Science & Technology Facilities Council

Preprint

RAL-P-2013-001

The augmented block Cimmino distributed method

IS Duff, R Guivarch, D Ruiz, M Zenadi

February 2013

Submitted for publication in SIAM Journal on Matrix Analysis and Applications

RAL Library
STFC Rutherford Appleton Laboratory
R61
Harwell Oxford
Didcot
OX11 0QX

Tel: +44(0)1235 445384
Fax: +44(0)1235 446403
email: libraryral@stfc.ac.uk

ISSN 1361- 4762

The Augmented Block Cimmino Distributed method

Iain S. Duff[1,2], Ronan Guivarch[3], Daniel Ruiz[3] and Mohamed Zenadi[3]

ABSTRACT

We introduce and study a novel way of accelerating the convergence of the block Cimmino method by augmenting the matrix so that the subspaces corresponding to the partitions are orthogonal. This results in a requirement to solve a relatively smaller symmetric positive definite system. We consider ways to reduce the size of this system using a controlled departure from orthogonality. We illustrate the performance of our method on some realistic test problems.

Keywords: sparse matrices, hybrid solver, block Cimmino, multifrontal method, orthogonalization of subspaces

AMS(MOS) subject classifications: 05C50, 05C70, 65F50

This report is available through the URL `http://www.stfc.ac.uk/CSE/36276.aspx`. It has also appeared as CERFACS report TR/PA/13/11 and INPT(ENSEEIHT)-IRIT report RT-APO-13-2.

[1]CERFACS, 42 Avenue Gaspard Coriolis, 31057, Toulouse, France (`duff@cerfacs.fr`).

[2]R 18, RAL, Oxon, OX11 0QX, England (`iain.duff@stfc.ac.uk`). The research of this author was supported in part by the EPSRC Grant EP/I013067/1.

[3]Université de Toulouse, INPT(ENSEEIHT)-IRIT, France ({`guivarch,ruiz,zenadi`}`@enseeiht.fr`).

Scientific Computing Department
R 18
Rutherford Appleton Laboratory
Oxon OX11 0QX

February 18, 2013

Contents

1 Introduction

We study the solution of the system

$$Ax = b \tag{1.1}$$

where A is an $m \times n$ sparse matrix, x is an n-vector and b is an m-vector. In the following, we assume the system is consistent and for simplicity we suppose that A has full rank.

We will study the solution of the system (1.1) using the block Cimmino method, an iterative method using block-row projections. In this method, the system (1.1) is subdivided into strips of rows as in the following:

$$\begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_p \end{pmatrix} x = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{pmatrix}. \tag{1.2}$$

Let $P_{\mathcal{R}(A_i^T)}$ be the projector onto the range of A_i^T and ${A_i}^+$ be the Moore-Penrose pseudo-inverse of the partition A_i. The block Cimmino algorithm then computes a solution iteratively from an initial estimate $x^{(0)}$ according to:

$$u_i = A_i^+ \left(b_i - A_i x^{(k)} \right) \quad i = 1, ...p \tag{1.3}$$

$$x^{(k+1)} = x^{(k)} + \omega \sum_{i=1}^{p} u_i \tag{1.4}$$

where we note the independence of the set of p equations, which is why the method is so attractive in a parallel environment. The block Cimmino method is described in more detail by Ruiz (1992).

Although the matrix in equation (1.1) can be rectangular and the Cimmino method can work on such systems (Elfving 1980), for our main discussion we will assume that A is square and of order n.

With the above notations, the iteration equations are thus:

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \omega \sum_{i=1}^{p} A_i^+ \left(b_i - A_i x^{(k)} \right) \\ &= \left(I - \omega \sum_{i=1}^{p} A_i^+ A_i \right) x^{(k)} + \omega \sum_{i=1}^{p} A_i^+ b_i \\ &= Q x^{(k)} + \omega \sum_{i=1}^{p} A_i^+ b_i. \end{aligned}$$

The iteration matrix for block Cimmino $H = I - Q$ is then a sum of projectors $H = \omega \sum_{i=1}^{p} \mathcal{P}_{\mathcal{R}(A_i^T)}$. It is thus symmetric and positive definite and so we can solve

$$Hx = \xi, \tag{1.5}$$

where $\xi = \omega \sum_{i=1}^{p} A_i^+ b_i$ using conjugate gradient or block conjugate gradient methods. As ω appears on both sides of equation (1.5), we can set it to one.

There are many aspects to implementing the block Cimmino method. These include the original partitioning of the equations as in (1.2), the use of block conjugate gradients, and the way to solve the underdetermined systems on each partition.

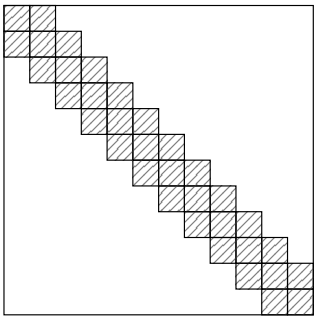


Figure 1.1: The matrix permuted to block tridiagonal form.

A common partitioning strategy is to use an ordering like Reverse Cuthill-McKee (Chan and George 1980) to permute the matrix to block triangular form as in Figure 1.1 and then, if each partition incorporates at least two block rows, both the even and the odd partitions are mutually orthogonal (no column overlap) and the resulting matrix partition is called a two-block partitioning. This was studied in detail by Elfving (1998). The matrix H in this case is the sum of two orthogonal projectors. Therefore, the maximum eigenvalue of the iteration matrix H is 2 and the eigenvalues are clustered around 1. Indeed, it can be shown that the eigenvalues in this two-block case are equal to 1 plus or minus the cosine of the principal angles between the two subspaces. Arioli, Duff, Noailles and Ruiz (1992) used this property to obtain a better eigenvalue distribution by opening the angles between these subspaces using simple column scalings or ellipsoidal norms.

In the solution of the underdetermined systems (1.3), A_i^+ is the Moore Penrose pseudo-inverse of A_i and we can solve these systems knowing that:

$$A_i^+ \;=\; A_i^T (A_i A_i^T)^{-1},$$

but it is better from a sparsity and a numerical point of view to solve instead

$$A_i u_i \;=\; r_i, \qquad (r_i = b_i - A_i x^{(k)})$$

using the augmented system approach :

$$\begin{pmatrix} I & A_i^T \\ A_i & 0 \end{pmatrix} \begin{pmatrix} u_i \\ v_i \end{pmatrix} \;=\; \begin{pmatrix} 0 \\ r_i \end{pmatrix}.$$

In our implementation, we use the sparse direct multifrontal solver MUMPS (Amestoy, Duff, L'Excellent and Koster 2001) to solve this augmented system. An added benefit to the robustness of a direct solver is that MUMPS can exploit the sparsity structure within the blocks, giving us another level of parallelism to that already obtained from the block Cimmino iteration.

There are two ways that ill-conditioning can affect the solution using block Cimmino.

- Within the blocks where the systems being solved are symmetric indefinite problems as ill-conditioning can cause any sparse direct method to behave poorly or unpredictably.
- The other way is across the blocks where there can be problems if the subspaces are far from orthogonal.

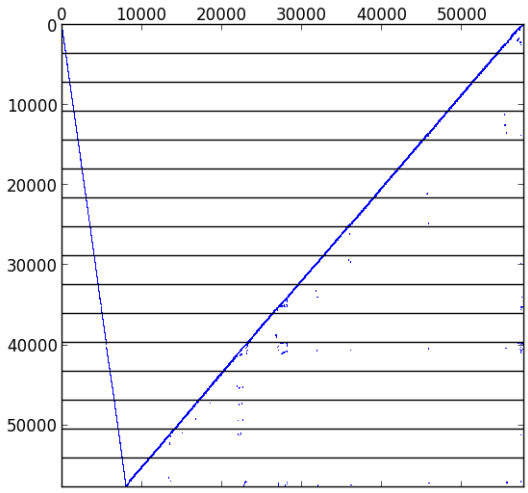


Figure 1.2: Nonzero pattern of the matrix `bayer01`.

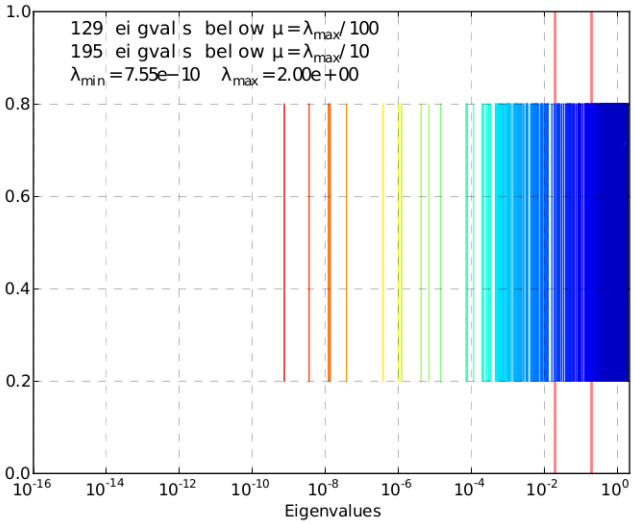


Figure 1.3: Spectrum of block Cimmino iteration matrix for `bayer01` with 16 uniform partitions.

In our discussion on the theoretical properties of our algorithm, we will use a realistic test matrix to illustrate this behaviour. This is the matrix `bayer01` obtained from Bayer AG by Friedrich Grund and available from the sparse matrix collection at the University of Florida (Davis 2008). It is of order 57735 and has 277774 nonzero entries. We partition it into 16 uniform partitions, and we show its pattern in Figure 1.2. In Figure 1.3, we show the spectrum of the iteration matrix, H, for the `bayer01` matrix when using block Cimmino with these 16 uniform partitions.

Although we see that there is a good clustering of eigenvalues around the value 1 (and this property is true in general) the matrix may still be badly conditioned. One way to increase the clustering while reducing the ill-conditioning is to open the angles between subspaces corresponding to the different partitions. One method for doing this is to reorder the matrix and partition it following the level sets obtained from the use of the Cuthill-McKee algorithm on AA^T

(Drummond, Duff and Ruiz 1993). Although this helps to open the angles, it is costly, difficult to implement in a distributed memory environment, and does not always give better results.

As an alternative, we use a hypergraph partitioner `PaToH` (Catalyürek and Aykanat 1999) to find a row permutation that will make the partitions less interconnected. `PaToH` provides permutations so that the reordered matrix is in bordered block diagonal form where the blocks on the diagonal will usually be underdetermined. We notice that, with the matrix in this form, the overlap is only within the boundary columns that `PaToH` is trying to minimize. One can input to `PaToH` the desired number of partitions and, in this present comparison, we use the value 16, the same number as for our uniform partition. We notice in Figure 1.4 that the intermediate eigenvalues have been shifted towards 1 which improves the clustering of eigenvalues in the iteration matrix.

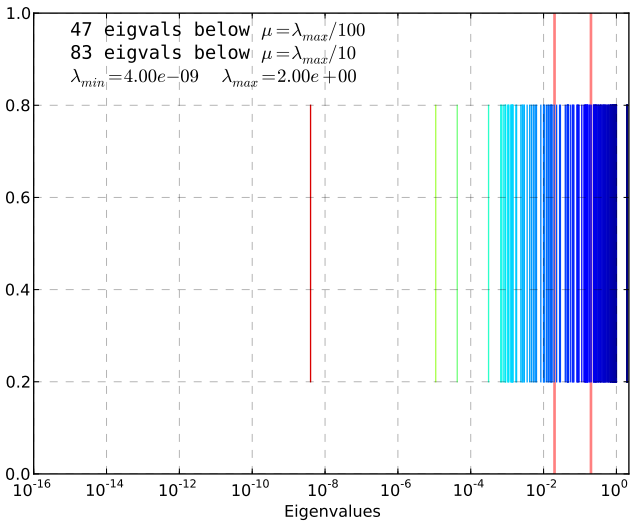


Figure 1.4: Spectrum of block Cimmino iteration matrix for `bayer01` with 16 partitions obtained using the hypergraph partitioner `PaToH`.

We notice in Figure 1.5 that a uniform partitioning of the matrix `bayer01` presents long plateaux during the convergence of the conjugate gradient iterations. These plateaux are due to the presence of clusters of small eigenvalues. However, as we might expect, hypergraph partitioning reduces the number of small eigenvalues as shown in Figure 1.4 and consequently improves greatly the convergence as we see from the `PaToH` curve in Figure 1.5.

Although we see that improving the iteration matrix makes the conjugate gradient method converge faster, the plateaux are still present. These can be reduced even more by using block CG as in Arioli, Duff, Ruiz and Sadkane (1995*b*) but they will still be present. Further techniques, such as the combination of Chebyshev preconditioning with CG can be beneficial to extract some near-invariant subspace (corresponding to all eigenvalues below some threshold, typically $\lambda_{max}/100$) from the Krylov subspace and to reuse this spectral information to speed up further solutions with the same matrix but new right-hand sides (Golub, Ruiz and Touhami 2007). Such techniques are, however, limited because there may be many eigenvalues in such fixed intervals, and so the memory requirements can become prohibitive.

We will discuss in the following a novel way of augmenting the system to force orthogonality

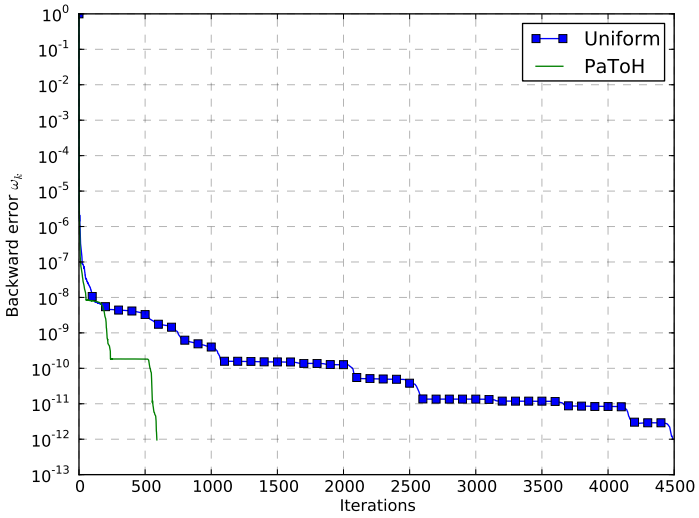


Figure 1.5: The convergence of block Cimmino iteration for `bayer01` with 16 uniform partitions in comparison with 16 partitions obtained using `PaToH`.

between the subspaces. We describe our augmentation in Section 2 and consider ways of reducing the extra work caused by the augmentation in Section 3.

We present our conclusions in Section 4.

2 The augmented block Cimmino method

For simplicity, assume that we have a matrix with a block tridiagonal structure as shown in Figure 2.1(a). Note that the block Cimmino method can work with any matrix structure.

In this figure we have defined four partitions A_1 to A_4. As we can see in this practical example, each partition is interconnected only with its neighbours. Thus, the product of these partitions can be represented by the following :

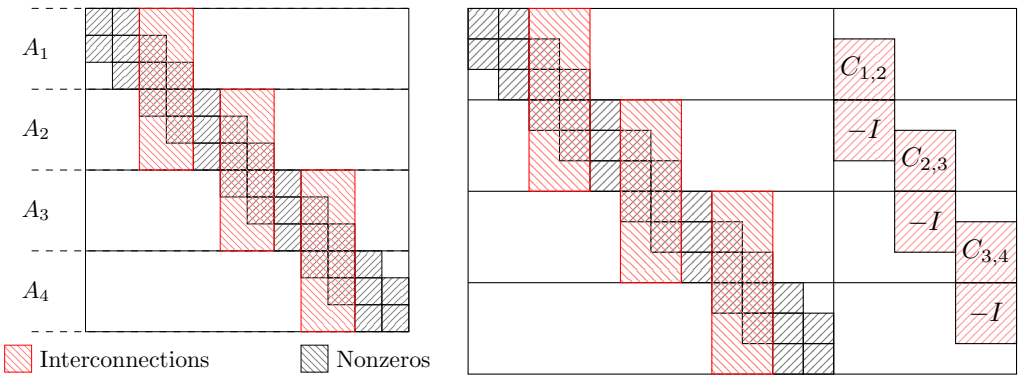
$$A_i A_j^T = \begin{cases} 0 & \text{if } j \neq i \pm 1 \\ A_{ij} A_{ji}^T & \text{if } j = i \pm 1 \end{cases}$$

where A_{ji} and A_{ij} are the submatrices of A_i and A_j respectively that overlap columnwise with each other, see Figure 2.1(a).

We then augment the matrix A to generate a matrix $\bar{A}$ with partitions $\bar{A}_i$ so that the inner products $\bar{A}_i \bar{A}_j^T$ are zero. We consider three different ways to augment the matrix to obtain these zero matrix products.

- One can repeat the submatrices A_{ij} and A_{ji}, reversing the signs of one of them as in the following

$$\left[\begin{array}{ccccc|cc} A_{1,1} & A_{1,2} & & & & A_{1,2} & \\ & A_{2,1} & A_{2,2} & A_{2,3} & & -A_{2,1} & A_{2,3} \\ & & & A_{3,2} & A_{3,3} & & -A_{3,2} \end{array}\right] \tag{2.1}$$



(a) Block tridiagonal structure (b) Normal equations based augmentation

Figure 2.1: Illustrative example of an augmentation process.

- We can also use the normal equations so that, for each pair of neighbouring partitions A_i and A_j, we expand them on the right with $C_{ij} = A_{ij}A_{ji}^T$ and $-I$ respectively as in the following

$$\left[\begin{array}{ccccc|cc} A_{1,1} & A_{1,2} & & & & C_{1,2} & \\ & A_{2,1} & A_{2,2} & A_{2,3} & & -I & C_{2,3} \\ & & & A_{3,2} & A_{3,3} & & -I \end{array}\right] \tag{2.2}$$

  It would also be possible to expand with $-I$ and C_{ij}^T which will reduce the number of augmenting columns if C_{ij} has more columns than rows.

- Finally, we can use an SVD decomposition of C_{ij}, viz. $C_{ij} = U_{ij}\Sigma_{ij}V_{ij}$. We then replace C_{ij} by the product $U_{ij}\Sigma_{ij}$ and $-I$ by $-V_{ij}$. We will discuss the benefits of this approach in Section 3 when investigating ways to compress information in these various augmentation approaches.

Once we build these submatrices, we place them so that they start in the same column as shown in Figure 2.1(b).

In the case of augmentation by repeating the submatrices A_{ij} and A_{ji}, the product of each pair of partitions $\bar{A}_i$ and $\bar{A}_j$ gives

$$\bar{A}_i\bar{A}_j^T = A_{ij}A_{ji}^T - A_{ij}A_{ji}^T,$$

and in the case of the augmentation using normal equations C_{ij} we obtain

$$\bar{A}_i\bar{A}_j^T = A_{ij}A_{ji}^T - C_{ij}.$$

Notice that in both cases we get $\bar{A}_i\bar{A}_j^T = 0$.

Which of these alternatives to use will depend on the structure of the overlaps and we want to choose the one that minimizes the number of augmenting columns. In the following, we first concentrate on the C_{ij} and $-I$ augmentation as shown in Figure 2.1(b) before we compare, in Table 2.2, its behaviour with the approach (2.1). To prevent the C_{ij} blocks from different neighbours creating new interconnections, we shift them side by side columnwise as shown in the illustrative example. We note that in cases where there is a largely full column in the submatrices

A_{ij}, the second formulation (2.2) might involve a full C_{ij} and so become too expensive with respect to the first formulation (2.1).

The resulting augmented matrix is in all cases of the form $\bar{A} = \begin{bmatrix} A & C \end{bmatrix}$. However, to ensure that our new system $\bar{A} \begin{bmatrix} x \\ y \end{bmatrix} = b$ has the same solution x, we add extra constraints to the system to force y to be equal to 0, resulting in the new system

$$\begin{bmatrix} A & C \\ 0 & I \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

$$\begin{array}{ccccc} \text{hence} & Ax + Cy & = & b \\ \text{and} & y & = & 0 \\ \text{thus} & Ax & = & b. \end{array}$$

Now that we have shown that we have the same solution from our new system, we have one remaining problem. The partitions in $\begin{bmatrix} A & C \end{bmatrix}$ are mutually orthogonal, but they are not orthogonal with the extra rows in the bottom partition $Y = \begin{bmatrix} 0 & I_k \end{bmatrix}$, where k is the number of columns of C. Therefore, we project Y^T onto the orthogonal complement of the upper part using the orthogonal projector

$$P = P_{\mathcal{R}(\bar{A}^T)} = P_{\bigoplus_{i=1}^{p} \mathcal{R}(\bar{A}_i^T)} = \sum_{i=1}^{p} P_{\mathcal{R}(\bar{A}_i^T)}$$

which holds as a sum because of the enforced numerical orthogonality between the blocks $\bar{A}_i$ of $\bar{A}$.

The resulting projected set of rows

$$\begin{bmatrix} B & S \end{bmatrix} = W = Y\,(I - P) \tag{2.3}$$

is orthogonal to all other partitions of $\bar{A}$. However, the right-hand side has to change to maintain the same solution, viz

$$\begin{aligned} f = \begin{bmatrix} B & S \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} &= Y\,(I - P) \begin{bmatrix} x \\ 0 \end{bmatrix} \\ &= -YP \begin{bmatrix} x \\ 0 \end{bmatrix} \\ &= -Y\bar{A}^{+}\bar{A} \begin{bmatrix} x \\ 0 \end{bmatrix} \\ &= -Y\bar{A}^{+}b \end{aligned}$$

resulting in the new linear system

$$\begin{bmatrix} A & C \\ B & S \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ f \end{bmatrix} \tag{2.4}$$

which, if $f = -Y\bar{A}^{+}b$, has the same solution as the previous one, with x corresponding to the solution of the original system (1.1).

Once the new augmented matrix has been built, we can apply block Cimmino to it by keeping the partitions that were defined for $\bar{A}$ and including W as a single partition. The eigenvalues of the iteration matrix in this case are all 1. This contrasts with the spectrum of the non-augmented system shown in Figures 1.3 and 1.4. Since all eigenvalues of the iteration matrix are 1 the block Cimmino method will converge in one step. That is, because of the orthogonality of $W = [B \;\; S]$ with $\bar{A}$, the solution is given by:

$$\begin{bmatrix} x \\ y \end{bmatrix} \;\; = \;\; \bar{A}^{+}b + W^{+}f. \tag{2.5}$$

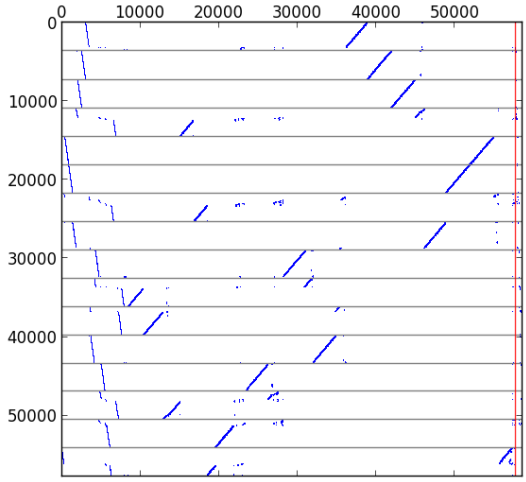


Figure 2.2: Pattern of the $[A \; C]$ part when augmenting the matrix `bayer01`.

To illustrate what we said previously, we show a picture of the partitioned $\bar{A}$ built from the `bayer01` matrix in Figure 2.2, and the S matrix generated from the augmentation process in Figure 2.3. This is a graphic illustration of the relative sizes of the unaugmented matrix and the matrix S. Although it is relatively much smaller, the dimension of S is still 918 which can be reduced to 804 by using transposes of the submatrices C_{ij} if they are rectangular and have more columns than rows (as mentioned above). Although the order of C is 1.4% of the dimension of the original matrix, the construction and solution of this matrix can still be a significant part of the overall computation which is why we seek to reduce this order in Section 3.

We show in Table 2.1 the size of S for four of our test matrices partitioned either uniformly with p equally sized partitions, or using `PaToH` with the same number of partitions. From this we see that, in the case of uniform partitioning, the dimension of S can be even greater than the original dimension and also that the strategy of using C_{ij} or C_{ij}^{T}, whichever has fewer columns (shown in the column **Reduced** S in the table), can be very beneficial.

We compare in Table 2.2 the size of S when augmenting the matrix either using the normal equations C_{ij} or using the submatrices A_{ij}/A_{ji}. We notice that most of the time the $A_{ij}/-A_{ji}$ gives a smaller S.

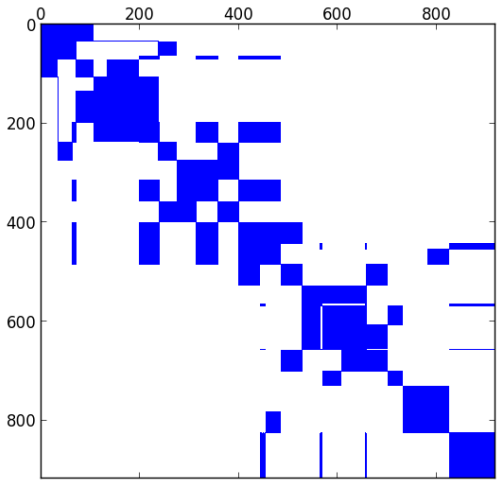


Figure 2.3: Pattern of the S matrix for the augmented system for the matrix `bayer01`.

Matrix	**Size**	**#Part.**	**Partitioning**	**Size of S**	**Reduced S**
`gre_1107`	1107	5	Uniform	1227	1138
			`PaToH`	499	373
`bayer01`	57735	16	Uniform	2951	2469
			`PaToH`	918	804
`lhr34c`	35152	8	Uniform	10 108	1803
			`PaToH`	1815	1470
`lhr71c`	70304	16	Uniform	20 283	3671
			`PaToH`	3203	2679

Table 2.1: Dimension of S for some test matrices with C_{ij} augmentation.

2.1 The matrices W and S

In this subsection, we examine some properties of the matrices W and S. The size of S depends directly on the number of columns in the C block. Thus, fewer interconnections between the partitions implies a reduced size of C and S.

Since $\begin{bmatrix} B & S \end{bmatrix} = Y(I - P)$, we see that

$$S = Y(I - P)Y^T \tag{2.6}$$

from which, as P is a projection matrix, we can immediately see that S is symmetric.

S is a restriction of the orthogonal projector $(I - P)$ whose eigenvalues are in the range $[0, 1]$. Therefore the eigenvalues of S are in the range $[0, 1]$. We see this in Figure 2.4.

From the definition of W,

$$\begin{aligned} WW^T &= [B\ S][B\ S]^T \\ &= BB^T + SS^T \\ &= BB^T + S^2 \end{aligned} \tag{2.7}$$

Matrix	Size	#Part.	Augmentation	Size of reduced S
`gre_1107`	1107	5	C_{ij} based	373
			A_{ij} based	412
`bayer01`	57735	16	C_{ij} based	804
			A_{ij} based	542
`lhr34c`	35152	8	C_{ij} based	1470
			A_{ij} based	919
`lhr71c`	70304	16	C_{ij} based	2679
			A_{ij} based	1799

Table 2.2: Comparison of the size of S with different augmentation approaches.

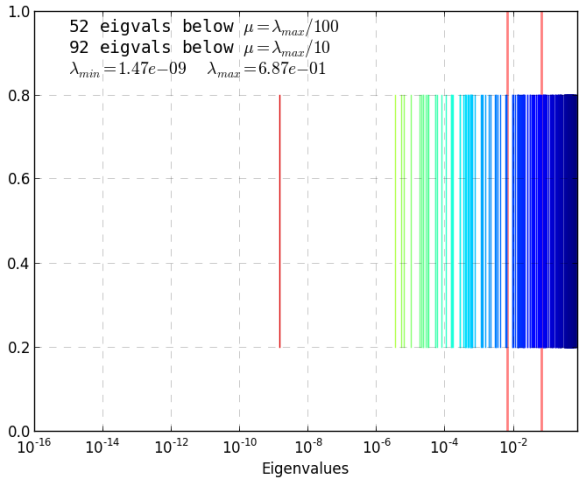


Figure 2.4: Eigenvalues of S matrix for `bayer01`.

but also from equation (2.3) and because $(I - P)$ is an orthogonal projector, we can compute WW^T as

$$\begin{aligned} WW^T &= Y\,(I-P)\,(I-P)^T\,Y^T \\ &= Y\,(I-P)^2\,Y^T \\ &= Y\,(I-P)\,Y^T \\ &= S \end{aligned} \tag{2.8}$$

and thus

$$BB^T = S - S^2 \tag{2.9}$$

(which also shows that the eigenvalues of S lie between 0 and 1).

The matrix $S = Y(I-P)Y^T$ reflects the bad conditioning of the Cimmino iteration matrix. Hopefully, the size of S is small enough to be an improvement compared to the original Cimmino iteration matrix. This helps to reduce the length of plateaux in the convergence of conjugate gradients while having Krylov spaces of smaller dimensions. This aspect is also seen in domain decomposition methods where the problem is condensed into a smaller matrix called the Schur

complement that is usually denoted by S (we have chosen the S notation as an analogy with this).

As W is a partition in our augmented block Cimmino algorithm, the solution obtained using equation (2.5) involves W^+. Since W^+ can be expressed as $W^T(WW^T)^{-1}$, we have from equation (2.8) that

$$W^+ \;=\; W^T S^{-1} \;=\; (I-P)Y^T S^{-1}. \tag{2.10}$$

This involves only S and P. Therefore, the computation using W^+ can be easily performed.

2.2 Solving the augmented system

Due to the orthogonality between the partitions of $\bar{A}$, we have

$$\bar{A}^+ b = \textstyle\sum_{i=1}^{p} \bar{A}_i^+ b_i \quad \text{and} \quad P = \textstyle\sum_{i=1}^{p} P_{\mathcal{R}(\bar{A}_i^+)} \tag{2.11}$$

which can be used jointly with equation (2.10) to build a solution of the augmented system

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

through equations (2.5) and (2.3). This solution can be expressed as

$$\begin{aligned} \begin{bmatrix} x \\ y \end{bmatrix} &= \bar{A}^+ b + W^+ f \\ &= \textstyle\sum_{i=1}^{p} \bar{A}_i^+ b_i - (I-P)\, Y^T S^{-1} Y \sum_{i=1}^{p} \bar{A}_i^+ b_i. \end{aligned}$$

An algorithm for solving this system becomes quite simple to define. This can be done in the following steps:

1. Build $w = \bar{A}^+ b$, using equation (2.11), and then by simple restriction set $f = -Yw$.
2. Solve $Sz = f$ using a direct solver as S should be small.
3. Expand $\bar{z} = Y^T z$ and then project it onto the null space of $\bar{A}$ viz. $u = (I-P)\,\bar{z}$.
4. Then sum $w + u$ to obtain the solution $\begin{bmatrix} x \\ y \end{bmatrix}$, where $y = 0$.

Note that we don't need to build B explicitly, only S is used. In that respect, since S is of smaller size and symmetric positive definite, we can either build S and factorize it or use it implicitly in a conjugate gradient procedure through matrix-vector products with $Y(I-P)Y^T$ which implies a sequence of contraction, projections and expansion of the vector.

3 Filtered augmented block Cimmino

In this section we propose an approach to reduce the order of the matrix S. We do this by reducing the number of columns of C which exactly determines the order of the matrix S. Of course, one way of doing this is to use a partitioning where there is already good orthogonality between subspaces so that there are only a few columns in C (Arioli, Drummond, Duff and Ruiz 1995*a*). We study three dropping strategies to reduce the order of S:

- We drop columns from C after it is generated, removing from C_{ij} the columns where the maximum element is smaller than some predefined threshold. Thus, the columns that have at least one element larger than the threshold are kept in C_{ij}, and we keep only the corresponding columns in the matrix $-I$. We illustrate the results of this dropping strategy in Section 3.1.

- In the case where we augment the matrix using the submatrices A_{ij}, a drop based on a straight threshold for entries in each column is not the correct way to go as it can destroy useful information present in these columns. We propose to drop a column depending on a relative scaled norm of the column. We discuss this in more detail in Section 3.2.

- We finish by studying another way to drop columns in C by building an SVD $U_{ij}\Sigma_{ij}V_{ij}^T$ of each C_{ij}. From usual data compression techniques, we then keep only the k largest singular values and the corresponding columns in U_{ij} and V_{ij}. We thus obtain a compressed matrix $U_{ij_{1,k}}\Sigma_{ij_{1,k}}V_{ij_{1,k}}^T$ of k columns. Rather than taking a fixed number of the largest columns, we can can just select those columns with singular values above a given threshold. We study both cases in Section 3.3.

The main problem, when filtering out columns in C, is that the partitions of the augmented matrix $\bar{A}$ will no longer be mutually orthogonal, so that the matrix $\sum_{i=1}^{p} P_{\mathcal{R}(\bar{A}_i^T)}$ will no longer be an orthogonal projector. Thus, when solving systems involving $\sum_{i=1}^{p} P_{\mathcal{R}(\bar{A}_i^T)}$, we will no longer get the projector $P_{\bigoplus_{i=1}^{p}\mathcal{R}(\bar{A}_i^T)}$directly. However, we can recover this, based on intrinsic properties of both the block Cimmino iterative scheme and of the conjugate gradient method on semi-positive definite systems.

Block Cimmino gives the minimum norm solution $u = \bar{A}^+ y$ to the system $\bar{A}u = y$ as shown by Elfving (1980). The iteration matrix of block Cimmino, $\sum_{i=1}^{p} \bar{A}_i^+ \bar{A}_i$, is symmetric semi-positive definite, and solving the consistent linear system

$$\sum_{i=1}^{p} \bar{A}_i^+ \bar{A}_i u = \sum_{i=1}^{p} \bar{A}_i^+ y_i$$

with conjugate gradients also yields the minimum norm solution as shown by Kaasschieter (1988). Using these properties, we are able to recover $w = \bar{A}^+ b$, needed in step 1 of our solution scheme in Section 2.2, through conjugate gradient iterations to solve the system

$$\sum_{i=1}^{p} \bar{A}_i^+ \bar{A}_i w = \sum_{i=1}^{p} \bar{A}_i^+ b_i,$$

in which the matrix is the iteration matrix of block Cimmino applied to the system

$$\bar{A}w = b \tag{3.1}$$

with the same partitioning as originally used on A.

The same issue arises in the construction of the matrix $S = Y(I - P)Y^T$ which involves the projector $P = \bar{A}^+\bar{A}$. We can use the same approach as above simply replacing the right-hand side b in (3.1) by $\bar{A}e_j$ where e_j are canonical vectors from the columns of the matrix Y^T.

Solving the system

$$\bar{A}z = \bar{A}e_j$$

gives the minimum norm solution

$$\begin{aligned} z &= \bar{A}^{+}\bar{A}e_j \\ &= Pe_j \end{aligned}$$

where e_j is a canonical vector from the matrix Y^T. Accelerating this solution using conjugate gradients amounts to solving the system

$$\sum_{i=1}^{p} \bar{A}_i^{+}\bar{A}_i z = \sum_{i=1}^{p} \bar{A}_i^{+}\bar{A}_i e_j.$$

We solve the previous system for all e_j in Y^T to obtain the matrix PY^T that can be used to build $S = Y(I-P)Y^T$ explicitly. As the solutions associated with these columns are independent from each other we may also exploit an extra level of parallelism.

If we can keep the dimension of S small enough, then it is attractive to build it explicitly and factorize it using a direct method. This is the approach we will consider in the following.

3.1 Filtering C_{ij}

We now examine the effect of dropping entries in the contributing subblocks C_{ij} on the dimension of S and on the subsequent solution of the problem.

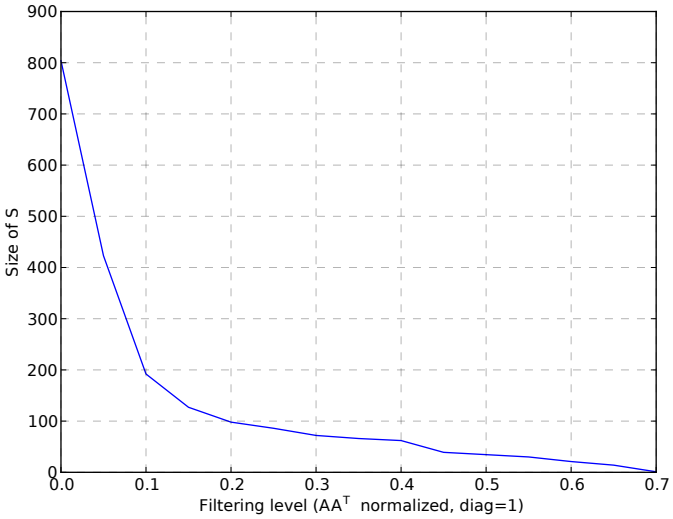


Figure 3.1: Number of columns of S against filtering level on C, for `bayer01`.

We show in Figure 3.1 the reduction in the order of S for `bayer01` as we increase the relative dropping tolerance τ for filtering out entries in the C_{ij}. Since the structure of S has large dense blocks, the value of reducing its dimension from nearly 804 to less than 192 when dropping at 0.1 is obvious. The effect of this dropping on the convergence of the block Cimmino iteration used to build $w = \bar{A}^{+}b$ is shown in Figure 3.3.

We notice that as one might expect, the more columns that we drop the slower the convergence. This is due to the fact that the more columns of C that we drop, the more we destroy the orthogonality between partitions and, in the limit, we may be just left with the original Cimmino iteration matrix and its bad conditioning. Looking at the eigenvalue distribution of

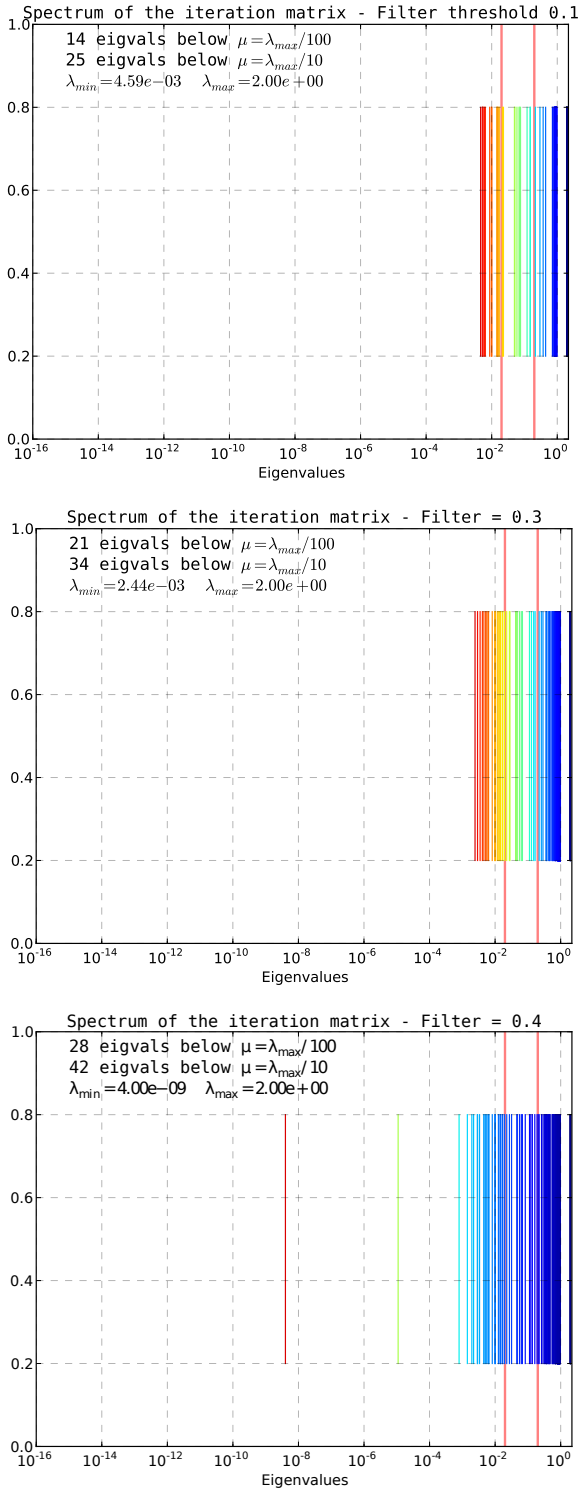


Figure 3.2: Eigenvalues of the iteration matrix after filtering C columns with different filtering thresholds.

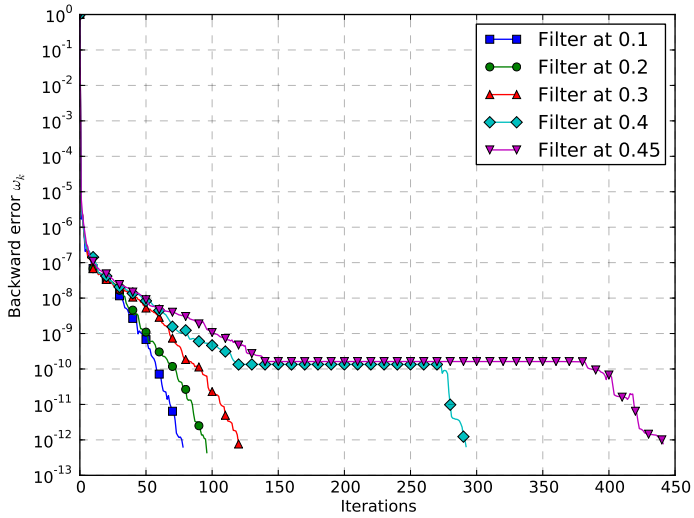


Figure 3.3: The effect of dropping columns of C on the convergence of the block Cimmino process.

Filtering threshold τ	0	0.1	0.3	0.4	0.45
Size of S	804	192	72	62	39
Min. Iterations to build S	1	4	11	13	13
Avg. Iterations to build S	1	15	26	34	62
Max. Iterations to build S	1	48	55	60	101
Nb. Iter. $w = \bar{A}^{+}b$	1	58	91	281	402
Nb. Iter. $u = (I - P)\,\bar{z}$	1	51	71	212	202
$\|r\| / (\|A\|\|x\| + \|b\|)$	$3e-16$	$1e-11$	$1e-11$	$1e-11$	$1e-11$
$\|r\|/\|b\|$	$2e-15$	$6e-11$	$6e-11$	$4e-11$	$5e-11$
$\|x^* - x\|_\infty / \|x^*\|_\infty$	$3e-11$	$4e-10$	$1e-09$	$4e-09$	$9e-09$

Table 3.1: Filtering columns of $C_{ij}/-I$ in ABCD on `bayer01`.

the iteration matrix in each case, as shown in Figure 3.2, one can see that the more we drop the more intermediate eigenvalues appear. So when the smallest eigenvalue changes from 4.59×10^{-3} to 2.44×10^{-3} (for a filtering threshold of 0.1 and 0.3 respectively), we require an additional 43 iterations. Moreover, when dropping more columns, two very small eigenvalues appear, one at 4×10^{-9} and another at 1×10^{-5}. This explains the appearance of plateaux in the convergence profile in Figure 3.3 when filtering at 0.4 and 0.45.

We summarize the results of filtering C for the `bayer01` matrix in Table 3.1. The order of S decreases quite significantly as we increase the filtering level although, after a certain point, the gains are less noticeable. As we expect, at larger filtering thresholds, when the partitions of $\bar{A}$ depart more and more from being orthogonal, we observe that CG requires more iterations. The same goes for the construction of S which requires more iterations too. The resulting accuracy is still good, even if somewhat degraded from the unfiltered case.

3.2 Filtering A_{ij}

We examine now the effect of dropping entries in the submatrices A_{ij} and its corresponding $-A_{ji}$ when using the first variant (2.1) for the augmentation process. To avoid losing important information when dropping, we look at the norm of the outer product of the k-th column from each submatrix with respect to the number of entries generated by this outer product. This is implemented by applying the following steps on each couple $A_{ij}/-A_{ji}$ in C:

- We compute the Frobenius norm of each rank-one update corresponding to the k-th column $\begin{bmatrix} A_{ij(*,k)} & -A_{ji(*,k)} \end{bmatrix}^T$. Denoting this by $frob_{ij}(k)$, we can compute it by :

$$frob_{ij}(k) = \| A_{ij(*,k)} \|_2 \| A_{ji(*,k)} \|_2.$$

- We define d_{ij}, an under-estimate of the number of entries in $A_{ij}A_{ji}^T$, by :

$$d_{ij} = \max_k \left(card\{|A_{ij}(*,k)| > 0\} \times card\{|A_{ji}(*,k)| > 0\} \right),$$

  where $card\{y\}$ is the number of entries in y.

- To identify the dominant contributions from the different rank-one updates, we first compute as a reference value the mean distribution of the rank-one contributions :

$$\nu_{ij} = \frac{\text{average}\left(frob_{ij}(k)\right)}{\sqrt{d_{ij}}}$$

- For each column, we then count the number of influential entries :

$$card_{ij}(k) = card\{\left|A_{ij(*,k)}\right| \geq \frac{\nu_{ij}}{\| A_{ji(*,k)} \|_\infty}\},$$

$$card_{ji}(k) = card\{\left|A_{ji(*,k)}\right| \geq \frac{\nu_{ij}}{\| A_{ij(*,k)} \|_\infty}\}.$$

  If $card_{ij}$, respectively $card_{ji}$, is zero, we set it to m_{ij}, respectively m_{ji}, the number of rows in A_{ij}, respectively A_{ji}.

- Next, we define for each of the columns a scaled norm

$$\mu_{ij}(k) = \frac{frob_{ij}(k)}{\sqrt{card_{ij}(k) \times card_{ji}(k)}}. \tag{3.2}$$

- Finally, using a given threshold τ we select the columns to retain in the couple $A_{ij}/-A_{ji}$ satisfying $\mu_{ij}(k) \geq \tau$.

By following these steps, we try to extract some information regarding the numerical density of each rank-one update corresponding to the columns of a given couple $A_{ij}/-A_{ij}$. This numerical density is then used to select the most influential columns in each couple (those above the threshold τ).

We show in Figure 3.4 the reduction in the order of S for `bayer01` as we increase the relative dropping tolerance τ for filtering the entries in the $A_{ij}/-A_{ji}$ couples. We show in Figure 3.5 the

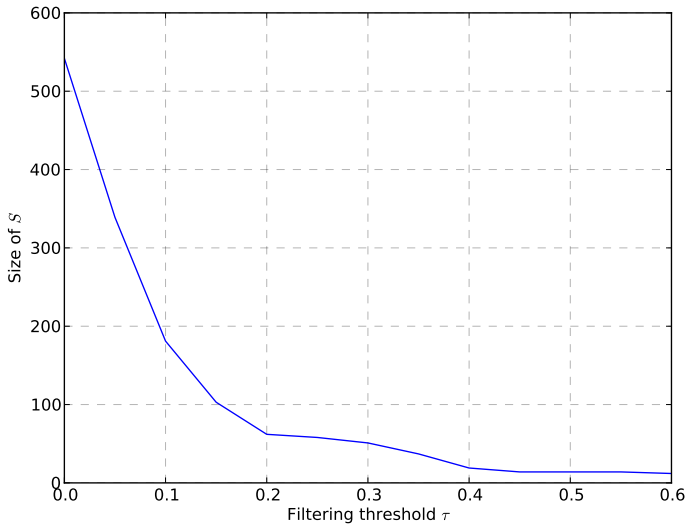


Figure 3.4: Number of columns of S against the filtering level on A_{ij}, for `bayer01`.

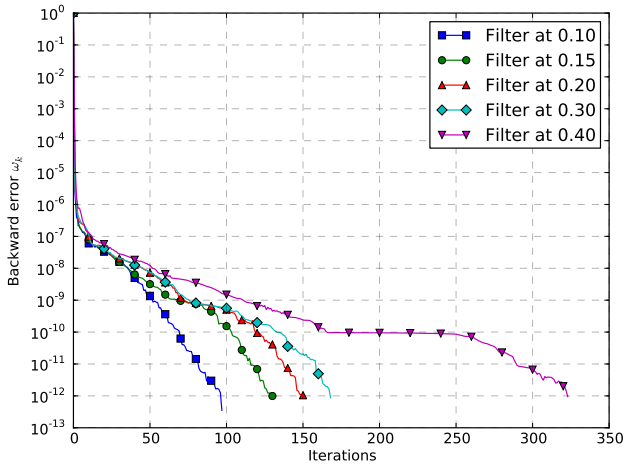


Figure 3.5: The effect of dropping columns on the convergence of the block Cimmino process.

effect of this dropping strategy on the block Cimmino iteration cost when computing $w = \bar{A}^+ b$. More detailed results are summarized in Table 3.2.

We notice the same behaviour as when filtering the C_{ij} normal equations subblocks in the sense that, the more we drop the slower the convergence. We notice that we require 152 iterations to build w with the $A_{ij}/-A_{ji}$ strategy when we have 62 columns in S, while we require 281 iterations using the $C_{ij}/-I$ strategy with the same number of columns in S. We are also able to reduce the number of columns to 19 without seeing too strong a degradation in the convergence while still keeping good numerical properties.

We compare in Figure 3.6 the values of two different scalings of the $frob_{ij}(k)$ norms, $\mu_{ij}(k)$ and $\bar{\mu}_{ij}(k) = \frac{frob_{ij}(k)}{\sqrt{nnz\{A_{ij(:,k)}\} \times nnz\{A_{ji(:,k)}\}}}$. We notice that using the scaled norms $\mu_{ij}(k)$ we retain more columns since nnz is the total number of entries in the column whereas $card_{ji}(k)$ in equation

Filtering threshold τ	0	0.1	0.15	0.2	0.3	0.4
Size of S	542	181	103	62	50	19
Min. Iterations to build S	1	4	4	4	6	14
Avg. Iterations to build S	1	18	23	25	29	72
Max. Iterations to build S	1	51	55	63	63	120
Nb. Iter. $w = \bar{A}^{+}b$	1	97	130	152	168	323
Nb. Iter. $u = (I - P)\,\bar{z}$	1	91	121	141	160	76
$\|r\|/\left(\|A\|\|x\| + \|b\|\right)$	$6e-16$	$1e-11$	$1e-11$	$1e-11$	$1e-11$	$3e-11$
$\|r\|/\|b\|$	$3e-15$	$5e-11$	$6e-11$	$6e-11$	$6e-11$	$8e-11$
$\|x^* - x\|_\infty/\|x^*\|_\infty$	$2e-11$	$1e-10$	$2e-10$	$5e-10$	$6e-10$	$1e-09$

Table 3.2: Filtering columns of $A_{ij}/-A_{ji}$ in ABCD on `bayer01`.

(3.2) is only the number of entries in the column greater than a threshold. We show the effect of using these two measures for the scaled norm of the rank-one updates in Figure 3.6 where we have ordered the columns using the scaled norm $\bar{\mu}_{ij}(k)$. We then see that, when filtering at 0.4 for example, there are four spikes that increase the size of S from 15, when using $\bar{\mu}_{ij}(k)$, to 19. When we look at the number of iterations, we notice that adding those four columns reduces the number of iterations from 498, when using $\bar{\mu}_{ij}$, to 323 so clearly the influence of these four columns is a strong one. We thus use the scaled norm $\mu_{ij}(k)$ to determine which columns to retain when using the couples $A_{ij}/-A_{ji}$.

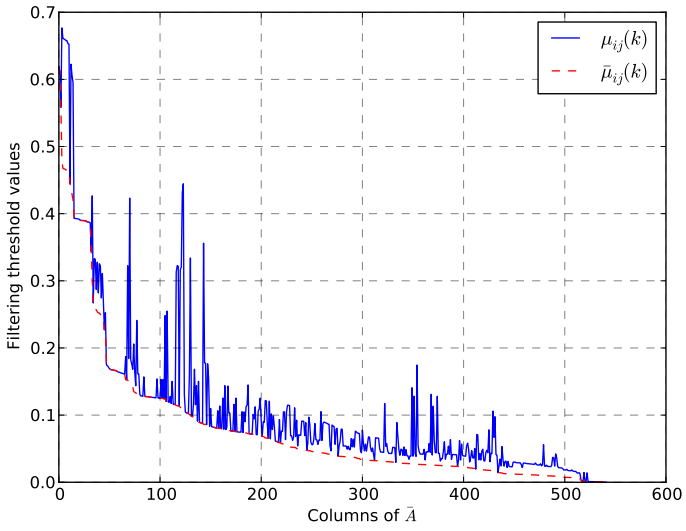


Figure 3.6: Comparison of two different scaled norms for each rank-one update in $A_{ij}/-A_{ji}$.

3.3 Compressing C_{ij} with SVD

We now study the SVD approach in which we decompose each C_{ij} into $U_{ij}\Sigma_{ij}V_{ij}^T$. We then try to select the minimum number of columns with enough information to maintain good convergence and reduce at the same time the order of S. The resulting augmented matrix is of the following

form:

$$\left[\begin{array}{ccccc|cc} A_{1,1} & A_{1,2} & & & & \hat{U}_{1,2}\hat{\Sigma}_{1,2} & \\ & A_{2,1} & A_{2,2} & A_{2,3} & & -\hat{V}_{1,2} & \hat{U}_{2,3}\hat{\Sigma}_{2,3} \\ & & & A_{3,2} & A_{3,3} & & -\hat{V}_{2,3} \end{array}\right] \tag{3.3}$$

where $\hat{U}_{ij}$, $\hat{\Sigma}_{ij}$ and $\hat{V}_{ij}$ are respectively the reduced U_{ij}, Σ_{ij} and V_{ij}, with only those selected columns.

We look, first, at a simple case where we compute the k largest singular values and build the augmented matrix as in (3.3). The results from this approach are shown in Table 3.3. We notice that the results are slightly worse than when filtering using $C_{ij}/-I$ or $A_{ij}/-A_{ji}$ for comparable size of S.

Singular values kept per block	5	10	12	15
Size of S	108	209	249	308
Nb. Iter. $w = \bar{A}^{+}b$	188	102	94	46

Table 3.3: Selecting k largest singular values from C_{ij} in ABCD on `bayer01`.

We try now to improve the previous selection by looking at the smallest singular value within all the set of selected columns. This is implemented as follows:

- Select k columns from all the SVD decompositions applied to the C_{ij} blocks.
- Find the smallest singular value among them.
- Use it as a threshold to select further columns whose singular value is larger than this value.

We show in Table 3.4, the results obtained on the `bayer01` matrix when using this approach. For instance, by selecting 2 columns initially, we see that the smallest singular value is 0.2. Then, we select all columns whose corresponding singular value is larger than 0.2. The results show that we are able to improve the previous results for comparable sizes of S. However, we lose the ability to control explicitly the number of columns.

Singular values initially selected per block	2	3	4	5
Smallest singular value	0.2	0.11	0.049	0.02
Size of S	96	171	376	481
Nb. Iter. $w = \bar{A}^{+}b$	141	107	45	12

Table 3.4: Selecting singular values larger than a threshold from C_{ij} in ABCD on `bayer01`.

3.4 Cost analysis

An important aspect of the ABCD method is the reusability of the S matrix for multiple, successive, solves for different right-hand sides. To evaluate the gains obtained using our method, we look at the operation counts needed to solve several, successive, right-hand sides using classical block Cimmino, and using the ABCD method with and without filtering.

We first define the cost of each solution. The costly part in each block Cimmino iteration is the computation of $P_{\bigoplus_{i=1}^{p}\mathcal{R}(A_i^T)}y$. Its cost is basically the number of flops needed by a sparse direct solver to compute a forward-backward substitution. We will denote this cost by $\mathcal{C}_{BC}(A)$. In the case of ABCD, the cost of computing $w = \sum_{i=1}^{p}\bar{A}_i^+ b_i$ is $\mathcal{C}_{BC}(\bar{A})$ per iteration. We can obtain a value for these quantities from MUMPS.

The initial cost of solving $Sz = f$ is a combination of a Cholesky factorization on S with a forward-backward substitution. As we treat the matrix S as a dense matrix, the estimation of the first step is

$$\mathcal{C}_{S,init} = \frac{1}{3}n_s^3 + 2n_s^2,$$

while for the later steps we need only a forward-backward substitution, a bound on this cost is

$$\mathcal{C}_S = 2n_S^2.$$

We show in Table 3.5 the estimations for solving a linear system, with multiple right-hand sides successively, using the classical block Cimmino method, ABCD without filtering and ABCD with different filtering thresholds. We denote by it_{bc} the number of iterations needed to solve a single right-hand side using block Cimmino, by it_p the number of iterations to compute both $w = \sum_{i=1}^{p}\bar{A}_i^+ b_i$ and $u = (I - P)\bar{z}$ and by it_s the sum over the number of iterations needed to build all the columns of S. In the case where we use ABCD without filtering, the number of iterations it_p is equal to 2 as we need a single one to build w and another to build u.

	The cost estimate in floating-point operations	
	First right-hand side	Next right-hand sides
Block Cimmino	$it_{bc} \times \mathcal{C}_{BC}(A)$	$it_{bc} \times \mathcal{C}_{BC}(A)$
ABCD (no filtering)	$(n_s + 2) \times \mathcal{C}_{BC}(\bar{A}) + \mathcal{C}_{S,init}$	$2 \times \mathcal{C}_{BC}(\bar{A}) + \mathcal{C}_S$
ABCD (with filtering)	$(it_s + it_p) \times \mathcal{C}_{BC}(\bar{A}) + \mathcal{C}_{S,init}$	$it_p \times \mathcal{C}_{BC}(\bar{A}) + \mathcal{C}_S$

Table 3.5: Estimation of flops needed per step.

We use the estimations from Table 3.5 to generate the graphs in Figure 3.7 where we show the amount of work needed to solve multiple right-hand sides successively when using the $C_{ij}/-I$ augmentation strategy, where we compare the plain block Cimmino on the original matrix, the augmented version without filtering, and with multiple filtering thresholds. We illustrate in this figure how quickly one may expect to recover the extra work induced by the augmentation process.

We notice that using the full augmentation process without filtering recovers this extra work after just two solutions. Further solutions require very much less work, they require only one block Cimmino iteration plus the cost of a Forward/Backward substitution, making the graph look flat.

Filtering at 0.1 requires more work at the first run. Indeed, the number of columns in S combined with the number of iterations to build a single column of S makes the initial cost very large. This initial cost is less when dropping at 0.3, as the size of S is dramatically reduced while the number of iterations increases only slightly. But this cost goes back up when dropping further, the number of iterations needed for each column increases faster than the reduction in the number of columns in S.

With higher filtering thresholds, it takes more solutions to recover the original cost, and each of these solutions requires more CG iterations, which can be seen in the increased slope of the

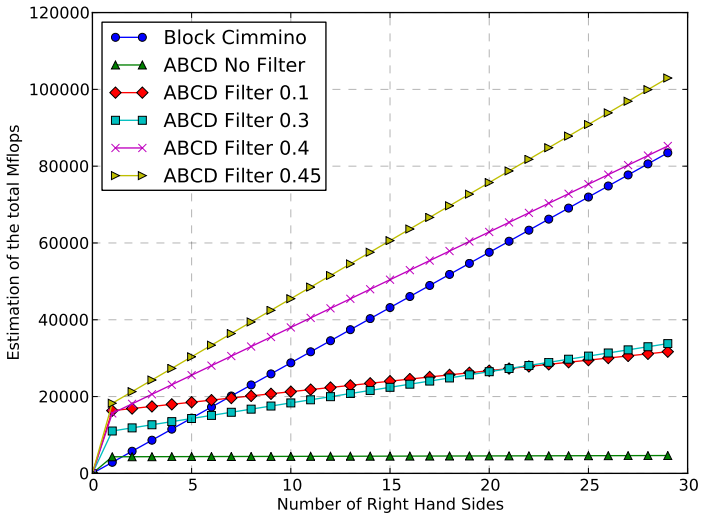


Figure 3.7: Comparison of solving multiple right-hand sides successively when using $C_{ij}/-I$ couples.

corresponding straight lines in Figure 3.7. We observe that when the filtering threshold gets too large, the gain per solution is not so effective. When using a filter level of 0.4, we require around 30 solutions to compensate for the extra work which is not unreasonable given the low memory requirements at that level of filtering. Finally, increasing the filtering threshold even further makes the approach useless or counterproductive with respect to the plain block Cimmino solve.

We show in Figure 3.8 the costs when using the $A_{ij}/-A_{ji}$ augmentation strategy. Without filtering, we perform much better than the original block Cimmino iteration. This is due to the fact that S very small. However, and in contrary to what we have seen with the $C_{ij}/-I$ augmentation strategy, the combination of the small size of S and the small number of iterations to build columns of S makes it possible to use a filter threshold of 0.4, for which the size of S is only 19, and we recover the cost of generating S after only 10 subsequent solves. In this augmentation strategy it is possible to reduce the storage for S while requiring relatively few iterations to build a single column.

4 Conclusions

The block Cimmino method for solving sets of sparse linear equations has the merit of being embarrassingly parallel, but its convergence is equivalent to block Jacobi on the normal equations and can be slow. Preprocessing techniques, partitioning strategies and the use of block conjugate gradients can improve its convergence. However, all of these have their drawbacks or limitations.

We have proposed a novel technique to augment the systems to force the block Cimmino subspaces to be orthogonal so that only one iteration is required. This involves the solution of a relatively smaller positive definite system that we solve by forming the matrix and using a direct solver. We showed that this method is particularly useful when solving multiple successive right hand-sides. When this auxiliary system is small, the method works well. However, when it is large, the memory costs can become prohibitive.

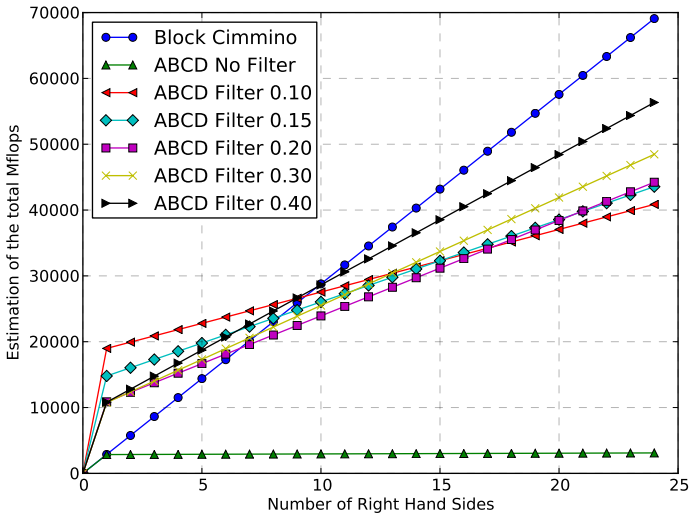


Figure 3.8: Comparison of solving multiple right-hand sides successively when using $A_{ij}/-A_{ji}$ couples.

We have then looked at methods to reduce the size of this auxiliary system at the cost of losing partly the orthogonality obtained by the augmentation and thus increasing the number of iterations for solution.

This study opens up new doors for future investigations. A parallel computation of the auxiliary system can indeed be easily conducted as the computation of each column in this auxiliary matrix is independent. This independence between the columns is the first level of parallelism, and can be supported by other levels of parallelism through the Cimmino blocking and the direct solver.

Further investigations concern the solution of the auxiliary system implicitly (without building it) inside the block Cimmino iterations. This can be performed by an inner iteration of conjugate gradients. A preconditioner has to be used as the matrix is ill conditioned.

References

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

Arioli, M., Drummond, A., Duff, I. S. and Ruiz, D. (1995*a*), Parallel block iterative solvers for heterogeneous computing environments, *in* M. Moonen and F. Catthoor, eds, 'Algorithms and Parallel VLSI Architectures III', Elsevier, Amsterdam, pp. 97–108.

Arioli, M., Duff, I. S., Noailles, J. and Ruiz, D. (1992), 'A block projection method for sparse matrices', *SIAM J. Scientific and Statistical Computing* **13**, 47–70.

Arioli, M., Duff, I. S., Ruiz, D. and Sadkane, M. (1995*b*), 'Block Lanczos techniques for accelerating the Block Cimmino method', *SIAM J. Scientific Computing* **16**(6), 1478–1511.

Catalyürek, U. and Aykanat, C. (1999), 'Patoh: A multilevel hypergraph partitioning tool, version 3.0', *Bilkent University, Department of Computer Engineering, Ankara.*

Chan, W. M. and George, A. (1980), 'A linear time implementation of the reverse Cuthill-Mckee algorithm', *BIT* **20**, 8–14.

Davis, T. A. (2008), 'University of Florida sparse matrix collection, `http://www.cise.ufl.edu/research/sparse/matrices/`'.

Drummond, L. A., Duff, I. S. and Ruiz, D. (1993), A parallel distributed implementation of the block conjugate gradient algorithm, Technical Report TR/PA/93/02, CERFACS, Toulouse, France.

Elfving, T. (1980), 'Block-iterative methods for consistent and inconsistent linear equations', *Numerische Mathematik* **35**(1), 1–12.

Elfving, T. (1998), 'A stationary iterative pseudoinverse algorithm', *BIT* **38**(2), 275–282.

Golub, G. H., Ruiz, D. and Touhami, A. (2007), 'A hybrid approach combining Chebyshev filter and conjugate gradient for solving linear systems with multiple right-hand sides', *SIAM J. Matrix Analysis and Applications* **29**(3), 774–795.

Kaasschieter, E. (1988), 'Preconditioned conjugate gradients for solving singular systems', *Journal of Computational and Applied mathematics* **24**(1), 265–275.

Ruiz, D. F. (1992), Solution of large sparse unsymmetric linear systems with a block iterative method in a multiprocessor environment, Phd thesis, Institut National Polytechnique de Toulouse. CERFACS Technical Report, TH/PA/92/06.