des architectures des machines cibles. Dans un second temps, nous prenons en compte à la fois des critères de charge de travail et de mémoire pour une prise de décision dynamique optimale.

Mots-clés: calcul distribué, calcul parallèle, élimination de Gauss, matrices creuses, maximum matching, méthode multifrontale, ordonnancement, séquencement de tâches.

Abstract

We consider the three different phases (analysis, factorization, solution) for the direct solution of large sparse systems of linear equations.

During the analysis phase, preprocessing is applied in order to, on the one hand, permute the matrix to decrease the number of nonzeros in the factors and, on the other hand, to determine pivots that ensure as much as possible a stable factorization. During the factorization phase, the independence of the computations must be exploited to achieve a good performance on a distributed memory parallel computer. Our study contributes to the improvement of these aspects for some classes of matrices which are known for being difficult or for which default strategies clearly do not perform well.

In the first part of our thesis, we develop numerical and structural preprocessing strategies for symmetric indefinite matrices. We also study, in a more prospective way, static pivoting approaches for LDL^T factorizations and orderings for augmented systems.

In the second part, we present orderings for highly unsymmetric matrices that aim at decreasing the fill-in and at stabilizing the factorization. These orderings are based on hybrid metrics which take into account both structural and numerical information.

In the last part, we discuss task scheduling strategies in a parallel multifrontal solver, MUMPS. We study two kinds of strategies. Firstly, we investigate strategies that take into account heterogeneous architectures. Secondly, we study strategies that mix information about workload and memory to make optimal dynamic decisions.

Keywords: distributed computing, Gaussian elimination, maximum matching, multifrontal method, ordering, parallel computing, sparse matrices, tasks scheduling.

Thèse préparée au CERFACS, CERFACS Report Ref: TH/PA/04/105
42, Avenue Gaspard Coriolis. 31057 Toulouse Cedex 01. France.