Science & Technology Facilities Council

Preprint

RAL-P-2014-005

# Partitioning strategies for the Block Cimmino algorithm

T Drummond, IS Duff, R Guivarch, D Ruiz, M Zenadi

June 2014

Submitted for publication in Journal of Engineering Mathematics

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**

# Partitioning strategies for the block Cimmino algorithm

Tony Drummond[1], Iain S. Duff[2,3], Ronan Guivarch[4], Daniel Ruiz[4] and Mohamed Zenadi[4]

**ABSTRACT**

In the context of the block Cimmino algorithm, we study preprocessing strategies to obtain block partitionings that can be applied to general linear systems of equations $\mathbf{Ax} = \mathbf{b}$. We study strategies that transform the matrix $\mathbf{AA}^T$ into a matrix with a block tridiagonal structure. This provides a partitioning of the linear system for row projection methods because block Cimmino is essentially equivalent to block Jacobi on the normal equations and the resulting partition will yield a two-block partition of the original matrix. Therefore the resulting block partitioning should improve the rate of convergence of block row projection methods such as block Cimmino.

We discuss a way of obtaining a partitioning using a dropping strategy that gives more blocks at the cost of relaxing the two-block partitioning. We then use a hypergraph partitioning that works directly on the matrix $\mathbf{A}$ to reduce directly the connections between blocks.

We give numerical results showing the performance of these techniques both in their effect on the convergence of the block Cimmino algorithm and in their ability to exploit parallelism.

**Keywords:** sparse matrices, unsymmetric matrices, iterative methods, Cuthill McKee, hypergraph partitioning

**AMS(MOS) subject classifications:** 65F05, 65F50

---

This report is a significantly revised version of the report RAL-P-2013-010.

[1]LADrummond@lbl.gov, MS 50F, Lawrence Berkeley National Laboratory, One Cyclotron Road, Berkeley, CA 94720, USA.

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

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

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

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

May 9, 2014

# Contents

# 1 Introduction

The benefit of the block Cimmino algorithm for solving sparse linear systems is that the solution process reduces to the solution of a sequence of much smaller independent systems within a very simple iterative scheme. We describe this algorithm in Section 2. By using this algorithm it is possible to extend techniques suitable for smaller systems, for example sparse direct methods, to much larger systems. The Achilles heel of such an approach is that the iterative method is effectively a block Jacobi method on the normal equations and so can suffer from slow convergence.

In this paper, we examine how novel partitioning schemes can be used to accelerate the convergence of the block Cimmino algorithm while, at the same time, improving the capability of the method in exploiting parallelism.

After our description of the algorithm in Section 2, we indicate, in Section 3, why we target orderings to block tridiagonal form and how this can be used to find a partitioning to reduce the iteration count of block Cimmino. We then discuss two algorithms for obtaining such a form in Sections 4.1 and 4.2. In Section 4.3, we describe a different approach to obtain a partitioning.

We then describe experiments comparing these approaches and show their efficacy on real problems in Section 5. We present our conclusions in Section 6.

# 2 Block Cimmino method

We consider a block projection method for the solution of the linear system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.1}$$

where $\mathbf{A}$ is a nonsingular large sparse unsymmetric matrix of order $n$ although we note that the approach can also be used when the system is rectangular.

The blocks are obtained by partitioning the system (2.1) into $p$ strips of rows, with $p \leq n$, as in:

$$\begin{pmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \vdots \\ \mathbf{A}_p \end{pmatrix} \mathbf{x} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_p \end{pmatrix}. \tag{2.2}$$

The block Cimmino method projects the current iterate simultaneously onto the manifolds corresponding to the strips and takes a convex combination of all the resulting vectors. That is the algorithm computes a solution iteratively from an initial estimate $\mathbf{x}^{(0)}$ according to:

$$\mathbf{u}_i = \mathbf{A}_i^+ \left(\mathbf{b}_i - \mathbf{A}_i \mathbf{x}^{(k)}\right) \quad i = 1, ....p \tag{2.3}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \nu \sum_{i=1}^{p} \mathbf{u}_i \tag{2.4}$$

where $\mathbf{A}_i^+$ is the Moore-Penrose pseudo-inverse of the matrix $\mathbf{A}_i$.

A very attractive aspect of the equations (2.3) is that the $p$ sets of equations are independent of each other so that we can solve them simultaneously on a parallel computer.

The iteration equations (2.3) and (4) can be written as:

$$\begin{aligned} \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \nu \sum_{i=1}^{p} \mathbf{A}_i^+ \left( \mathbf{b}_i - \mathbf{A}_i \mathbf{x}^{(k)} \right) && (2.5) \\ &= \left( I - \nu \sum_{i=1}^{p} \mathbf{A}_i^+ \mathbf{A}_i \right) \mathbf{x}^{(k)} + \nu \sum_{i=1}^{p} \mathbf{A}_i^+ \mathbf{b}_i && (2.6) \\ &= \mathbf{Q}\mathbf{x}^{(k)} + \nu \sum_{i=1}^{p} \mathbf{A}_i^+ \mathbf{b}_i, && (2.7) \end{aligned}$$

so, if we define the iteration matrix $\mathbf{H}$ by $\mathbf{H} = \mathbf{I} - \mathbf{Q}$, then

$$\mathbf{H} = \nu \sum_{i=1}^{p} \mathcal{P}_{\mathcal{R}(\mathbf{A_i^T})}$$

where $\mathcal{P}_{\mathcal{R}(\mathbf{A}_i^T)}$ is a projection matrix onto the range of $\mathbf{A}_i^T$. We thus must solve the equations

$$\mathbf{H}\mathbf{x} \;=\; \nu \sum_{i=1}^{p} \mathbf{A}_i^+ \mathbf{b}_i. \tag{2.8}$$

We note that, because the matrix $\mathbf{H}$ in equation (2.8) is a sum of projection matrices, it is symmetric positive definite. We are thus able to use the conjugate gradient algorithm to solve equation (2.8). We also note that, as $\nu$ appears on both sides of equation (2.8), it can be set to 1.

At each step of the conjugate gradient algorithm we must solve for the $p$ projections in equations (2.3) viz.

$$\mathbf{A}_i \mathbf{u}_i \;=\; \mathbf{r}_i, \quad (\mathbf{r}_i = \mathbf{b}_i - \mathbf{A}_i \mathbf{x}^{(k)}), \quad i = 1, ....p. \tag{2.9}$$

In our approach we choose to solve these equations using the augmented system

$$\begin{pmatrix} \mathbf{I} & \mathbf{A}_i^T \\ \mathbf{A}_i & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{u}_i \\ \mathbf{v}_i \end{pmatrix} \;=\; \begin{pmatrix} \mathbf{0} \\ \mathbf{r}_i \end{pmatrix} \tag{2.10}$$

that we will solve using a direct method. We use the multifrontal parallel solver (MUMPS) (Amestoy, Duff, L'Excellent and Koster 2001) to do this. The main other techniques for solving equation (2.9) are using normal equations or a QR factorization. The former has numerical and storage issues while the latter lacks a good distributed solver. We avoid both problems with our approach.

A general study of the convergence of the block Cimmino method and of other related block-row and block-column methods can be found in Elfving (1980). A block SSOR algorithm accelerated by conjugate gradients is introduced by Kamath and Sameh (1988). Several other papers have discussed block SSOR including comparisons with block Cimmino (Arioli, Duff, Noailles and Ruiz 1992*b*, Bramley 1989, Bramley and Sameh 1992, Ruiz 1992). More recent

work on accelerating the algorithm can be found in Duff, Guivarch, Ruiz and Zenadi (2013) where the system (2.1) is augmented so that the blocks in the partitions are mutually orthogonal. Determining the solution to the original system then reduces to solving a much smaller but denser positive definite system. Our aim in this paper is to focus on preprocessing and partitioning strategies of the original system (2.1) to bring it to the form (2.2).

If the matrix $\mathbf{A}$ is ill conditioned then there are some rows that are almost linearly dependent and, as mentioned by Bramley and Sameh (1992), these linear combinations may occur within a block or across several blocks after row partitionings of the form (2.2). Assuming that the projections in the block Cimmino algorithm are computed exactly on the subspaces (say by using a direct method), then the rate of convergence of the block Cimmino algorithm depends only on the conditioning across the blocks. If we consider additionally conjugate gradient acceleration of the block Cimmino method (Arioli et al. 1992*b*, Bramley and Sameh 1992), the convergence behaviour of the resulting method is directly linked to the spectrum of the $n \times n$ matrix $\mathbf{H}$ formed as the sum of the previously mentioned projections and given by

$$\mathbf{H} = \sum_{i=1}^{p} {\mathbf{A}_i}^T \left(\mathbf{A}_i \mathbf{A}_i^T\right)^{-1} \mathbf{A}_i \ , \tag{2.11}$$

since, as $\mathbf{A}$ is nonsingular, the block-rows $\mathbf{A}_i$ have full row rank. An efficient implementation of block Cimmino requires a combination of a robust method for computing the projections and a partitioning strategy that minimizes the ill-conditioning across the blocks.

Let us see now how the ill-conditioning across the blocks can be expressed. Let the matrix be partitioned as in (2.2), and let the **QR** decomposition of the blocks ${\mathbf{A}_i}^T$ be given by

$$\begin{aligned}
&{\mathbf{A}_i}^T = \mathbf{Q}_i \mathbf{R}_i, && i = 1, \ldots, p \ \text{ where } \mathbf{A}_i \text{ is an } m_i \times n \text{ matrix of full row rank} \\
&\mathbf{Q}_i \quad n \times m_i, && {\mathbf{Q}_i}^T \mathbf{Q}_i = \mathbf{I}_{m_i \times m_i} \\
&\mathbf{R}_i \quad m_i \times m_i, && \mathbf{R}_i \text{ is a nonsingular upper triangular matrix;}
\end{aligned}$$

then:

$$\begin{aligned}
\mathbf{H} &= \sum_{i=1}^{p} \mathbf{A}_i^T \left(\mathbf{A}_i \mathbf{A}_i^T\right)^{-1} \mathbf{A}_i \\
&= \sum_{i=1}^{p} \mathbf{Q}_i \mathbf{Q}_i^T \\
&= \left(\mathbf{Q}_1, \cdots, \mathbf{Q}_p\right) \left(\mathbf{Q}_1, \cdots, \mathbf{Q}_p\right)^T .
\end{aligned} \tag{2.12}$$

From the theory of the singular value decomposition (Golub and Kahan 1965, Golub and Van Loan 2013), the nonzero eigenvalues of $\left(\mathbf{Q}_1, \cdots, \mathbf{Q}_p\right) \left(\mathbf{Q}_1, \cdots, \mathbf{Q}_p\right)^T$ are also the nonzero eigenvalues of $\left(\mathbf{Q}_1, \cdots, \mathbf{Q}_p\right)^T \left(\mathbf{Q}_1, \cdots, \mathbf{Q}_p\right)$.

Thus the spectrum of the matrix $\mathbf{H}$ is the same as that of the matrix

$$\begin{pmatrix} \mathbf{I}_{m_1\times m_1} & {\mathbf{Q}_1}^T\mathbf{Q}_2 & \cdots & \cdots & {\mathbf{Q}_1}^T\mathbf{Q}_p \\ {\mathbf{Q}_2}^T\mathbf{Q}_1 & \mathbf{I}_{m_2\times m_2} & {\mathbf{Q}_2}^T\mathbf{Q}_3 & \cdots & {\mathbf{Q}_2}^T\mathbf{Q}_p \\ \vdots & & \ddots & & \vdots \\ {\mathbf{Q}_p}^T\mathbf{Q}_1 & & \cdots & & \mathbf{I}_{m_p\times m_p} \end{pmatrix}, \tag{2.13}$$

where the ${\mathbf{Q}_i}^T\mathbf{Q}_j$ are matrices whose singular values represent the cosines of the principal angles between the subspaces $\mathcal{R}({\mathbf{A}_i}^T)$ and $\mathcal{R}({\mathbf{A}_j}^T)$ (Björck and Golub 1973).

Note that the principal angles satisfy $0 \leq \Psi_1 \leq \cdots \leq \Psi_{m_{ij}} \leq \pi/2$, and that having $\Psi_k = \pi/2$, $k = 1, \ldots, m_{ij}$, is equivalent to $\mathcal{R}({\mathbf{A}_i}^T)$ being orthogonal to $\mathcal{R}({\mathbf{A}_j}^T)$. Intuitively, the wider the principal angles between the subspaces, the closer $\mathbf{H}$ is to the identity matrix, and the faster the convergence of the conjugate gradient acceleration.

Nevertheless, even the knowledge of the principal angles between every pair of subspaces would not in general give us any a priori information about the spectrum and the ill-conditioning of the resulting matrix $\mathbf{H}$. In the next section, we analyse the case for only two blocks and show that there exists a strong relationship between these principal angles and the spectrum of the iteration matrix $\mathbf{H}$.

# 3 Block tridiagonal structures and two-block partitioning

If the block rows $\mathbf{A}_i$ are nearly mutually orthogonal, i.e. $\mathbf{A}\mathbf{A}^T$ is strongly block-diagonally dominant, we can expect that the method will converge very quickly if the projections are computed accurately. Conversely, the structure of $\mathbf{A}\mathbf{A}^T$ tells us about the orthogonality of the subspaces represented by block partitions of $\mathbf{A}$. If the block $(i, j)$th entry of $\mathbf{A}\mathbf{A}^T$ is zero then the subspaces corresponding to the blocks $\mathbf{A}_i$ and $\mathbf{A}_j$ are orthogonal. Thus, if $\mathbf{A}\mathbf{A}^T$ is block tridiagonal, the blocks of $\mathbf{A}$ are such that the even numbered blocks are orthogonal as are also the odd-numbered blocks. Thus if we solve the projected subproblems accurately (using say a direct method) then we also solve the subproblems corresponding to the odd and even numbered blocks accurately. To this end, we can exploit algorithms which maximize the minimum element on the diagonal, and which reorder $\mathbf{A}$ to block tridiagonal form (Ruiz 1992). The case of block tridiagonal matrices, which is common in PDE discretization problems for instance, is very important because, for such structures, we can introduce a partitioning with a good degree of parallelism and a convergence directly related to the principal angles described in the previous section.

Consider, for instance, a block tridiagonal matrix $\mathbf{A}$ partitioned in block rows $\mathbf{A}_i$. It can be

reordered as

$$\mathbf{A} = \begin{bmatrix} * & * & & & & & & & & & & \\ * & * & * & & & & & & & & & \\ & * & * & * & & & & & & & & \\ & & & * & * & * & & & & & & \\ & & & & * & * & * & & & & & \\ & & & & & * & * & * & & & & \\ & & & & & & & * & * & * & & \\ & & & & & & & & * & * & & \\ & & & * & * & * & & & & & & \\ & & & & * & * & * & & & & & \\ & & & & & & & * & * & * & & \\ & & & & & & & & * & * & * & \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_3 \\ \mathbf{A}_5 \\ \mathbf{A}_2 \\ \mathbf{A}_4 \end{bmatrix} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \end{bmatrix}, \tag{3.1}$$

where $\mathbf{B}_1 = \{\mathbf{A}_i \ : \ i \text{ odd}\}$ and $\mathbf{B}_2 = \{\mathbf{A}_i \ : \ i \text{ even}\}$. Note that, in this partitioning of $\mathbf{A}$, the blocks in $\mathbf{B}_1$ each comprise three block rows from the block tridiagonal form while those in $\mathbf{B}_2$ comprise two. The partitioning in equation (3.1) is called a two-block partitioning. We denote by $n_1$ and $n_2$ the number of rows in $\mathbf{B}_1$ and $\mathbf{B}_2$ respectively.

With such a partitioning, because of the structural orthogonality between the blocks $\mathbf{A}_i$ within $\mathbf{B}_1$ and within $\mathbf{B}_2$ respectively, the matrix $\mathbf{H}$ in (2.11) can be considered as $\mathbf{P}_1 + \mathbf{P}_2$, with $\mathbf{P}_1 = \mathbf{P}_{\mathcal{R}({\mathbf{B}_1}^T)}$ and $\mathbf{P}_2 = \mathbf{P}_{\mathcal{R}({\mathbf{B}_2}^T)}$, where each of these two projectors is in fact a sum of independent projectors acting on orthogonal subspaces.

It was shown by Elfving (1980) that, with such a two-block partitioning, the spectrum of matrix $\mathbf{H}$ is

$$\begin{array}{lll} \lambda_k = 1 + \cos\Psi_k & & k = 1, \ldots, n_2 \\ \lambda_k = 1 - \cos\Psi_{k-n_2} & & k = n_2+1, \ldots, 2n_2 \\ \lambda_k = 1 & & k = 2n_2+1, \ldots, n \end{array} \tag{3.2}$$

where $\{\Psi_k\}_1^{n_2}$ are the principal angles between $\mathcal{R}({\mathbf{B}_1}^T)$ and $\mathcal{R}({\mathbf{B}_2}^T)$.

On the one hand, matrices in this form can be easily partitioned into sufficient blocks to utilize all the processes of the target machine (provided the matrix $\mathbf{A}$ is large enough in comparison with the size of the tridiagonal substructure). On the other hand, the expression (3.2) for the spectrum tells us that the block Cimmino algorithm with conjugate gradient acceleration in general works better and, in exact arithmetic, takes not more than $2n_2$ steps for convergence where, without loss of generality, we assume $n_2 < n_1$.

The idea of exploiting such sparsity structures appropriately in row-projection methods has already been widely studied and discussed. For instance Kamath and Sameh (1988) have developed a three-block partitioning strategy of this type which they have exploited within their block SSOR algorithm accelerated with conjugate gradients. Further work and discussions can also be found in Bramley (1989) and Bramley and Sameh (1992). In Arioli et al. (1992*b*), some particular scalings used in conjunction with two-block partitioning strategies are investigated. These scalings help to open the principal angles between the two subspaces $\mathcal{R}({\mathbf{B}_1}^T)$ and $\mathcal{R}({\mathbf{B}_2}^T)$ and can be related to oblique projections associated with ellipsoidal norms.

# 4 Partitioning Strategies

We now discuss three strategies for preprocessing the matrix to obtain a partitioned system (2.2) that will normally yield better convergence of the block Cimmino algorithm than just using a naive partitioning.

## 4.1 Partitioning Strategy I

As discussed in the previous section, the main idea behind the so called two-block partitioning strategy is to exploit structural orthogonality between the subspaces $\mathcal{R}({\mathbf{A}_i}^T)$ defined by the partitioning (2.2). We have indicated that this structural orthogonality can be analysed on the basis of the sparsity pattern of the normal equations matrix $\mathbf{A}\mathbf{A}^T$. For example, for two-block partitionings of the type described above, the sparsity pattern of matrix $\mathbf{A}\mathbf{A}^T$ is block tridiagonal, with diagonal blocks of a size corresponding to the number of rows in each block $\mathbf{A}_i$ defined by the partitioning (2.2).

This simple remark can be used to define a preprocessing strategy that will enable the construction of two-block partitionings for sparse matrices with any type of sparsity structure (Arioli, Drummond, Duff and Ruiz 1995*a*). The idea is to first permute the rows of the matrix $\mathbf{A}$ based on permutations that transform the normal equations matrix $\mathbf{A}\mathbf{A}^T$ into block tridiagonal form. We thus determine a permutation matrix $\mathbf{P}$ such that

$$\mathbf{B} = \mathbf{P}\mathbf{A}\mathbf{A}^T\mathbf{P}^T \tag{4.1}$$

has a block tridiagonal form. To this end, we exploit an implementation of the Cuthill-McKee algorithm (Cuthill and McKee 1969, Duff, Erisman and Reid 1986, George 1971) for ordering symmetric matrices. We then solve the row-wise permuted system of equations

$$\widehat{\mathbf{A}}\mathbf{x} = \widehat{\mathbf{b}} \tag{4.2}$$

with $\widehat{\mathbf{A}} = \mathbf{P}\mathbf{A}$, and $\widehat{\mathbf{b}} = \mathbf{P}\mathbf{b}$, using the block Cimmino algorithm. From the block tridiagonal structure of the matrix $\mathbf{B}$, the block row partition (2.2) of $\widehat{\mathbf{A}}$ is defined with blocks of rows with each block determined by the size of the diagonal blocks in the block tridiagonal structure of $\mathbf{B}$, or by the number of rows in a contiguous subset of these diagonal blocks. We illustrate this in Figure 4.1. The row partitioning we obtain from this still has the properties of the two-block partitioning described in Section 3.

This preprocessing **Strategy I** for general sparse matrices exploits only the sparsity structure in the normal equations matrix $\mathbf{A}\mathbf{A}^T$ and not the numerical values of the entries in this matrix. We should mention and will illustrate it explicitly by the experiments in the following sections that, for very general sparsity patterns, the block tridiagonal structure obtained with the Cuthill-McKee Algorithm has diagonal blocks with very differing sizes leading to an unbalanced partitioning with the block-row projections requiring very different amounts of work. From the discussion in Section 2, we recall also that the main objective when defining the partitioning is to minimize the effects of ill-conditioning across the blocks, which simply means keeping

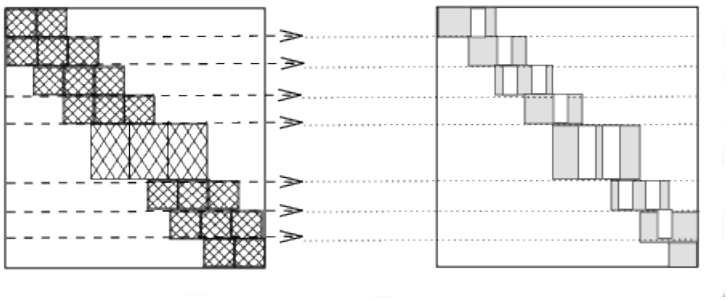


Figure 4.1: Row partitioning of $\widehat{\mathbf{A}}$ (on the right) from block tridiagonal structure of $\mathbf{B}$ (on the left).

the principal angles between every pair of subspaces as open as possible. Strict orthogonality between every second block of rows is only one step in this direction; we may also try to take into account in some way the angles themselves when defining the reorderings and partitionings.

## 4.2 Partitioning Strategy II

The preprocessing **Strategy II** that we now introduce will take into account the numerical values in the matrix $\mathbf{A}\mathbf{A}^T$ as well as the sparsity structure of that matrix in an attempt to define a partitioning strategy with numerical properties close to that of two-block partitioning but with more flexibility for building the blocks and potentially a much better balanced partition.

In this preprocessing strategy the matrix $\mathbf{A}\mathbf{A}^T$ is first normalized through:

$$\begin{aligned} \mathbf{C} &= \mathbf{A}\mathbf{A}^T, \\ \mathbf{D} &= \mathbf{diag}(\mathbf{C}), \\ \mathbf{S} &= \mathbf{D}^{-\frac{1}{2}}\mathbf{C}\mathbf{D}^{-\frac{1}{2}}. \end{aligned}$$

The entries in the normalized matrix $\mathbf{S}$ correspond to the cosine of the principal angle between every pair of rows (in other words, the degree of collinearity between any pair of rows), and we may expect that if such a cosine is relatively small, then the corresponding pair of rows are almost orthogonal and can be considered so. Our next step in this preprocessing strategy is then to keep only the nonzero entries in $\mathbf{S}$ which are above a given threshold $\tau$ in absolute value, viz

$$\mathbf{F} = \mathbf{filter}(|\mathbf{S}|, \tau),$$

and to permute the resulting matrix $\mathbf{F}$ into the block tridiagonal form

$$\widehat{\mathbf{B}} = \mathbf{P}\mathbf{F}\mathbf{P}^T, \tag{4.3}$$

using the Cuthill-McKee algorithm as before. We then solve (4.2) by using the row partitioning for $\widehat{\mathbf{A}}$ defined by the block tridiagonal structure of $\widehat{\mathbf{B}}$.

In practice, we do not want to use the normal equations matrix $\mathbf{A}\mathbf{A}^T$ to solve the subproblems, but we first ensure that the diagonal entries will be one by scaling the original

matrix $\mathbf{A}$ so that the rows have 2-norm equal to one. This can be done by using the HSL routine `MC77` (Ruiz 2001). Of course, since numerical values have been dropped from the normal equations matrix, we cannot expect that the resulting partition will provide two subsets of structurally orthogonal blocks of rows but, if the values dropped are sufficiently small, we may expect that the numerical properties of the resulting iteration matrix will be relatively close to that of the "*strict*" two-block partitioning case. Additionally, since the filtered matrix $\mathbf{F}$ has less entries than the original normal equations matrix $\mathbf{C}$, the resulting block tridiagonal permuted matrix $\widehat{\mathbf{B}}$ will surely have a smaller bandwidth than $\mathbf{B}$ and this may help to define a partitioning on $\mathbf{A}$ with more blocks, better balanced projections, and a higher degree of parallelism. In Section 5, we will experiment and compare these two preprocessing strategies with another strategy that we will now describe.

## 4.3 Hypergraph partitioning

A main aim of the partitioning strategies that we have just described is to decouple the blocks to reduce the number of block Cimmino iterations. We now look at a way to do this more directly.

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is a generalization of the concept of graphs, where $\mathcal{V}$ are the vertices and $\mathcal{N}$ are the hyperedges, also called nets. In a hypergraph, nets can connect more than two vertices whereas, in a graph, edges connect only two of them.

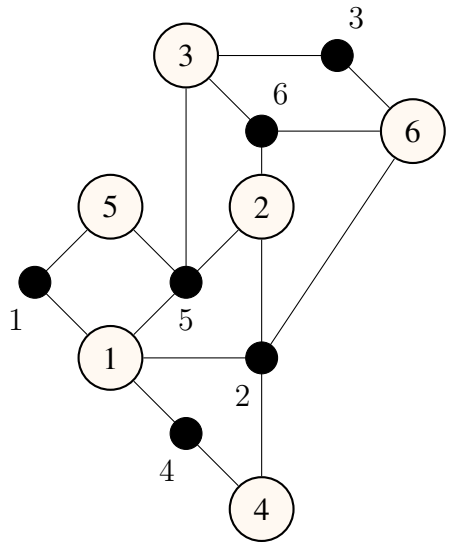



Figure 4.2: A hypergraph representing a sparse matrix. Large circles are the vertices (corresponding to the rows) and small dark circles are the nets (corresponding to the columns).

Hypergraphs can be useful to represent a sparse matrix where the vertices can be associated with the rows and the nets with the columns. We show in Figure 4.2 an example of a hypergraph with 6 vertices and 6 nets representing a $6 \times 6$ sparse matrix shown in Figure 4.3. Notice how we can easily see from the hypergraph all the connections between the different rows in the matrix.

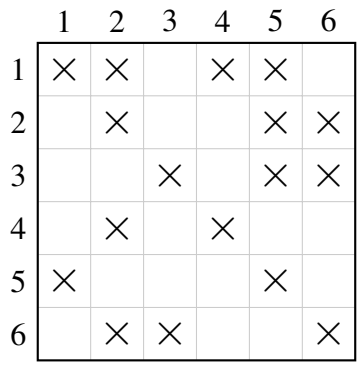



Figure 4.3: The sparse matrix represented by the hypergraph in Figure 4.2.

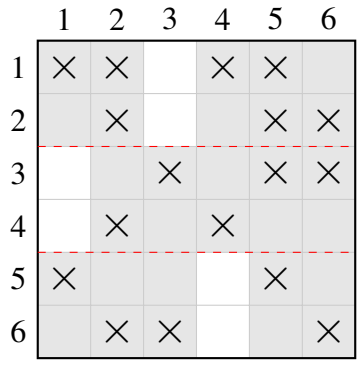



Figure 4.4: A partitioning of the matrix in Figure 4.3 with the columns causing communications shaded in gray.

If we partition the rows of the matrix uniformly into three sets (blocks) as in Figure 4.4, we see that each block has overlapping columns with all the other blocks. This creates all-to-all interactions between the blocks. Columns 2, 5 and 6 are shared by all blocks, whereas columns 1, 3 and 4 are shared between two blocks. A partitioning with such an overlap will be prohibitive for parallel runs because of the communication cost.

A $k$-way hypergraph partitioning creates a partitioning $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_k\}$, where $\mathcal{V}_i$ is a nonempty subset of the vertices $\mathcal{V}$ where $\mathcal{V}_i \cap \mathcal{V}_j = 0, i \neq j$ and $\bigcup_{i=1}^{k} \mathcal{V}_i = \mathcal{V}$, i.e. all the blocks in $\Pi$ are pairwise disjoint and their union is equal to $\mathcal{V}$. The problem of minimizing the connections between the blocks is NP-hard (Lengauer 1990).

A possible partitioning of the previous hypergraph is shown in Figure 4.5 where the three regions represent the three blocks. To easily distinguish the three blocks we duplicate the hyperedges 2, 5 and 6, corresponding to the column overlaps. The interconnections and the resulting permuted matrix are shown in Figure 4.6.

We see that, compared to the partitioning in Figure 4.4, the number of global interconnections reduces from 4 to 2 and the total number of interconnections reduces from 6 to 3.

We use the hypergraph partitioning scheme of Çatalyürek and Aykanat (1999*a*). Their code,

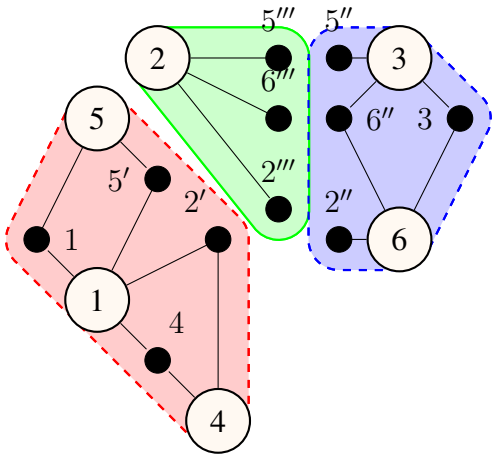



Figure 4.5: A possible partitioning of the hypergraph in Figure 4.2.

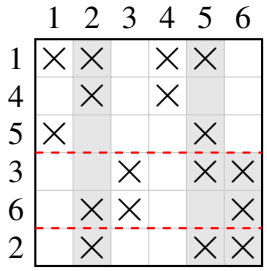



Figure 4.6: The partitioning using a hypergraph partitioner of the matrix from Figure 4.2. The columns in gray show the interconnections.

`PaToH` (Çatalyürek and Aykanat 1999*b*), generates a $k$-way partitioning of a sparse matrix. An input parameter in the code permits us to balance the number of rows in each block. In the following experiments, we will use two levels of balancing. The first is a weak balancing where we allow the blocks to have a large difference in number of rows while reducing greatly the number of interconnections. This helps to group the interconnected rows within the blocks as much as possible. The second level of balancing is strong balancing where the target is to have an almost equal number of rows per block. Although we note that such a balancing does not directly balance the amount of work for each projection, it is a good heuristic that is readily available within the `PaToH` code.

# 5 Partitioning Experiments

In this section, we perform some experiments with the block Cimmino solver using block CG acceleration (Arioli et al. 1995*a*, Arioli, Duff, Ruiz and Sadkane 1995*b*) and focus on the effects of the preprocessing strategies on the performance. We have therefore chosen a fixed block size for the block CG acceleration and stop the iterations on the basis of a normwise backward

error (Arioli, Duff and Ruiz 1992*a*)

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

of less than $10^{-12}$. A small value for $\omega_k$ means that the algorithm is normwise backward stable (Oettli and Prager 1964) in the sense that the solution $\mathbf{x}^{(k)}$ is the exact solution of a perturbed problem where the max norm of the error matrix is less than or equal to $\omega_k$.

In Section 5.1, block Cimmino is used to solve the SHERMAN3 linear system from the Harwell-Boeing matrix collection (Duff, Grimes and Lewis 1997). We choose this test problem because the matrix is very typical of a wide range of matrices arising from the solution of discretized partial differential equations but is difficult to solve because of the inclusion of temperature in the model. Section 5.2 contains the results from experiments of runs of block Cimmino on the bayer01 problem from the sparse matrix collection at the University of Florida (Davis 2008). This problem has been chosen to be representative of larger problems with structures quite different from those from PDE discretizations. We then perform some experiments with a parallel implementation in Section 5.3.

## 5.1 Solving the SHERMAN3 problem

The matrix SHERMAN3 is an unsymmetric matrix of order 5005. This matrix comes from the discretization of partial differential equations extracted from a three dimensional oil reservoir simulation model on a $35 \times 11 \times 13$ grid using a seven-point finite difference approximation.

The pattern of the matrix SHERMAN3 is shown in Figure 5.1. In the first experiment, a naive block-row partition of the linear system is used, with eight equal-sized blocks of rows. This results in blocks of 625 rows except for the last one of 630.

Figure 5.2 shows the spectrum of $\mathbf{H}$ when using the naive partitioning shown in Figure 5.1. Figure 5.2 shows a large cluster of eigenvalues around 1 but a few trailing eigenvalues associated with bad conditioning (the smallest eigenvalue is of the order $10^{-8}$).

The block Cimmino method is numerically independent of any column permutations so, after the row-partitioning of the system, we also perform some column permutations to group together columns belonging to the same subsets of row-partitions in order to facilitate the communication phase in the algorithm when merging the results from the different projections.

In the second round of experiments, the preprocessing **Strategy I** from Section 4 is used. From the block tridiagonal structure of the permuted normal equations matrix of SHERMAN3, the matrix is partitioned into blocks of rows. Figure 5.3.b shows the pattern of matrix SHERMAN3 after row permutation following the preprocessing **Strategy I**, using the Cuthill-McKee algorithm, to permute the normal equations matrix into block tridiagonal form, as shown in Figure 5.3.a. The resulting partitioning shown in Figure 5.3.b is a two-block partitioning, and columns belonging to the same subsets of blocks have been grouped together as mentioned before. The block-row partitions have been defined using the block tridiagonal structure in the normal equations matrix. As in the case of the naive partitioning, we again aim at obtaining 8 blocks of about 625 rows each.

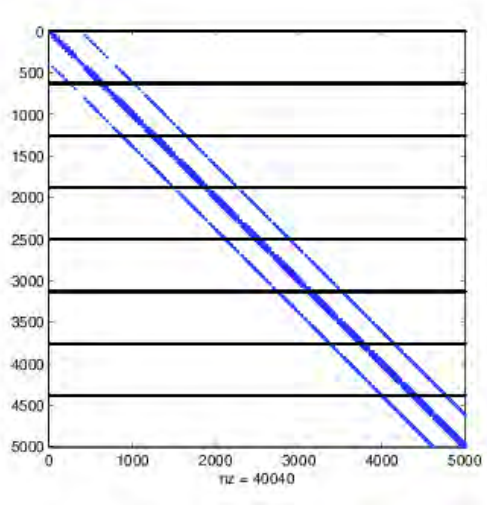



Figure 5.1: Sparsity pattern of the SHERMAN3 matrix. The matrix has been partitioned into 8 equal-sized blocks of rows using the original ordering. We call this the naive partitioning.

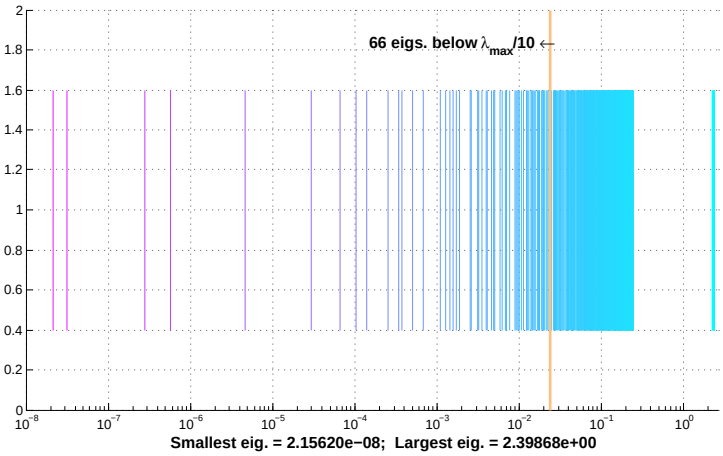
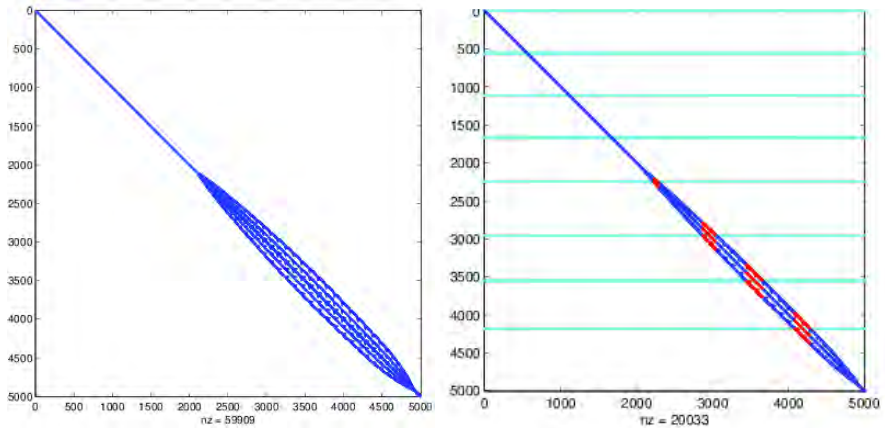



Figure 5.2: Eigenvalue spectrum of $\mathbf{H}$ for the SHERMAN3 problem with a naive partitioning.




(a) (b)

Figure 5.3: (a) Permuted normal equations from SHERMAN3 using the Cuthill-McKee algorithm. (b) Matrix SHERMAN3 after row permutations following the preprocessing **Strategy I** and with additional column grouping.

The sparsity pattern of the permuted SHERMAN3 matrix after completion of preprocessing **Strategy II** is shown in Figure 5.5.b. In this case we dropped the entries of the normalized normal equations matrix that are below 0.2, although other values can be used as we discuss later. We plot in Figure 5.4 all the entries of the normal equations matrix, where we see that dropping at 0.2 will remove less than half of the entries. The matrix with the entries dropped was permuted using the ordering from the Cuthill-McKee algorithm. The associated block-row partition, indicated in Table 5.1, was obtained from the block tridiagonal structure of the matrix $\widehat{\mathbf{B}}$ in (4.3), as described in Section 4, with the aim of again obtaining 8 blocks of about 625 rows each.

In the case of the hypergraph partitioning, we use two different imbalance parameters, a weak balancing which tolerates blocks up to 8 times larger than other blocks, and a strong balancing which tolerates up to 50% imbalance.

As expected, the matrix $\widehat{\mathbf{B}}$ in Figure 5.5.a has a smaller bandwidth than the matrix in Figure 5.3.a. Thus the preprocessing **Strategy II** offers more degrees of freedom to define the row partitions than **Strategy I**, since they are of smaller size. Therefore, we can define more partitions while maintaining a good balance between the number of rows in each block.

The spectrum of $\mathbf{H}$ for strategies **I** and **II** is shown in Figures 5.6 and 5.7, respectively. Notice that compared to the spectrum of $\mathbf{H}$ arising from the naive partitioning the spectrum presents a better clustering of the eigenvalues. In **Strategy I** the largest eigenvalue is 2, consistent with a two-block partitioning. However, using **Strategy II**, the largest eigenvalue is only slightly larger than 2, and thus we call **Strategy II** a **near two-block partitioning**. The convergence results in Table 5.2 illustrate the effect of the better clustering of eigenvalues obtained using **Strategy II**.

The second column of Table 5.2 shows results from the runs of the block Cimmino method using different partitioning strategies. These results should be multiplied by the block size (8 in our case) to obtain the number of matrix-vector operations, and we show them in the last column so we can easily compare the amount of work. We note that the naive partitioning requires the

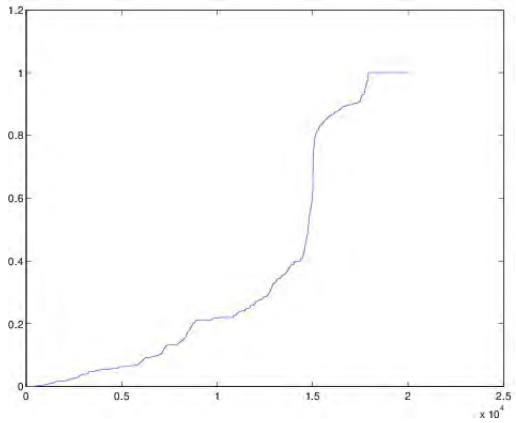



Figure 5.4: Normalized nonzero entries in the normal equations matrix from SHERMAN3. On the x-axis, the entries are displayed in increasing order of their absolute numerical value (given on the y-axis).

| | **Number of rows in each part** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Original Matrix A** | 625 | 625 | 625 | 625 | 625 | 625 | 625 | 630 |
| **Strategy I** | 625 | 625 | 625 | 635 | 629 | 628 | 784 | 454 |
| **Strategy II (drop 0.1)** | 625 | 625 | 625 | 744 | 707 | 669 | 628 | 382 |
| **Strategy II (drop 0.2)** | 625 | 625 | 625 | 627 | 641 | 671 | 630 | 561 |
| **Strategy II (drop 0.4)** | 637 | 637 | 637 | 637 | 637 | 637 | 637 | 546 |
| **PaToH (weak balancing)** | 1019 | 396 | 1015 | 400 | 544 | 544 | 544 | 543 |
| **PaToH (strong balancing)** | 626 | 626 | 626 | 625 | 626 | 625 | 626 | 625 |

Table 5.1: Description of the 8 block-row partitions obtained for matrix SHERMAN3.

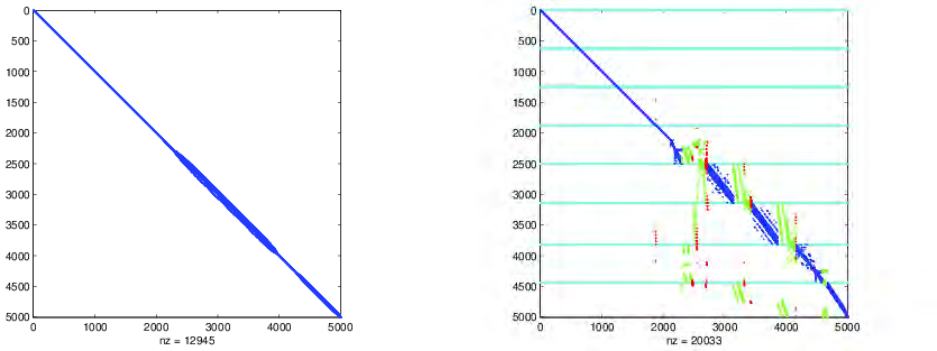
highest number of iterations for convergence since neither the structure of the matrix nor the value of the matrix entries were taken into account.

However, when we take into account both the value of the entries and the structure of the matrix as in **Strategy II**, we are able to reduce the iteration count to 68 when dropping at 0.2 compared to 102 when using **Strategy I**. We examine further the effect of dropping on the convergence by showing counts for dropping at 0.1 and 0.4. We see that, as we increase the dropping threshold from 0.1 to 0.2, we get a finer partition with the higher values in the blocks. However, as we increase the dropping threshold more (to 0.4) we start to lose these connections resulting in an increase in iteration count.

Using PaToH, we can reduce the iteration count to 52 when we use weak balancing and tolerate some large blocks in order to reduce the interconnections between them. We notice that, when using a strong balancing, the iteration count rises to 74.

| | Nb. of iterations | M-V operations |
|---|---|---|
| **Naive partitioning** | 190 | 1520 |
| **Strategy I** | 102 | 816 |
| **Strategy II (drop 0.1)** | 73 | 584 |
| **Strategy II (drop 0.2)** | 68 | 544 |
| **Strategy II (drop 0.4)** | 89 | 712 |
| **PaToH (weak balancing)** | 52 | 416 |
| **PaToH (strong balancing)** | 74 | 592 |

Table 5.2: Convergence of SHERMAN3 in the three different sets of experiments.




(a) (b)

Figure 5.5: (a) Sparsity pattern of matrix $\hat{\mathbf{B}}$ in (4.3), obtained after removing nonzero entries less than 0.2 from the normalized SHERMAN3 normal equations matrix, and using the Cuthill-McKee algorithm to permute the resulting matrix into block tridiagonal form. (b) Matrix SHERMAN3 after row permutations following the preprocessing strategy II and with additional column grouping.

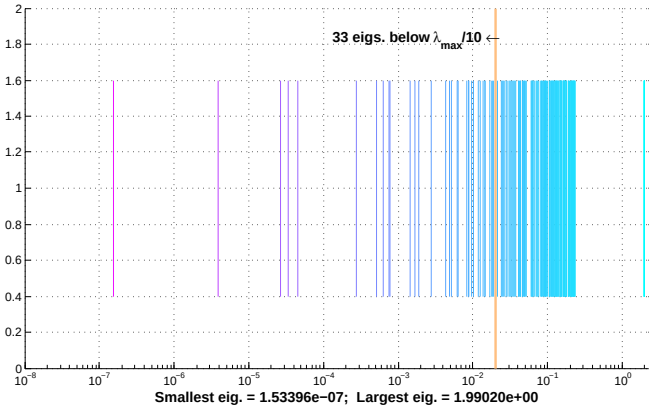



Figure 5.6: Eigenvalue spectrum of $\mathbf{H}$ for the permuted SHERMAN3 problem arising from using **Strategy I.**

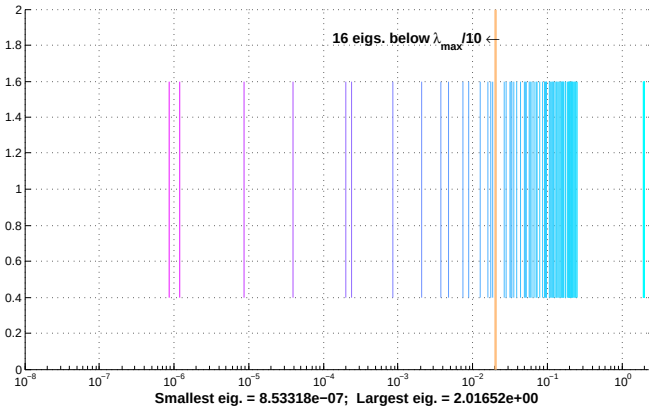



Figure 5.7: Eigenvalue spectrum of **H** for the permuted SHERMAN3 problem arising from using **Strategy II** with a drop at 0.2.

## 5.2 Solving the `bayer01` problem

We now look at these strategies on a larger problem, `bayer01`. This matrix was obtained from Bayer AG by Friedrich Grund and is available from the sparse matrix collection at the University of Florida (Davis 2008). It is of order 57735 and has 277774 nonzero entries. We show the pattern of the matrix in Figure 5.8 where we have superimposed the naive partition with 16 blocks.

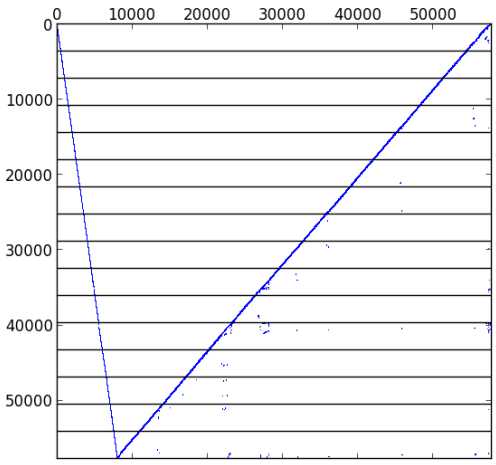



Figure 5.8: Nonzero pattern of the matrix `bayer01`showing the naive partitioning.

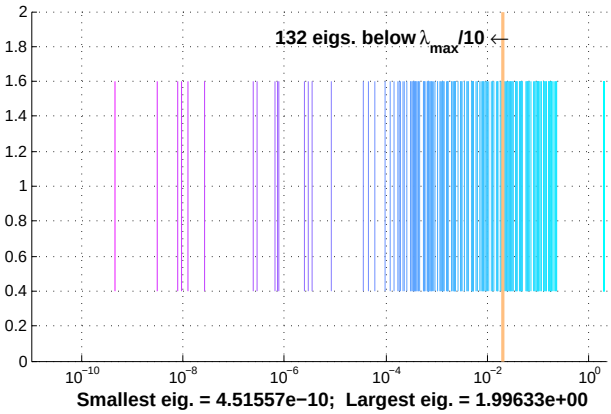



Figure 5.9: Spectrum of block Cimmino iteration matrix, $\mathbf{H}$, for `bayer01` with 16 uniform partitions.

If we run the block Cimmino algorithm on the matrix partitioned as in Figure 5.8 then the resulting spectrum of the iteration matrix $\mathbf{H}$ is shown in Figure 5.9. We see that, while there is a good clustering of the eigenvalues around the value 1, just as we noticed with the matrix `SHERMAN3`, the matrix is still quite badly conditioned. Indeed, many small eigenvalues are present and these eigenvalues will increase the iteration count when using the conjugate gradient acceleration.

If, however, we first reorder the matrix using `PaToH` and then partition it, we get the spectrum for the iteration matrix shown in Figure 5.10 where we note that the intermediate eigenvalues have been shifted towards 1 thus improving the clustering of eigenvalues in the iteration matrix. The reduction in the number of small eigenvalues is expected to improve the convergence.

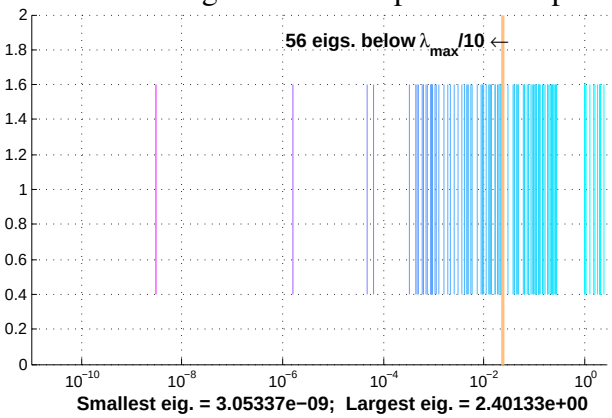



Figure 5.10: Spectrum of block Cimmino iteration matrix for `bayer01` with 16 blocks obtained using the hypergraph partitioner `PaToH`.

| | Smallest block. | Largest block. | Ratio | Time |
|---|---|---|---|---|
| **Original Matrix A** | 3608 | 3615 | 1.00 | n/a |
| **Strategy I** | 1673 | 4342 | 2.60 | 0.25 |
| **Strategy II (drop 0.05)** | 1448 | 4017 | 2.77 | 0.42 |
| **Strategy II (drop 0.1)** | 3318 | 3660 | 1.10 | 0.55 |
| **Strategy II (drop 0.2)** | 3522 | 3634 | 1.03 | 0.94 |
| **`PaToH` (weak balancing)** | 952 | 7597 | 7.98 | 0.40 |
| **`PaToH` (strong balancing)** | 3608 | 3609 | 1.00 | 0.35 |

Table 5.3: Information on the 16 block-row partitions obtained for matrix `bayer01`.

Table 5.3 summarizes the partitioning information after applying the different strategies. As there are 16 blocks, we show only the smallest, the largest, the ratio between the largest and the smallest block, and the time it takes to obtain the partitioning sequentially on a Intel Xeon E5-2687w at 3.1GHz. We notice that when using **Strategy I** we are constrained by the size of the level sets so that balancing the partition is quite difficult, and the largest block is more than 2.5 times the size of the smallest block. This is because it is not possible to permute the normal equation matrix to one with a small bandwidth. We overcome this problem with **Strategy II** and obtain better load balancing. This improves as we increase the dropping threshold parameter. The increase in execution time from **Strategy I** to **Strategy II** that continues when increasing the dropping threshold is related to a higher number of level sets in the Cuthill-McKee process.

In the case of PaToH, weak balancing gives the greatest freedom for partitioning, and our largest block is nearly 8 times larger than the smallest. If we constrain this freedom by strengthening the balancing, we obtain a partition with almost equal-sized blocks. We notice that the hypergraph partitioning on bayer01 takes slightly longer than using **Strategy I**, and that using strong balancing is faster than using the weak balancing.

| | **Nb. of iterations** | **M-V operations** |
|---|---|---|
| **Naive partitioning** | 256 | 2048 |
| **Strategy I** | 459 | 3672 |
| **Strategy II (drop 0.05)** | 270 | 2160 |
| **Strategy II (drop 0.1)** | 204 | 1632 |
| **Strategy II (drop 0.2)** | 343 | 2744 |
| **PaToH (weak balancing)** | 52 | 416 |
| **PaToH (strong balancing)** | 105 | 840 |

Table 5.4: Convergence of bayer01 with the different partitioning strategies.

We show in Table 5.4 the results for the matrix bayer01. We notice a similar behaviour as with the SHERMAN3 matrix where the preprocessing **Strategy II** improves the iteration count when dropping until a certain threshold. However, the preprocessing **Strategy I** is worse than just using the naive partitioning. This behaviour was seen on several problems which makes this strategy unreliable. Finally, using PaToH both with weak and strong balancing gives the best results.

## 5.3 Parallel Experiments

In this section we solve some standard test problems with a parallel version of our code. This version uses a distributed block CG acceleration, illustrated in Figure 5.11, where the blocks are distributed over multiple processes called masters. All these processes solve the problem in a distributed manner and communicate only for dot product and matrix-vector computations. The matrix-vector part involves the projections on the subspaces that are solved using the direct solver MUMPS (Amestoy et al. 2001). As MUMPS is itself a parallel code, it can use supplementary processes, called workers in our illustration.

We list, in Table 5.5, the matrices from the linear systems that we will solve in parallel. We have seen that PaToH gives a better partitioning compared to the other strategies. Thus we only do runs with PaToH in this section. For the first three problems we use a block size of 1, which means a classical CG acceleration, while for the fourth problem we use a block size of 4 as it is needed for better convergence. We will compare the effect of balancing when using PaToH,

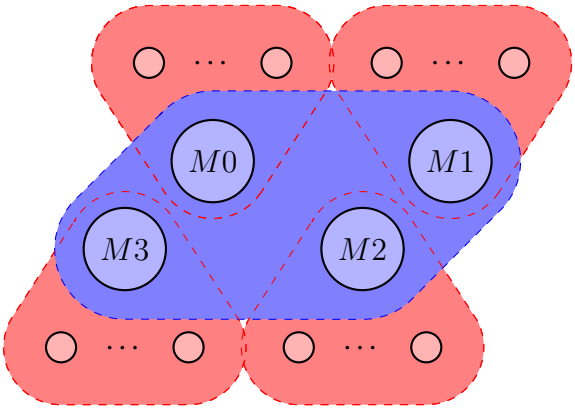



Figure 5.11: Representation of how the processes are organized. The $Mk$ circles are the masters, the other circles are the workers.

and see whether the lower iteration count of weak balancing compensates for its more poorly balanced partition.

| **Problem** | **Size** | **Nonzeros** | **Application** | **Parts** |
|---|---|---|---|---|
| $N_1$: torso3 | 259,156 | 4,429,042 | 3D model of torso | 16 |
| $N_2$: CoupCons3D | 416,800 | 17,277,420 | structural problem | 32 |
| $N_3$: cage13 | 445,315 | 7,479,343 | DNA electrophoresis | 256 |
| $N_4$: Hamrle3 | 1,447,360 | 5,514,242 | Circuit Simulation | 64 |

Table 5.5: Matrices from the University of Florida Sparse Matrix Collection.

In Table 5.4 in the previous section we saw that using weak balancing can result in much faster convergence than using strong balancing. However, as we saw in Table 5.3 there is a dramatic difference in the ratio of sizes of blocks in the partitions for these two strategies. When running in parallel this latter effect can be very important and may well more than offset any reduction in the number of iterations. For example, in our runs on the `cage13` problem ($N_3$ in Table 5.5), on a 32 core machine, we varied the balancing parameter and partitioned the matrix into 256 blocks. With weak balancing, the factorization took about 19 seconds and the block CG converged in 16 iterations in about 5 seconds. Using strong balancing, the factorization took about 10 seconds and the block CG converged in 17 iterations in about 4 seconds. Clearly, there will always be a trade-off between getting a well balanced partition and reducing the iteration count that will be problem dependent.

Our parallel experiments are on a machine with nodes having two quad-core Nehalem sockets and 32GB of memory for both processors. In all cases we will use the 8 cores of each node and increase the number of nodes. This way we test for 1, 8, 16, 32, 64 and 128 MPI processes. The matrices are partitioned using `PaToH` with weak balancing.

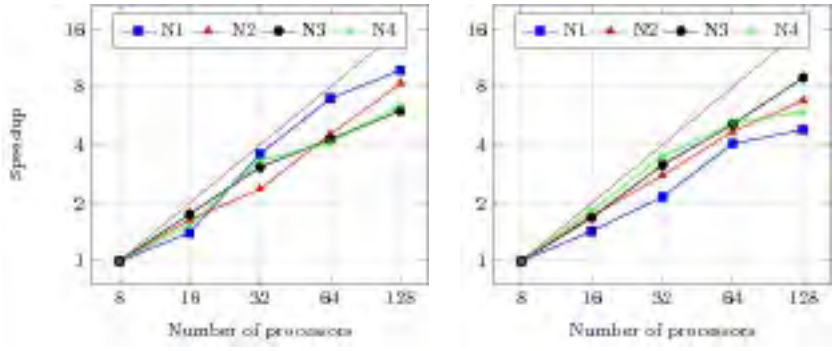
We show in Tables 5.6 and 5.7 the time spent in seconds to factorize the augmented systems

| | Factorization | | | | | |
|---|---|---|---|---|---|---|
| MPI procs | 1 | 8 | 16 | 32 | 64 | 128 |
| $N_1$ | 46.89 | 12.11 | 8.68 | 3.37 | 1.74 | 1.24 |
| $N_2$ | 83.27 | 14.41 | 8.78 | 6.11 | 3.18 | 1.73 |
| $N_3$ | 77.09 | 28.15 | 16.04 | 9.16 | 6.58 | 4.68 |
| $N_4$ | 34.8 | 6.67 | 4.33 | 2.01 | 1.62 | 1.05 |

Table 5.6: The elapsed time in seconds to factorize the augmented systems.

| | block CG | | | | | |
|---|---|---|---|---|---|---|
| MPI procs | 1 | 8 | 16 | 32 | 64 | 128 |
| $N_1$ | 21.57 | 6.34 | 4.43 | 2.94 | 1.56 | 1.32 |
| $N_2$ | 272.21 | 63.49 | 36.89 | 22.88 | 13.50 | 9.37 |
| $N_3$ | 41.67 | 13.64 | 8.14 | 4.33 | 2.67 | 1.53 |
| $N_4$ | 3905.0 | 773.0 | 422.4 | 221.3 | 149.52 | 130.13 |

Table 5.7: The elapsed time in seconds for the block CG acceleration to converge.




a. Factorization speedup b. block CG speedup

Figure 5.12: Parallel results of the factorization of augmented systems and block CG acceleration speedups.

(2.10) and the time taken for the block CG to converge. In these runs we have performed a strong scaling analysis showing the speedups when a single system is solved on an increasing number of processes. As we see in both plots in Figure 5.12, we obtain a good speedup on all four problems. However, the speedup is not uniform. It is very good for each problem until the number of processes is equal to the number of blocks when it becomes somewhat poorer. The reason for this lies in the way our algorithm handles the partitions, when we have less processes than blocks, each process will handle more than one partition. Increasing the number of processes will decrease the amount of work done by each process which results in good speedups. However, when we increase the number of processes above the number of blocks, the

masters will handle only a single block and hence we no longer have speedups from the first level of parallelism. Other processes (the workers) will be used by the direct solver MUMPS yielding a second level of parallelism. Further discussion on this aspect of our work is beyond the scope of this paper.

# 6 Conclusions

We have shown that preprocessing the original system of equations to obtain a different partitioning of the system can have a significant effect on the convergence of the block Cimmino algorithm resulting in a more robust and efficient implementation. Indeed, a good preprocessing strategy can minimize the ill-conditioning between the different blocks. Our **Strategy I** did this by enforcing a two-block partitioning but it was not very good at obtaining a balanced blocking or at reducing the number of iterations. We thus tried **Strategy II** which did much better with respect to a naive partitioning but at the cost of losing, to a controlled extent, the properties of the two-block partitioning. Although this did better than **Strategy I**, it was still not a robust approach and did not always provide an improvement over a naive partitioning of the original system. It also required the selection of a dropping threshold parameter that was dependent on the problem being solved and was hard to choose in advance. We thus examined a strategy that uses a hypergraph model to partition the original system with the aim of directly reducing the connection between blocks. We used the PaToH software to effect this partition and found that it compared very favourably to our strategies based on the normal equations. It was also more robust in that it always produced partitions better than a naive partition of the initial matrix. It was also possible to use PaToH to obtain a well balanced partition (that is with subblocks of roughly equal size). For the experiments using a parallel version of our code, we thus chose only to examine the performance of the partitioning using PaToH.

Our implementation of the block Cimmino method uses block conjugate gradients to accelerate its convergence. In order to exploit parallelism, we developed an MPI implementation of our block Cimmino with block CG acceleration. We used strong and weak partitioning strategies to define the subblock sizes (number and choice of rows in $\mathbf{A}$ in each subblock). We observed that forcing a more balanced workload distribution produced better speedups, even if the overall iteration count was higher.

Future multi-level hybrid parallel systems may be able to exploit more non-uniform block sizes and dynamically adjust the number of processes to the size of the block in a manner that reduces the iteration count, minimizes load imbalance, and produces better resource utilization. We have shown that we can exploit both the parallelism from the partitioning as well as that from the direct solver MUMPS so that the block Cimmino computational scheme should be a good approach for solving systems on heterogeneous multi-level parallel systems.

# Acknowledgments


- This work was partially granted by the ANR-BARESAFE project, ANR-11-MONU-004, Programme Modèles Numriques 2011, supported by the French National Agency for Research.
- This work was granted access to the HPC resources of CALMIP under the allocation 2013-P0989. We thank the four referees for their detailed comments on the first version of the manuscript.
- The research of I. S. Duff was supported in part by the EPSRC Grant EP/I013067/1.


# 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*), A parallel scheduler for block iterative solvers in heterogeneous computing environments, *in* D. H. B. et al., ed., 'Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing', SIAM, Philadelphia, pp. 460–465.

Arioli, M., Duff, I. S. and Ruiz, D. (1992*a*), 'Stopping criteria for iterative solvers', *SIAM J. Matrix Anal. and Appplics* **13**, 138–144. Special issue in honour of Gene's 60th birthday.

Arioli, M., Duff, I. S., Noailles, J. and Ruiz, D. (1992*b*), '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.

Björck, A. and Golub, G. H. (1973), 'Numerical methods for computing angles between linear subspaces', *Math. Comp.* **27**, 579–594.

Bramley, R. (1989), Row projection methods for linear systems, PhD Thesis 881, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL.

Bramley, R. and Sameh, A. (1992), 'Row projection methods for large nonsymmetric linear systems', *SIAM J. Scientific and Statistical Computing* **13**, 168–193.

Çatalyürek, Ü. V. and Aykanat, C. (1999*a*), 'Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication', *IEEE Transactions on Parallel and Distributed Systems* **10**(7), 673–693.

Çatalyürek, Ü. V. and Aykanat, C. (1999*b*), *PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0*, Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey. PaToH is available at `http://bmi.osu.edu/~umit/software.htm`.

Cuthill, E. and McKee, J. (1969), Reducing the bandwidth of sparse symmetric matrices, *in* 'Proceedings 24th National Conference of the Association for Computing Machinery, Brandon Press, New Jersey', Brandon Press, New Jersey, pp. 157–172.

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

Duff, I. S., Erisman, A. M. and Reid, J. K. (1986), *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford, England.

Duff, I. S., Grimes, R. G. and Lewis, J. G. (1997), The Rutherford-Boeing Sparse Matrix Collection, Technical Report RAL-TR-97-031, Rutherford Appleton Laboratory, Oxfordshire, England. Also Technical Report ISSTECH-97-017 from Boeing Information & Support Services, Seattle and Report TR/PA/97/36 from CERFACS, Toulouse.

Duff, I. S., Guivarch, R., Ruiz, D. and Zenadi, M. (2013), The augmented block cimmino distributed method, Technical Report TR/PA/13/11, CERFACS, Toulouse, France.

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

George, A. (1971), Computer implementation of the finite-element method, PhD thesis, Department of Computer Science, Stanford University, Stanford, California. Report STAN CS-71-208.

Golub, G. H. and Kahan, W. (1965), 'Calculating the singular values and pseudo-inverse of a matrix', *SIAM J. Numer. Anal.* **2**, 205–225.

Golub, G. H. and Van Loan, C. F. (2013), *Matrix Computations. Fourth Edition*, The Johns Hopkins University Press, Baltimore and London.

Kamath, C. and Sameh, A. (1988), 'A projection method for solving nonsymmetric linear systems on multiprocessors', *Parallel Computing* **9**, 291–312.

Lengauer, T. (1990), *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley–Teubner, Chichester, U.K.

Oettli, W. and Prager, W. (1964), 'Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides', *Numer. Math.* **6**, 405–409.

Ruiz, D. (2001), A scaling algorithm to equilibrate both row and column norms in matrices, Technical Report RAL-TR-2001-034, Rutherford Appleton Laboratory.

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.