Science & Technology Facilities Council

Preprint

RAL-P-2013-010

Partitioning strategies for the block Cimmino algorithm

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

July 2013

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 $Ax = b$. We study strategies that transform the matrix 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 consider obtaining a partitioning by using hypergraph partitioning that works directly on the matrix A to reduce the interconnections 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 available through the URL `http://www.stfc.ac.uk/CSE/36276.aspx`. It has also appeared as CERFACS report TR/PA/13/42 and INPT(ENSEEIHT)-IRIT report RT-APO-13-6.

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

[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]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

July 3, 2013

Contents

1 Introduction

The benefit of the block Cimmino algorithm that we describe in Section 2 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. It is therefore possible to extend techniques suitable for these smaller systems, for example sparse direct methods, to these 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 this convergence 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 reduce the iteration count of block Cimmino. We then discuss two algorithms for obtaining such a form in Section 4. In Section 5, we describe a different approach to obtaining a partitioning with similar properties.

We then describe experiments contrasting these approaches and show their efficacy on real problems in Section 6. We present our conclusions in Section 7.

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:

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

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.

A general study of the convergence of this method and of other related block-row and block-columns methods can be found in Elfving (1980). A block SSOR algorithm of this type introduced in Kamath and Sameh (1988), is accelerated using conjugate gradients. They study its robustness compared to some preconditioned conjugate gradient methods of the Yale package PCGPACK. There has been much prior work on block SSOR as well as 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 is augmented so that the partitions are orthogonal. The

solution of 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 linear combinations of rows that are almost equal to zero and, as mentioned by Bramley and Sameh (1992), these linear combinations may occur inside the blocks or across the 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 $\mathbf{E}_{RJ}$ formed as the sum of the previously mentioned projections, and given by

$$\mathbf{E}_{RJ} = \sum_{i=1}^{p} \mathbf{A}_i{}^T \left(\mathbf{A}_i \mathbf{A}_i{}^T\right)^{-1} \mathbf{A}_i \ , \tag{2.3}$$

assuming, for simplicity, that 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. Consider that the matrix is 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:

$$\begin{aligned}
\mathbf{E}_{RJ} &= \sum_{i=1}^{p} \mathbf{A}_i{}^T \left(\mathbf{A}_i \mathbf{A}_i{}^T\right)^{-1} \mathbf{A}_i \\
&= \sum_{i=1}^{p} \mathbf{Q}_i \mathbf{Q}_i{}^T \\
&= (\mathbf{Q}_1 \cdots \mathbf{Q}_p)(\mathbf{Q}_1 \cdots \mathbf{Q}_p)^T
\end{aligned} \tag{2.4}$$

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

Thus, the spectrum of the matrix $\mathbf{E}_{RJ}$ is the same as that of the matrix

$$\begin{pmatrix} \mathbf{I}_{m_1\times m_1} & {\mathbf{Q}_1}^T\mathbf{Q}_2 & \cdots & \cdots & {\mathbf{Q}_1}^T\mathbf{Q}_p \\ {\mathbf{Q}_2}^T\mathbf{Q}_1 & \mathbf{I}_{m_2\times m_2} & {\mathbf{Q}_2}^T\mathbf{Q}_3 & \cdots & {\mathbf{Q}_2}^T\mathbf{Q}_p \\ \vdots & & \ddots & & \vdots \\ {\mathbf{Q}_p}^T\mathbf{Q}_1 & & \cdots & & \mathbf{I}_{m_p\times m_p} \end{pmatrix} \tag{2.5}$$

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). These principal angles (Golub and Van Loan 2013, pages 584-585) are also defined successively by

$$\begin{aligned} \cos(\Psi_k) &= \max_{\mathbf{u}\in\mathcal{R}({\mathbf{A}_i}^T)} \max_{\mathbf{v}\in\mathcal{R}({\mathbf{A}_j}^T)} \frac{\mathbf{u}^T\mathbf{v}}{\|\mathbf{u}\|\ \|\mathbf{v}\|} \\ &= \frac{\mathbf{u}_k^T\mathbf{v}_k}{\|\mathbf{u}_k\|\ \|\mathbf{v}_k\|} \end{aligned} \tag{2.6}$$

subject to

$$\begin{aligned} \mathbf{u}^T\mathbf{u}_p &= 0 \qquad p = 1,\ldots,k-1 \\ \mathbf{v}^T\mathbf{v}_p &= 0 \qquad p = 1,\ldots,k-1, \end{aligned}$$

k varying from 1 to $m_{ij} = \min(\dim(\mathcal{R}({\mathbf{A}_i}^T)), \dim(\mathcal{R}({\mathbf{A}_j}^T)))$. The vectors $\left\{\mathbf{u}_1,\ldots,\mathbf{u}_{m_{ij}}\right\}$ and $\left\{\mathbf{v}_1,\ldots,\mathbf{v}_{m_{ij}}\right\}$ are called the principal vectors between the subspaces $\mathcal{R}({\mathbf{A}_i}^T)$ and $\mathcal{R}({\mathbf{A}_j}^T)$.

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 $\mathbf{E}_{RJ}$ is to the identity matrix, and thus the faster the convergence of the conjugate gradient acceleration should be.

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 $\mathbf{E}_{RJ}$. We will therefore focus in the following on particular partitionings for which there exists a strong relationship between these principal angles and the spectrum of the iteration matrix.

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. The partition of $\mathbf{A}$ is thus of the form

$$[\mathbf{A}] = \begin{bmatrix} \mathbf{B}_1 \\ \\ \\ \mathbf{B}_2 \end{bmatrix}, \tag{3.7}$$

where $\mathbf{B}_1 = \left\{ \bigcup_i \mathbf{A}_i \ / \ i \text{ odd} \right\}$ and $\mathbf{B}_2 = \left\{ \bigcup_i \mathbf{A}_i \ / \ i \text{ even} \right\}$. The partitioning in equation (3.7) is called a two-block partitioning. We denote by m_1 and m_2 the number of rows in $\mathbf{B}_1$ and $\mathbf{B}_2$ respectively. We call the block of smaller size, say $\mathbf{B}_2$ in the above example, the interface block.

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 $\mathbf{E}_{RJ}$ in (2.3) can be considered as $\mathbf{P}_1 + \mathbf{P}_2$, where $\mathbf{P}_1 = \mathbf{P}_{\mathcal{R}(\mathbf{B}_1{}^T)}$ 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 $\mathbf{E}_{RJ}$ is

$$\begin{aligned} \lambda_k &= 1 + \cos\Psi_k & k &= 1, \ldots, m_2 \\ \lambda_k &= 1 - \cos\Psi_{k-m_2} & k &= m_2+1, \ldots, 2m_2 \\ \lambda_k &= 1 & k &= 2m_2+1, \ldots, n \end{aligned}$$

where $\{\Psi_k\}_1^{m_2}$ 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 processors 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 above expression for the spectrum tells us that the block Cimmino algorithm with conjugate gradient acceleration in general works better for small interface block sizes (e.g. small m_2) and, in exact arithmetic, takes not more than $2m_2$ steps for convergence.

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 with their block SSOR algorithm accelerated with conjugate gradients (Bramley 1989, Bramley and Sameh 1992). In Arioli et al. (1992*b*), some particular scalings used in conjunction with two-block partitioning strategies are also 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 some ellipsoidal norms. Finally, the potential for parallelism in distributed memory environments of the block Cimmino method accelerated with the block conjugate gradient algorithm is investigated and discussed in detail by Drummond (1995) (Arioli, Drummond, Duff and Ruiz 1994, Arioli, Drummond, Duff and Ruiz 1995*a*).

4 Preprocessing Strategies

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. The idea is to permute first 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

$$\mathbf{B} = \mathbf{P}\mathbf{A}\mathbf{A}^T\mathbf{P}^T \tag{4.8}$$

has a block tridiagonal form. To this end, we exploit an implementation of the Cuthill-McKee Algorithm (see for instance 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

$$\hat{\mathbf{A}}\mathbf{x} = \hat{\mathbf{b}} \tag{4.9}$$

with $\hat{\mathbf{A}} = \mathbf{P}\mathbf{A}$, and $\hat{\mathbf{b}} = \mathbf{P}\mathbf{b}$, using the block Cimmino algorithm. From the block tridiagonal structure of the matrix $\mathbf{B}$, the block row partition (2.2) of $\hat{\mathbf{A}}$ 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. 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 values contained 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 a partitioning with a poor degree of parallelism because of unbalanced tasks in the block-row projections. 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 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.

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

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 tolerance value τ in absolute value, viz

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

and to permute the resulting matrix $\mathbf{F}$ into block tridiagonal form using the Cuthill-McKee algorithm as before

$$\widehat{\mathbf{B}} = \mathbf{P}\mathbf{F}\mathbf{P}^T \tag{4.10}$$

and solve (4.9) by using the row partitioning for $\hat{\mathbf{A}}$ defined by the block tridiagonal structure of $\hat{\mathbf{B}}$.

In practice, we do not want to form $\mathbf{A}\mathbf{A}^T$ 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 accomplished 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 $\hat{\mathbf{B}}$ 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 the following section, we will experiment and compare these two preprocessing strategies with another strategy that we will now describe.

5 Using a 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.

We use the hypergraph partitioning scheme of Çatalyürek and Aykanat (1999*a*). Their code, `PaToH` (Çatalyürek and Aykanat 1999*b*), finds a row permutation that will make the partitions less interconnected. `PaToH` provides permutations so that the reordered matrix is in bordered block diagonal form. 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 so can easily compare the effect of using PaToH with the partitionings that we have just described.

Another benefit of using PaToH is that the columns causing the interconnection between blocks are clearly identified and we use this in our approach for augmenting the system to explicitly generate a blocking where the blocks are orthogonal to each other (Duff et al. 2013).

6 Partitioning Experiments

In this section, we perform some experiments with the block Cimmino solver using Block-CG acceleration (Arioli, Duff, Ruiz and Sadkane 1995*b*, Arioli et al. 1995*a*) 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 ω_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 ω_k.

In Section 6.1, block Cimmino is used to solve the SHERMAN3 linear system from the Harwell-Boeing matrix collection (Duff, Grimes and Lewis 1997). Section 6.2 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). We then perform some experiments with a parallel implementation in Section 6.3.

6.1 Solving the SHERMAN3 problem

The matrix SHERMAN3 is a symmetric 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 original pattern of matrix SHERMAN3 is shown in Figure 6.1. In the first experiment, a trivial 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.

The spectrum of E_{RJ} from the original matrix with a straightforward partitioning as shown in Figure 6.1 is given in Figure 6.2. This shows a large cluster of eigenvalues around 1 but a few trailing eigenvalues associated with bad conditioning (smallest eigenvalue of the order of 10^{-8}).

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

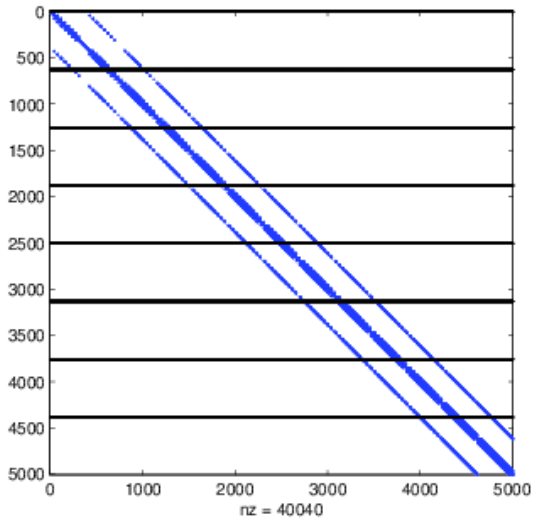


Figure 6.1: Sparsity pattern of the SHERMAN3 matrix. The matrix has been partitioned into 8 blocks of rows.

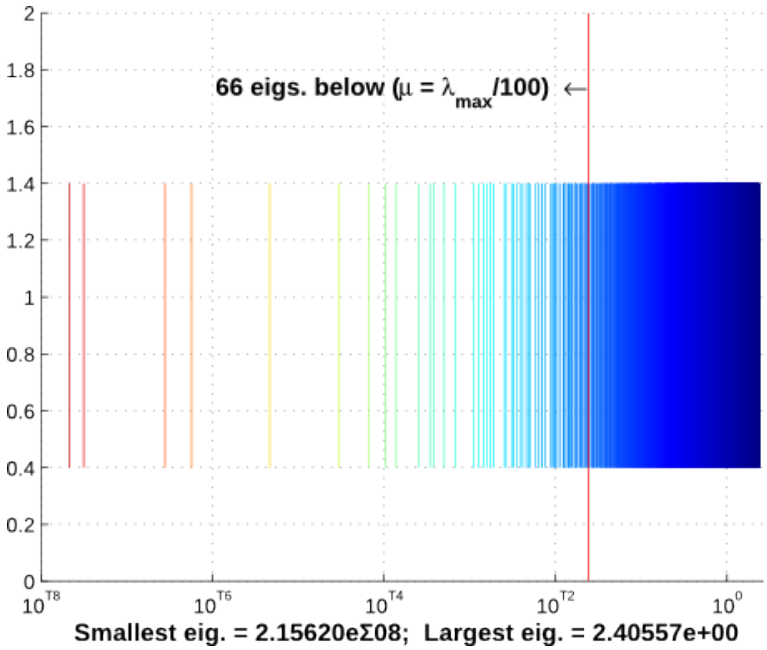
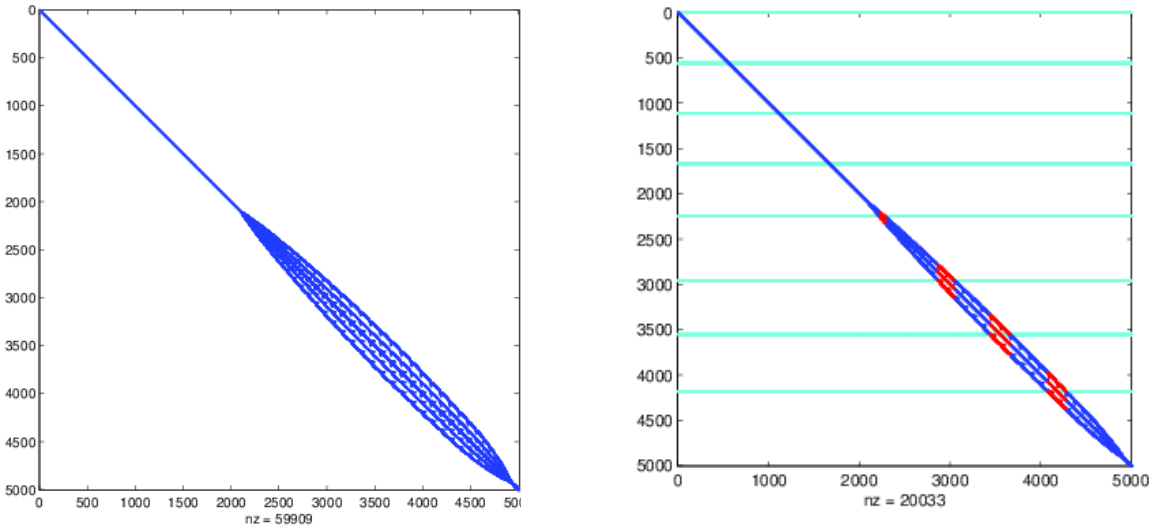


Figure 6.2: Eigenvalue spectrum of E_{RJ} for the SHERMAN3 problem.



(a) (b)

Figure 6.3: (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.

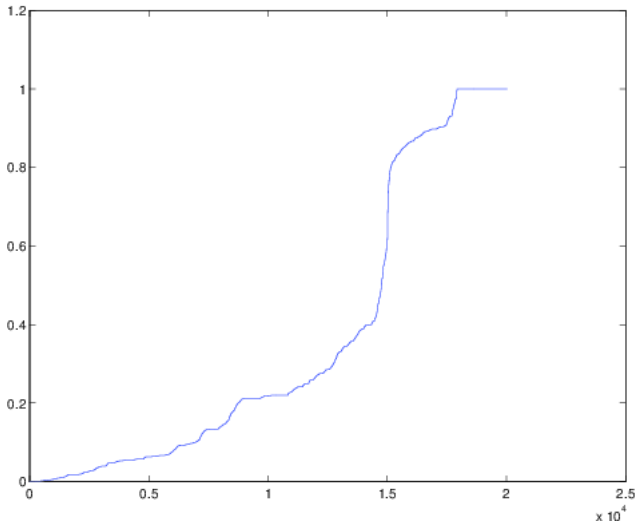


Figure 6.4: Normalized nonzero entries in the normal equations matrix from SHERMAN3. On the x-axis, the entries are displayed in increasing order of their absolute numerical value (given on the y-axis).

By doing so, we improve the parallel execution of the block Cimmino method. We do not illustrate this in the following because it is not very important for discussing and comparing the different preprocessing and partitioning strategies, and we refer to Drummond (1995) for more technical details on the distributed memory implementation of the block Cimmino method accelerated with a block-CG algorithm.

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 6.3-b shows the pattern of matrix SHERMAN3 after row permutation following the preprocessing strategy I, using a level set algorithm, Cuthill-McKee, to permute the normal equations matrix into block tridiagonal form, as indicated in Figure 6.3-a. The partitioning shown 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 first set of experiments described above, we again aim at obtaining 8 blocks of about 625 rows each.

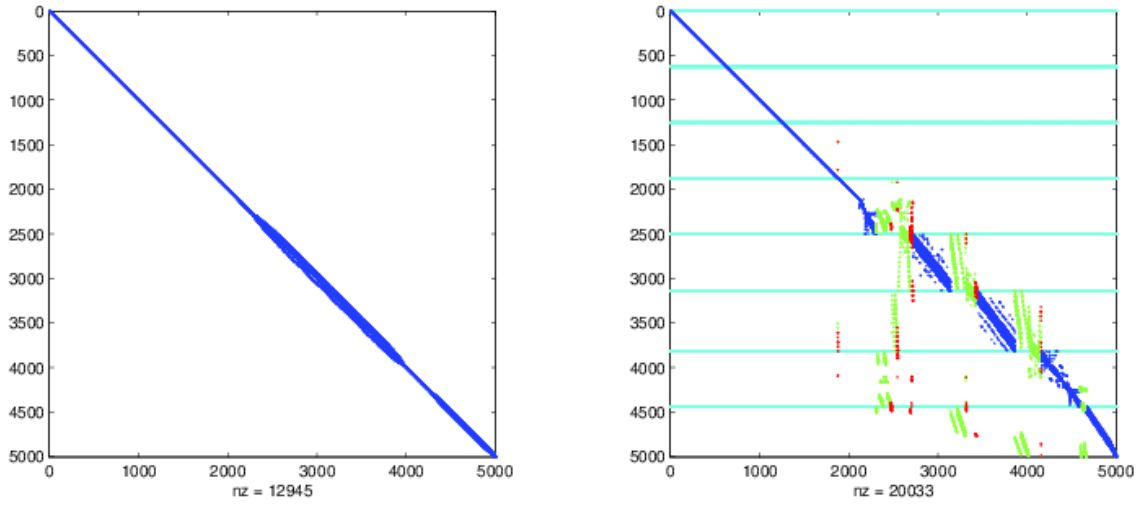
The sparsity pattern of the permuted SHERMAN3 matrix after completion of preprocessing strategy II is shown in Figure 6.5-b. In this case we dropped the normalized entries of the normal equations matrix that are below 0.2. We plot in Figure 6.4 all the entries of the normal equations, where we see that dropping at 0.2 will remove less than the half of 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 6.1, has been defined from the block tridiagonal structure of the matrix $\hat{\mathbf{B}}$ in (4.10), as described in Section 4, with the aim of again obtaining 8 blocks of about 625 rows each whenever possible.

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

Also, we notice that the matrix $\hat{\mathbf{B}}$ in Figure 6.5-a has a smaller bandwidth than the matrix in Figure 6.3-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 partition.

The spectrum of E_{RJ} for strategies I and II are shown in Figures 6.6 and 6.7, respectively. Notice that compared to the spectrum of E_{RJ} arising from the original matrix the spectrum presents a better clustering of the eigenvalues. In strategy I the largest eigenvalue is 2 which confirms a two-block partitioning. However in strategy II we get a largest eigenvalue only slightly larger than 2 and thus we call strategy II a **near two block partitioning**. The convergence results in Table 6.2 illustrate the effect of the better clustering of eigenvalues obtained with strategy II.

The second column of Table 6.2 shows convergence 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, we show them in the last column to easily compare the amount of work. We note that the uniform partitioning



(a) (b)

Figure 6.5: (a) Sparsity pattern of matrix $\hat{\mathbf{B}}$ in (4.10), 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.

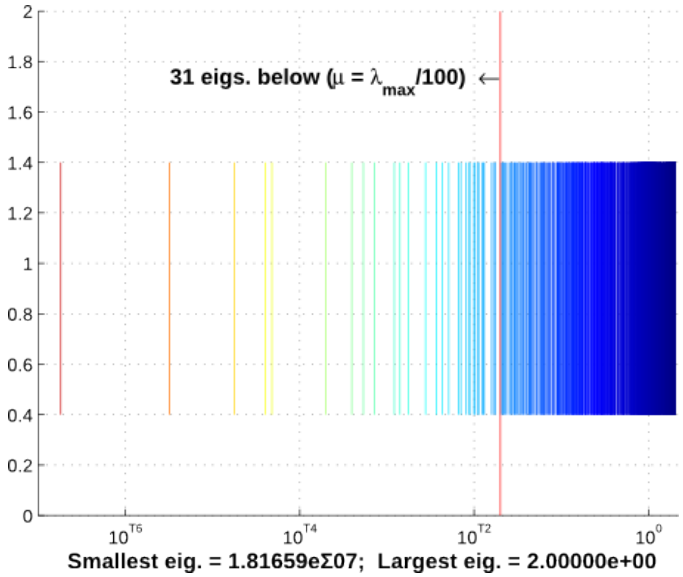


Figure 6.6: Eigenvalue spectrum of E_{RJ} for the permuted SHERMAN3 problem arising from using **strategy I.**

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

Table 6.1: Description of the 8 block-row partitions obtained for matrix SHERMAN3 in the different sets of experiments.

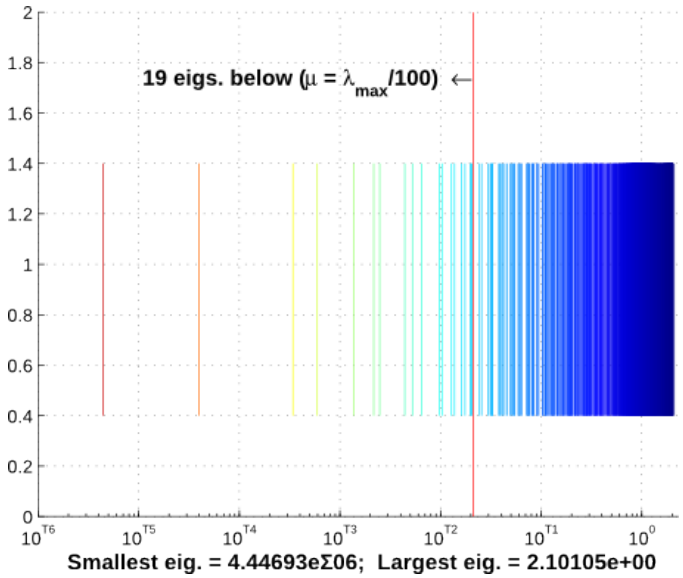


Figure 6.7: Eigenvalue spectrum of E_{RJ} for the permuted SHERMAN3 problem arising from using **strategy II** with a drop at 0.2.

requires the 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 value from 0.1 to 0.2, we get a finer partition with the higher values in the blocks.

However, as we increase the dropping value 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 loose balancing and tolerate some large partitions. By doing this, we have larger partitions and are able to reduce the interconnections between them. We notice that effect when tightening the balancing where the iteration count rises to 74.

6.2 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 6.8 where we have superimposed 16 uniform partitions.

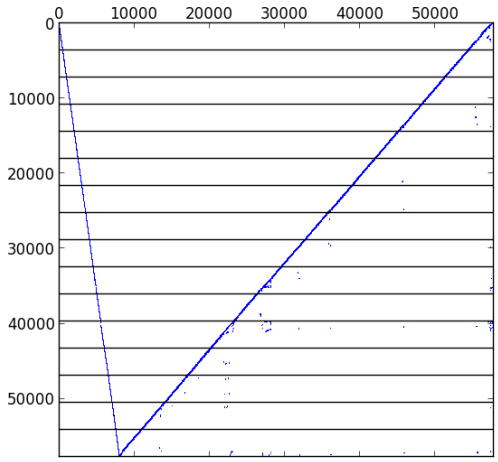


Figure 6.8: Nonzero pattern of the matrix `bayer01`.

	Number of iterations	Matrix-Vector operations
Original Matrix A	190	1520
Preprocessing strategy I	102	816
Preprocessing strategy II (drop 0.1)	73	584
Preprocessing strategy II (drop 0.2)	68	544
Preprocessing strategy II (drop 0.4)	89	712
`PaToH` partitioner (weak balancing)	52	416
`PaToH` partitioner (strong balancing)	74	592

Table 6.2: Convergence of SHERMAN3 in the three different sets of experiments.

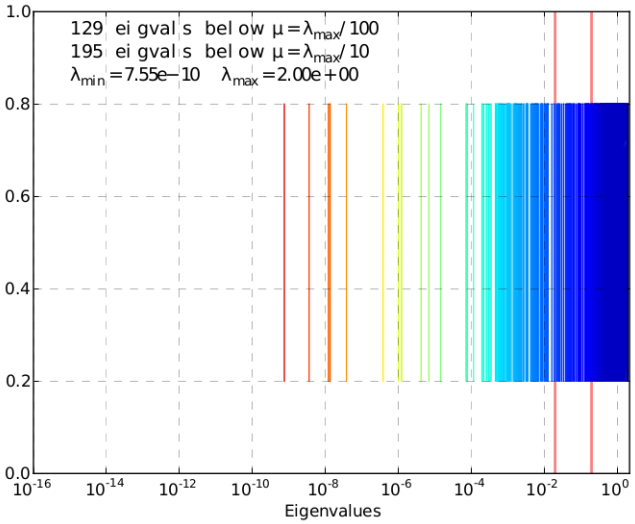


Figure 6.9: Spectrum of block Cimmino iteration matrix for `bayer01` with 16 uniform partitions.

If we run the block Cimmino algorithm on the matrix partitioned as in Figure 6.8 then the resulting spectrum of the iteration matrix is shown in Figure 6.9. We see that, while there is a good clustering of the eigenvalues around the value 1, just as we have 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 6.10 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.

Table 6.3 summarizes the partitioning information after applying the different strategies. As there are 16 partitions, we show only the smallest, the largest and the ratio between the largest and the smallest partition. We notice that when using strategy I we are constrained by the size of the level sets so that size balancing of the partitions is quite difficult, and the largest partition is more than 2.5 times the size of the smallest partition. As expected, by using strategy II we obtain better load balancing between partitions and this improves as we increase the dropping parameter. In the case of `PaToH`, weak balancing gives the greatest freedom for partitioning, and our largest partition 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 show in Table 6.4 the convergence 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 partitioning the original matrix. This behaviour was seen on several problems which makes this strategy unreliable. Finally, using `PaToH` both with weak and strong balancing gives better results.

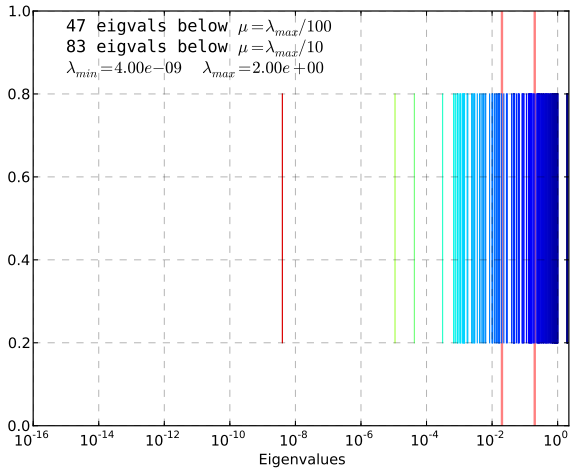


Figure 6.10: Spectrum of block Cimmino iteration matrix for `bayer01` with 16 partitions obtained using the hypergraph partitioner `PaToH`.

	Smallest partition	**Largest partition**	**Ratio**
Original Matrix A	3608	3615	1.002
Preprocessing strategy I	1673	4342	2.595
Preprocessing strategy II (drop 0.05)	1448	4017	2.774
Preprocessing strategy II (drop 0.1)	3318	3660	1.103
Preprocessing strategy II (drop 0.2)	3522	3634	1.032
`PaToH` partitioner (weak balancing)	952	7597	7.98
`PaToH` partitioner (strong balancing)	3608	3609	1

Table 6.3: Information on the 16 block-row partitions obtained for matrix `bayer01` using different strategies.

6.3 Parallel Experiments

In this section we solve some standard test problems with a parallel version of our method. This version uses a distributed Block-CG acceleration where the partitions are distributed over multiple processes. All these processes solve the problem in a distributed manner and communicate only for dot product and matrix-vector computations. The matrix-vector part, performed by `MUMPS` is also in parallel and uses supplementary cores that are not being used within the distributed Block-CG acceleration.

We list in Table 6.5 the linear systems that we will solve in parallel. We saw that `PaToH` partitioning gives better results compared to other partitioning strategies. Thus, we only do runs with `PaToH` in this section. We indicate the block-size used during the Block-CG acceleration, a

	Number of iterations	Matrix-Vector operations
Original Matrix A	256	2048
Preprocessing strategy I	459	3672
Preprocessing strategy II (drop 0.05)	270	2160
Preprocessing strategy II (drop 0.1)	204	1632
Preprocessing strategy II (drop 0.2)	343	2744
`PaToH` partitioner (weak balancing)	52	416
`PaToH` partitioner (strong balancing)	105	840

Table 6.4: Convergence of `bayer01` with the different partitioning strategies.

block-size of 1 is simply the classical CG acceleration. We will compare the effect of balancing when using `PaToH`, and see whether the lower iteration count of weak balancing compensates for its poorer balance of partition sizes.

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

Table 6.5: Linear systems from the University of Florida Sparse Matrix Collection.

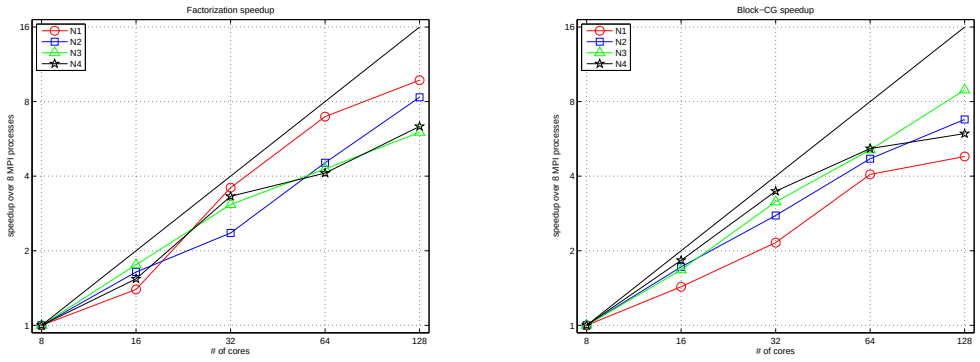
In the previous section, we noticed that weak balancing gave faster convergence but with a poorer balance on partition sizes. The drawback of such a strategy is that a large partition will take longer to factorize than others and therefore stall other processes until it finishes. To illustrate this, we did some runs on the `cage13` problem with a 32 core machine. In this case, we varied the balancing parameter and partitioned the matrix into 256 partitions. 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. We notice here that the size of partitions affected the factorization and slowed it down whereas it did not change the iteration count by much. However, in the cases where the iteration count is dramatically reduced, we might gain from the Block-CG phase more than what we lose in the factorization phase.

The following runs are on a machine with nodes having two quad-core Nehalem processors

and 36GB 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 and 64 mpi-processes.

	Factorization						**Block-CG**					
Cores	1	8	16	32	64	128	1	8	16	32	64	128
N_1	46.89	12.11	8.68	3.37	1.74	1.24	21.57	6.34	4.43	2.94	1.56	1.32
N_2	83.27	14.41	8.78	6.11	3.18	1.73	272.21	63.49	36.89	22.88	13.50	9.37
N_3	77.09	28.15	16.04	9.16	6.58	4.68	41.67	13.64	8.14	4.33	2.67	1.53
N_4	34.8	6.67	4.33	2.01	1.62	1.05	3905.0	773.0	422.4	22.13	149.52	130.13

Table 6.6: Parallel results showing the elapsed time in seconds for the factorization of the augmented systems and the Block-CG acceleration.



(a) Factorization speedup

(b) Block-CG speedup

Figure 6.11: Parallel results of the factorization of augmented systems and Block-CG acceleration speedups.

We show in Table 6.6 the time spent in seconds to factorize the augmented systems and the time taken for the Block-CG to converge.These results are also shown in Figures 11(a) and 11(b).

In these runs we have performed a strong scaling analysis showing the speedups when a single system is solved on an increasing number of processors. As we see in both plots in

Figure 6.11, we obtain a good speedup on all four problems. However, the speedup is not uniform. The speedup is very good for each problem until the number of processors is equal to the number of block partitions when it becomes somewhat poorer. The reason for this lies in the fact that our algorithm exploits parallelism at two-levels. As we start increasing the number of processors, the size of the subproblems decrease and the increasing instances of MUMPS obtain a speedup from the fact that they are factorizing smaller problems as well as the fact that the work in the block conjugate gradient algorithm is split across more processors. When we increase the number of processors further, the number of instances of MUMPS does not increase but, as it is a parallel code, it will be able to use more processors to effect the factorization and solution of the subproblems. However, the conjugate gradient iterations are still constrained by the number of partitions and so do not fully exploit the extra processors. Further discussion on this aspect of our work is outwith the scope of this paper.

7 Conclusions

The results of our work shows that preprocessing the original system of equations to obtain a different partitioning of the system can have a profound 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 good blocking of the original matrix. We thus tried Strategy II which did much better with respect to a regular partitioning but at the cost of losing to a controlled extent 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 straight partitioning of the original system. It also required the selection of a dropping parameter that was dependent on the problem being solved and so 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 interconnection between block partitions. We use the PaToH software to effect this partition and found that it compared very favourably to our strategies based on the normal equations and was also more robust in that it always produced partitions better than a uniform partition of the initial matrix. It was also possible to use PaToH to obtain a well balanced partition (that is with sublocks 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 and we found that it is usually better to use a strong balancing strategy to avoid bottlenecks from having disparately sized tasks. Although the speedup is not uniform, it is in general very good at obtaining parallelism from both the partitioning of the problem and from a parallel sparse direct solver.

Our implementation of the Block Cimmino method uses Block Conjugate Gradients to accelerate its convergence rate. In order to exploit parallelism, we have developed an MPI implementation of our Block Cimmino with Block CG acceleration. We have used strong and weak partitioning strategies to define the subblock sizes (number and choice of rows in 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 processing elements to the size of the block in a manner that reduces the iteration count, minimizes load imbalance, and produces better resource utilization. Our experiments have indicated that the block Cimmino computational scheme sets a very good stage for future numerical schemes to be run on heterogeneous multi-level parallel systems.

References

Arioli, M., Drummond, A., Duff, I. S. and Ruiz, D. (1994), A parallel scheduler for block iterative solvers in heterogeneous computing environments, Technical Report TR/PA/94/15, CERFACS, Toulouse, France.

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.os u.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/`'.

Drummond, T. (1995), Solution of general linear systems of equations using block Krylov based iterative methods on distributed computing environments, Phd thesis, Institut National Polytechnique de Toulouse. CERFACS Technical Report, TH/PA/95/40.

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.

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.

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.

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.