Technical Report
RAL-TR-2004-040

Parallel preconditioners based on partitioning sparse matrices

Iain S. Duff, Songklod Riyavong, and Martin B. Van Gijzen

December 23, 2004

ISSN 1358-6254

RAL-TR-2004-040

Parallel preconditioners based on partitioning sparse matrices[1]

Iain S. Duff[2], Songklod Riyavong[3], and Martin B. Van Gijzen[3]

ABSTRACT

We describe a method for constructing an efficient block diagonal preconditioner for accelerating the iterative solution of general sets of sparse linear equations. Our method uses a hypergraph partitioner on a scaled and sparsified matrix and attempts to ensure that the diagonal blocks are nonsingular and dominant. We illustrate our approach using the partitioner PaToH and the Krylov-based GMRES algorithm. We verify our approach with runs on problems from economic modelling and chemical engineering, traditionally difficult applications for iterative methods. Our approach and the block diagonal preconditioning lends itself to good exploitation of parallelism. This we also demonstrate.

Keywords: hypergraph model, matrix partitioning, block diagonal preconditioning, parallel matrix-vector product, parallel iterative method.

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

Running title: Parallel preconditioners based on matrix partitioning

---

[1]Current reports available by anonymous ftp to `ftp.numerical.rl.ac.uk` in directory pub/reports. This report is in file drvRAL2004040.ps.gz. The report also available through the URL `www.numerical.rl.ac.uk/reports/reports.html`.

It is also available as the CERFACS Technical Report TR/PA/04/114.

[2]i.s.duff@rl.ac.uk. This work was supported by the EPSRC Grant GR/S42170.

[3]riyavong@cerfacs.fr, gijzen@cerfacs.fr. CERFACS, 42 Ave G. Coriolis, 31057 Toulouse, France.

Computational Science and Engineering Department
Atlas Centre
Rutherford Appleton Laboratory
Oxon OX11 0QX

December 23, 2004

Contents

1 Introduction

Recently a number of hypergraph partitioning algorithms have been proposed for partitioning sparse matrices for performing matrix-vector products on a parallel computer. These algorithms determine a row, column, or block partitioning of the matrix such that the matrix can be distributed evenly over the processors while the communication due to overlapping columns or rows is minimised. Among these we mention PATOH (Çatalyürek and Aykanat 1999*b*), MONDRIAAN (Vastenhouw and Bisseling 2004), and HMETIS (Karypis and Kumar 1998). Based on such a partitioning it is possible to make an efficient parallel implementation of the matrix-vector products, which combines good load balancing with low communication volumes. The matrix-vector product is an important building block in many numerical algorithms.

Krylov subspace methods (Axelsson 1996, Saad 2003, van der Vorst 2003), for the iterative solution of linear systems of equations are a prime example of such an algorithm. These iterative methods are composed of only a few different operations: matrix-vector multiplication, inner product operations, vector updates, and preconditioning operations. The first three operations can be parallelised efficiently by making a row-partitioning using a hypergraph algorithm, see (Riyavong 2003*a*, Çatalyürek and Aykanat 1995, Çatalyürek and Aykanat 1999*a*, Vastenhouw and Bisseling 2004). The preconditioning operation, however, requires special attention.

In this paper, we study how to preprocess and row-partition the matrix in a way that allows us to construct an efficient preconditioner. We focus on block diagonal preconditioners because of their suitability for parallel computing. In order to be able to construct an efficient preconditioner the most relevant information in the matrix must be contained in the diagonal blocks. Hypergraph partitioners that minimize only the communication volume do not take into account the magnitude of the entries. Consequently, small entries are considered as important as large entries. For preconditioners, however, the magnitude of the entries influences the performance, and one can expect that a block diagonal preconditioner performs better if large entries are contained in the diagonal blocks.

We apply a combination of techniques to achieve this goal. Firstly, we maximise the entries on the main diagonal of the matrix. For this we use the HSL (HSL 2004) routine MC64 (Duff and Koster 1999, Duff and Koster 2001). Secondly, we apply the partitioner to a *sparsified* matrix, from which elements smaller than a tolerance are dropped. The tolerance is determined experimentally so that the Frobenius norm of the block diagonal preconditioner is maximised. Although we expect that the number of iterations is decreased by using this criterion, it is not necessarily the best one in terms of reducing the number of iterations.

We show experimentally that block diagonal preconditioners constructed by the preprocessing mentioned above give satisfactory results on our set of test matrices. All examples are solved with a modest number of iterations. Moreover, although the number of iterations usually grows when the number of processors is increased, we observe that this adverse effect can be reduced significantly using our dropping strategy. In all our experiments, we use the GMRES package (Frayssé, Giraud, Gratton and Langou 2003).

In Section 2, we briefly review the preconditioned GMRES method for solving general square matrices. In Section 3, we describe the hypergraph model of a general sparse matrix and hypergraph partitioning. The hypergraph is the structure of choice for general sparse matrices because it is not necessary to symmetrize the matrix as required by a simple graph model. We describe in detail the construction of block diagonal preconditioners in Section 4 and summarize the preprocessing algorithm. The numerical results and a discussion of them are given in Section 5. Finally, we present some conclusions in Section 6.

2 Review of preconditioned GMRES

In this section, we review ideas of preconditioned GMRES that are necessary to analyse the experimental results. For a full description of standard GMRES, the reference Saad and Schultz (1986) should be consulted. Consider the right preconditioned system

$$\begin{aligned} AM^{-1}z &= b \\ M^{-1}z &= x \end{aligned} \tag{2.1}$$

where M is the block diagonal preconditioner. The Krylov subspace is defined by

$$K^m(AM^{-1}, r_0) = span\{r_0, AM^{-1}r_0, \ldots, (AM^{-1})^{m-1}r_0\}. \tag{2.2}$$

We use GMRES(m), that is we restart the iteration every m steps. In the Arnoldi process that is used to construct orthonormal basis vectors for $K^m(AM^{-1}, r_0)$, the modified Gram-Schmidt orthonormalization scheme is usually preferred for reasons of numerical stability but in this work we use the classical Gram-Schmidt with reorthogonalization to construct orthonormal basis vectors. The latter scheme is the method of choice for parallel computing (Frayssé, Giraud and Kharraz-Aroussi 1998, Giraud, Langou and Rozložník 2002). GMRES(m) with a right preconditioner is given in the algorithm in Figure 2.1.

Step 3 of the algorithm can be rewritten as

$$w_k = Az \quad \text{and} \quad Mz = v_k. \tag{2.3}$$

In this work, the second system is solved by factorizing M into lower and upper triangular matrices using either of two different methods: complete decomposition using a sparse direct method and ILU(0). We compare the results of the two methods in terms of computing time and the number of iterations. Steps 4–5 are the classical Gram-Schmidt orthogonalization for w_k against all previous v_i. To ensure that we obtain the orthogonal basis with working precision, the resulting vectors are reorthogonalized if necessary.

To solve a linear system with GMRES in parallel, we distribute the coefficient matrix and the vectors among the processors. We partition the matrix row-wise and assign the row partitions to the processors. The matrix partitioning should be such that each processor has an evenly balanced computing load and the communication among them should be minimized. We discuss the matrix partitioning in the next section.

```
 1. Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|$, and $v_1 = r_0/\beta$
 2. **for** $k = 1 : m$
 3.     $w_k = AM^{-1}v_k$
 4.     $h_{ik} = (w_k, v_i)$, $i = 1 : k$
 5.     $w_k = w_k - \sum_{i=1}^{k} v_i h_{ik}$
 6.     $h_{k+1,k} = \|w_k\|_2$
 7.     $v_{k+1} = w_k / h_{k+1,k}$
 8.     Define $V_k = [v_1 \ \ldots \ v_k]$ and $\overline{H}_k = h_{ij}$, $1 \leqslant i \leqslant k+1; 1 \leqslant j \leqslant k$
 9. **end for**
10. Compute $y_m$ from $\min_{y \in R^m} \|\beta e_1 - \overline{H}_m y\|_2$, and $x_m = x_0 + M^{-1}V_m y_m$
11. **if** convergence **stop**, **else** set $x_0 = x_m$ and goto 1.
```

Figure 2.1: The GMRES algorithm

3 Partitioning a general sparse matrix

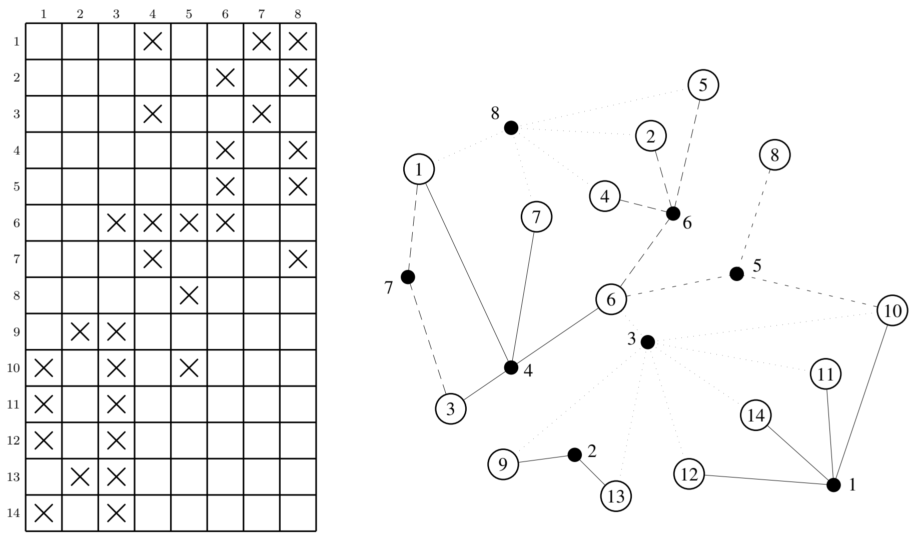

Figure 3.1: Rectangular sparse matrix (left) and hypergraph representation (right)

Undirected graphs have many shortcomings for modelling general sparse matrices (Hendrickson 1998, Hendrickson and Kolda 1998, Hendrickson and Kolda 2000*a*, Hendrickson and Kolda 2000*b*). General sparse rectangular matrices, or sparse square matrices with an unsymmetric pattern, however, can be modelled in a straightforward way using hypergraphs. Basically, the hypergraph model $H(V, N)$ of a sparse matrix consists of vertices V and hyperedges or nets N. Figure 3.1 illustrates a hypergraph representation of a general sparse matrix. The vertices represent the row indices and the hyperedges represent the column indices. In the figure, different linestyles, which have no common vertices, correspond to different hyperedges (columns). For example, the solid line with solid circle labelled 4 represents the fourth hyperedge which corresponds to the fourth column of the matrix containing the row indices 1, 3, 6, and 7. Similarly, we can use the column indices of the matrix as the hypergraph vertices and the row indices as the hyperedges. Since in this work we partition the matrix rowwise, the vertices of the hypergraph represent the row indices.

To ensure that we partition the matrix so that a good load balance is achieved while the communication is kept low, we partition the corresponding hypergraph so that the hyperedge-cuts are minimized and the vertices are partitioned evenly. The hyperedge-cut corresponds directly to the communication cost between processors in parallel computing, and therefore this quantity has to be minimized. The vertex partitioning corresponds to the distribution of the work load among the processors and it should be evenly partitioned for good load balancing. However, both goals cannot usually be attained at the same time, and a trade-off has to be made between minimizing communication cost and optimizing the load balance. This trade-off can be influenced by assigning weights to the vertices and hyperedges. In this work we apply a simple weighting strategy in which we assign as a weight for a hyperedge the number of entries in the corresponding column (Çatalyürek and Aykanat 1995, Çatalyürek and Aykanat 1999*a*). Assigning weights to both vertices and hyperedges before partitioning can improve the load balance, but at the expense of a higher communication cost.

The hypergraph partitioning tool PATOH (Çatalyürek and Aykanat 1999*b*) is used to partition all the test problems. We use this partitioner as a black box in our preprocessing algorithm although we could have used other partitioners like HMETIS (Karypis and Kumar 1998) and MONDRIAAN (Vastenhouw and Bisseling 2004) as an alternative. We selected PATOH on the basis of comparative experiments described by Riyavong (2003*b*), mainly on the basis of superior partitioning times.

The partitioner uses a multilevel algorithm. This algorithm consists of three phases: coarsening, initial partitioning, and uncoarsening and refinement. In the coarsening phase the vertices are grouped on the basis of the criterion described by Çatalyürek and Aykanat (1999*b*). Each group becomes a new vertex which has as its weight the sum of the weight of its entries. This phase reduces the number of vertices. This phase continues until the number of vertices becomes smaller than a predefined value, for example, a few hundred. The next phase partitions the weighted coarsened hypergraph, using heuristic algorithms described by Fiduccia and Mattheyses (1982) and Kernighan and Lin (1970). This phase starts with a random partitioning. Since the size of the hypergraph is small, the partitioning is run many times with different random starts and the best partitioning is selected for the next step. The last phase uncoarsens the partitioned coarsest hypergraph back to the original one. At each level of uncoarsening, the partitioning is refined by running a Fiduccia-Matheyses based iterative improvement heuristic on the hypergraph, starting from the initial partitioning (the coarsest hypergraph). We refer to Karypis (1998), Çatalyürek and Aykanat (1999*a*), and Çatalyürek and Aykanat (1999*b*) for more details and the references therein.

4 Constructing a block diagonal matrix for parallel computing

In this section we describe the construction of a block diagonal matrix for computation using GMRES. This is composed of the following steps: equilibrating the matrix, dropping entries, partitioning the *sparsified* matrix, and forming a block diagonal matrix. The result of these procedures is the matrix with the following block structure

$$B = \begin{pmatrix} B_{11} & B_{12} & \dots & B_{1K} \\ B_{21} & B_{22} & \dots & B_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ B_{K1} & B_{K2} & \dots & B_{KK} \end{pmatrix}, \tag{4.1}$$

where B_{ij} is a submatrix, with the block diagonal matrix given by $D = \text{diag}(B_{11}, B_{22}, \dots, B_{KK})$.

In the first step, we will place large entries on the main diagonal. To ensure that this is done efficiently, the matrix is scaled and permuted columnwise using the HSL routine MC64 (Duff and Koster 1999, Duff and Koster 2001). This transforms the original matrix into an *I-matrix* (Olschowka and Neumaier 1996). As a result, the diagonal blocks are structurally nonsingular (unless the matrix is singular). Moreover, a frequent observation is that this procedure also improves the conditioning of the matrix.

The next steps are an iterative procedure of dropping, partitioning, and constructing a block diagonal matrix for dropping parameters varying from 0.00 to 0.50, including the case with no drop. This determines experimentally an optimal dropping parameter in the range mentioned above which maximizes the relative Frobenius norm of the diagonal blocks

$$r = \frac{\|D\|_F}{\|B\|_F}. \tag{4.2}$$

With this heuristic dropping criterion we expect to improve the efficiency of the preconditioner. However, it it not necessarily the best criterion in terms of the number of iterations. The advantage of this dropping criterion is that it is not so expensive to compute. The purpose of dropping is to partition the sparsified matrix using only the large entries and to distribute them evenly among the processors. To do that, entries which are smaller than the (optimal) dropping tolerance τ are dropped before partitioning.

After dropping entries smaller than τ, we partition the sparsified matrix, obtaining a row permutation P. Then we apply P symmetrically to the scaled matrix (not the sparsified matrix) and this transforms it to the block matrix (4.1), the diagonal blocks of which will be used as the preconditioner. This preprocessing phase can be summarized by the algorithm in Figure 4.1.

Dropping entries before partitioning has the advantage that the diagonal blocks become more dominant but it also has the disadvantage that those entries not taken into account in the partitioning step have to be used to construct the matrix B. If these entries are not in the diagonal blocks, they will increase the communication cost.

```
     INPUT  matrix $A_0$ and a number of blocks $K$
    OUTPUT  A block matrix $B$ and optimal dropping tol $\tau$
        1.  $A_1 = D_r A_0 D_c Q$ using MC64
        2.  For $\tau = 0.0$ To $0.5$ Step $= 0.01$
        3.      $A_2 = A_1$ excluding entries whose magnitude $\leqslant \tau$
        4.      Partition rows of $A_2$ into $K$ parts, obtaining P
        5.      $B = PA_1P^T$
        6.      Compute $r$ using Equation (4.2)
        7.      If $r$ has the largest value so far, record the value of $\tau$
        8.  End For
```

Figure 4.1: Algorithm for determining drop tolerance.

Having constructed the matrix B, we will use it as a right preconditioner for GMRES in a parallel implementation in the following manner. We solve

$$BD^{-1}y = b, \quad \text{where} \quad Dx = y. \tag{4.3}$$

The blocks of rows are distributed among the processors as described by Riyavong (2003*a*), for example, the block $[B_{i1}, B_{i2}, \ldots, B_{iK}]$ is assigned to the ith processor. So solving the block diagonal systems in the preconditioning operation in this processor amounts to computing $v_i = D_i^{-1}u_i$, where $D_i = B_{ii}$ and u_i, v_i are the parts of the input/output vectors assigned to the processor. The above operation can be written as

$$D_i v_i = u_i. \tag{4.4}$$

In this work, we solve these equations accurately using the direct sparse solver MA48 (Duff and Reid 1996) from HSL (HSL 2004), or approximately using the ILU(0) routine from SPARSKIT (Saad 1990). If MA48 is used to evaluate the block diagonal preconditioner, we thus solve equation (4.4) with a complete LU decomposition. The fill-in of this decomposition can perturb the load balance among the processors and this method can be quite expensive when compared to incomplete decomposition. However, this method is considered because of its robustness. In ILU(0) all fill-ins are discarded. As a consequence, the load balance is maintained. This method, however, is less robust than a direct solution method.

5 Numerical experiments

5.1 Description of test matrices

Six matrices from different application areas are selected from Davis (2004) and Duff, Grimes and Lewis (1997). Their characteristics are shown in Table 5.1. The first matrix, `sherman2`, is from oil reservoir simulation. It arises in a three dimensional simulation model on a $6 \times 6 \times 5$ grid using a seven-point finite-difference approximation with 5 equations

matrix	n	nnz	pattern	application area
sherman2	1080	23094	unsym	Oil reservoir simulation
jan99jac040	13694	82842	unsym	Economic model
perrel	20700	511050	sym	PDE with chemistry
wang3	26064	177168	sym	Semiconductor device simulation
bayer01	57735	277774	unsym	Chemical process simulation
venkat25	62424	1717792	sym	Unstructured 2D Euler solver

Table 5.1: Test matrices

matrix	condest	
	orig	new
sherman2	1.42E+12	4.91E+02
jan99jac040	1.31E+14	3.90E+07
perrel	4.29E+03	1.45E+04
wang3	1.07E+04	3.43E+03
bayer01	3.33E+19	2.72E+04
venkat25	1.17E+08	1.68E+04

Table 5.2: condest of matrices before and after applying MC64

and unknowns per grid block. Matrix `jan99jac040` is a Jacobian matrix which arises from using Newton's method to solve a forward-looking macroeconomic model from the Bank of Canada. Next, `perrel` is from applying a finite-volume discretization to the Navier-Stokes equations coupled with chemistry. Matrix `wang3` is from the discretization of the electron continuity equation of a 3D diode with a piecewise doping profile in a nonuniform mesh. Matrix `bayer01` arises in chemical process simulation in the German chemical industry. This matrix is very ill-conditioned. The last matrix, `venkat25`, arises in computational fluid dynamics when solving a 2D unstructured Euler problem.

Table 5.2 compares the condition numbers (condest) of the matrices before and after scaling using MC64. The condition numbers are computed using the HSL routine MC41 and MUMPS(Amestoy, Duff, L'Excellent and Koster 2001). We find that the condition number of most matrices, especially the ill-conditioned matrices `sherman2`, `jan99jac040`, and `bayer01`, can be improved dramatically by scaling.

5.2 Results of dropping

Before partitioning the scaled matrix, we drop the entries with small modulus to ensure that large entries are contained in the diagonal blocks. To determine the optimal dropping parameters, we plot r against the dropping parameter which is obtained from the algorithm in Figure 4.1. For example, the optimal dropping parameter for matrix `jan99jac040` for

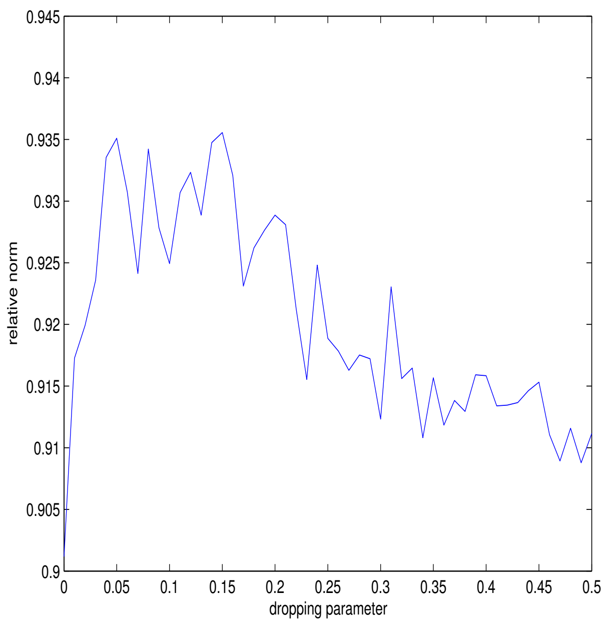

Figure 5.1: The plot of relative norm against dropping parameter for `jan99jac040`

4 partitions is 0.15 as shown in Figure 5.1. The optimal dropping parameters for all test matrices with number of partitions of 2, 4, 8, and 16 are given in Table 5.3. In the table, nnz(E) is the number of entries which are dropped before partitioning. The label 'no drop' means that the block diagonal matrix constructed without dropping has the highest Frobenius norm of the diagonal blocks. It is different from the case $\tau = 0$ because some of the matrices have entries that are explicitly zero. So dropping the zero entries can increase the norms of the diagonal blocks. Dropping with optimal τ optimises the Frobenius norms of the diagonal blocks of the partitioned matrix as shown in Table 5.4 where we compare the results of dropping to no drop. Hence we can expect a better performance of the resulting block diagonal preconditioners. We remark that the optimal value of the dropping parameter using our criterion is not guaranteed to be the same as the value that results in the minimum number of iterations. We expect, however, that the former should be closely related to the latter. For example, Figure 5.2 plots the number of iterations against the dropping parameters. The dropping parameter that gives the minimum number of iterations is 0.05 while the one obtained from our criterion is 0.15 but we can see that both of them give an almost equal number of iterations.

However, the dropping has a severe drawback. The entries outside the diagonal blocks entail communication among processors. The entries that are dropped before partitioning are brought back to the final matrix and can increase the communication cost if they are external to the diagonal blocks. From Table 5.3, we should get a significant effect from dropping for the matrices `perrel` and `jan99jac090` where the dropping parameters are

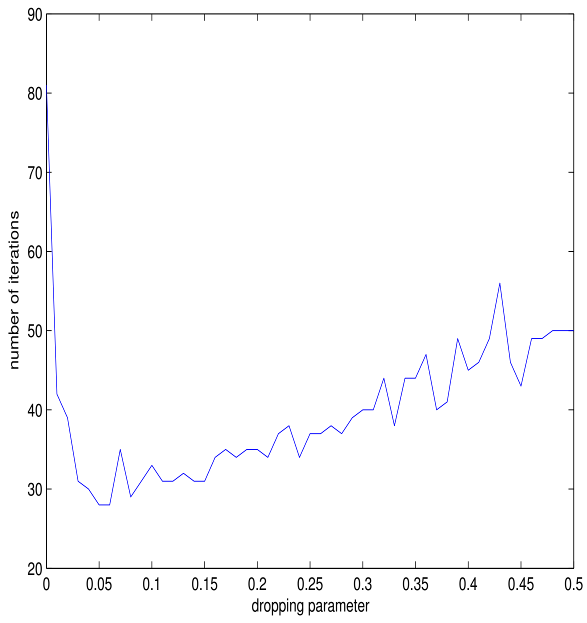

Figure 5.2: The plot of number of iterations against dropping parameter for `jan99jac040`

matrix	dropping parameters							
	2 partitions		4 partitions		8 partitions		16 partitions	
	nnz(E)	τ	nnz(E)	τ	nnz(E)	τ	nnz(E)	τ
sherman2	15396	0.01	19164	0.15	19494	0.19	19624	0.21
jan99jac040	25979	0.04	40754	0.15	30003	0.06	43444	0.19
perrel	373317	0.15	410339	0.27	414900	0.29	414900	0.29
wang3	540	0.06	540	0.06	0	no drop	0	no drop
bayer01	2680	0	2680	0	2680	0	2680	0
venkat25	0	no drop	642128	0.1	355855	0.05	29	0

Table 5.3: Optimal dropping parameter τ

matrix	$\|D\|_F/\|A\|_F$							
	2 partitions		4 partitions		8 partitions		16 partitions	
	no drop	opt drop	no drop	opt drop	no drop	opt drop	no drop	opt drop
sherman2	0.990	0.999	0.973	0.999	0.944	0.998	0.917	0.997
jan99jac040	0.940	0.971	0.913	0.935	0.880	0.911	0.871	0.896
perrel	0.997	0.999	0.995	0.999	0.989	0.996	0.985	0.993
wang3	0.998	0.998	0.996	0.996	0.994		0.991	
bayer01	0.997	0.998	0.997	0.998	0.996	0.997	0.995	0.996
venkat25	0.999		0.998	0.998	0.997	0.997	0.994	0.995

Table 5.4: Relative Frobenius norm between dropping and no dropping

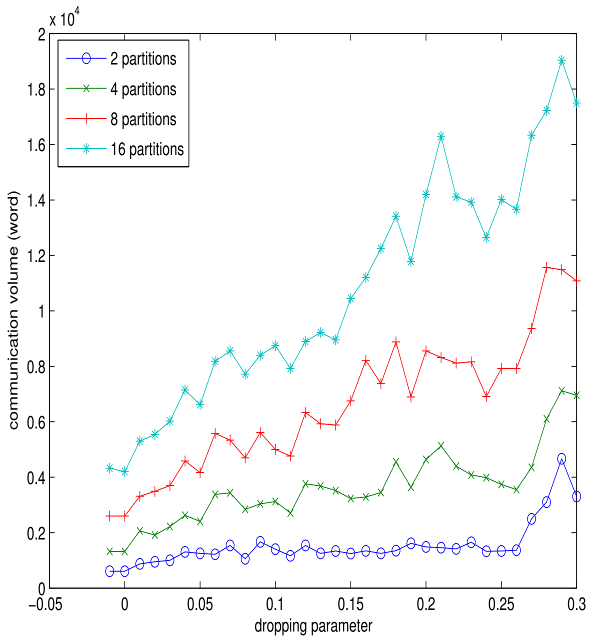

Figure 5.3: Dropping and communication volume for `perrel`

quite large. Hence, for these matrices we study the effect on the communication volumes due to the dropping. These are shown in Figures 5.3 and 5.4. The communication volume is measured in terms of data exchange between processors. Since we use double precision, the communication volume is measured in double-precision words. The communication cost among processors can be quite expensive, especially *local* communication arising in the matrix-vector products in step 3 of the algorithm in Figure 2.1, so the decrease in total elapsed times may not be noticed even though the number of iterations of the iterative solver decreases significantly. From these figures, the communication volumes increase almost monotonically with the dropping parameters. This indicates that large dropping parameters should be avoided when constructing the block diagonal matrix for parallel computing.

5.3 Results of parallel GMRES

In the experiments we use GMRES(50), the right-hand side is generated such that the solution vector is $(1, 2, 3, \ldots, n)^T$, the initial guess is $(0, 0, \ldots, 0)^T$, and the iteration is terminated when the relative residual norm $\|r\|/\|b\| < 10^{-8}$ or the number of iterations is larger than 3000. The calculations have been performed on a COMPAQ Alphaserver SC45 in which each node has 4 processors of EV68, 1 GHz, 64 KB L1 cache, 8 MB L2 cache, 8 GB shared memory. We use the number of processors equal to the number of diagonal blocks: 1 (for sequential computing), 2, 4, 8, and 16.

We first present the numerical results for unpreconditioned GMRES. Tables 5.5 and

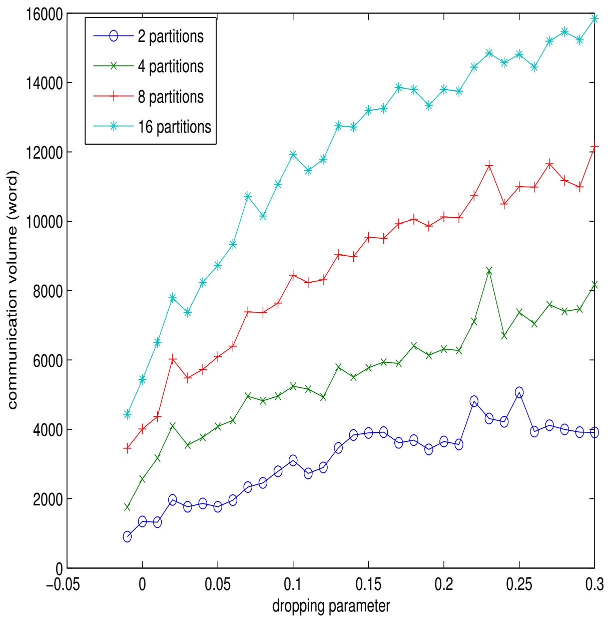

Figure 5.4: Dropping and communication volume for `jan99jac040`

5.6 show the number of iterations and the computing times of unpreconditioned GMRES, respectively. The sign † indicates that it does not converge within 3000 iterations. In fact, the matrices are already scaled by MC64 so these tables show the results of this *explicit* preconditioning. From the tables, GMRES converges to the required precision for only three matrices. Thus the unpreconditioned method is not robust. For each of these matrices, the number of iterations should be the same for any number of processors. In the case of `perrel`, the number of iterations differs slightly for different numbers of processors due to round-off effects. For studying the elapsed times, the relative speedup (Foster 1995) is defined as

$$S_{\text{rel}} = \frac{T_1}{T_K}, \tag{5.1}$$

where T_1 is the elapsed time on one processor and T_K is the time on K processors. Discarding `sherman2` because of its small size, we find that the results for `perrel` and `wang3` show superlinear speedup up to eight processors. The superlinear speedup can be attributed to cache effects. Note that, for most results, the elapsed time is slightly higher if dropping is applied, due to higher communication cost. We have to add that the measurements are made on a production machine, hence elapsed times may vary from run to run.

The number of iterations of preconditioned GMRES are shown in Tables 5.7 and 5.8 for ILU(0) and MA48, respectively. The elapsed times are shown in Tables 5.9 and 5.10. In these tables we give the results with and without dropping. We find that for the first three matrices, where the dropping parameter is relatively large, the number of iterations

matrix	Number of iterations								
	1 partition	2 partitions		4 partitions		8 partitions		16 partitions	
		dropping		dropping		dropping		dropping	
		no	yes	no	yes	no	yes	no	yes
sherman2	630	630	630	630	630	630	630	630	630
jan99jac040	†	†	†	†	†	†	†	†	†
perrel	2720	2682	2690	2615	2647	2790	2631	2560	2728
wang3	327	327	327	327	327	327	327	327	327
bayer01	†	†	†	†	†	†	†	†	†
venkat25	†	†	†	†	†	†	†	†	†

Table 5.5: Number of iterations: no preconditioner

matrix	elapsed times (sec)								
	1 partition	2 partitions		4 partitions		8 partitions		16 partitions	
		dropping		dropping		dropping		dropping	
		no	yes	no	yes	no	yes	no	yes
sherman2	0.32	0.21	0.22	0.27	0.23	0.26	0.37	0.50	0.52
jan99jac040	†	†	†	†	†	†	†	†	†
perrel	37.36	15.95	14.53	6.40	6.50	3.65	4.01	3.20	4.64
wang3	7.26	2.87	2.99	1.13	1.08	0.61	0.67	0.56	0.57
bayer01	†	†	†	†	†	†	†	†	†
venkat25	†	†	†	†	†	†	†	†	†

Table 5.6: Elapsed times: no preconditioner

is significantly decreased, both for ILU(0) and for MA48. We also notice that, for these matrices, dropping significantly increases $\|D\|_F/\|A\|_F$.

The results for the ILU(0) preconditioner in combination with GMRES(50) that are given in Table 5.7 show that the number of iterations increases with the number of processors. This is not surprising since the diagonal blocks are of smaller order if the number of blocks is increased. Without the dropping strategy, the increase in the number of iterations is very significant for the three smallest matrices, but modest for the largest three. With the dropping strategy, however, the increase in the number of iterations for the three smallest matrices is considerably reduced. Hence with dropping, the increase in the number of iterations is modest for all test matrices. We remark that the number of iterations for `bayer01` is increased if dropping is applied. This is possible because an increase in the Frobenius norm of the diagonal blocks does not guarantee a decrease in the number of iterations. To show that the observed effects are not due to the fact that we restart GMRES, we include for completeness the number of iterations for full GMRES in Table 5.11. This table confirms our observation that the optimal dropping strategy considerably reduces the increase in the number of iterations for the three smallest matrices. Moreover, for full GMRES we do not see an increase in the number of iterations for `bayer01` if we use dropping. Since the effect of dropping on the number of iterations is most prominent for the three smallest test matrices, we do not see a strong positive effect on the elapsed times that are shown in Table 5.9. For the three largest matrices we observe an almost linear speedup up to 16 processors.

In the case of MA48 we find similar results as for ILU(0). The number of iterations for one partition is 1, of course, since in that case a direct solution method is used. The number of iterations increases again with the number of processors, rather strongly for the three smallest matrices, and modestly (compared with 2 partitions) for the three largest matrices. The increase in the number of iterations is again significantly reduced if dropping is applied. The elapsed times that are shown in Table 5.10 show several instances of superlinear speedup. This can be explained by the fact that the triangular factors computed by MA48 become smaller in size if the number of partitions is increased. The number of operations for a solve with a triangular matrix depends nonlinearly on the dimension of this matrix. In fact, its complexity is $\mathcal{O}(n^{1+\delta})$, where $\delta > 0$ is a problem and code dependent parameter.

6 Conclusion

In this paper we have described a method to construct efficient block diagonal preconditioners using a hypergraph partitioner and we have tested our method on a set of matrices from different application areas. Our method consists of three steps. In order to maximise the elements on the main diagonal we first scale and permute the matrix using the HSL routine MC64. This ensures that the diagonal blocks are structurally nonsingular. This preprocessing step is essential for the matrix `bayer01`, which comes

matrix	Number of iterations								
	1 partition	2 partitions		4 partitions		8 partitions		16 partitions	
		dropping		dropping		dropping		dropping	
		no	yes	no	yes	no	yes	no	yes
sherman2	17	27	17	47	17	72	19	95	19
jan99jac040	45	80	48	100	67	128	98	141	103
perrel	12	13	12	19	13	23	14	34	22
wang3	71	100	99	119	109	125	125	129	129
bayer01	164	148	186	146	181	169	192	198	221
venkat25	176	188	188	192	190	198	196	220	220

Table 5.7: Number of iterations of GMRES(50): ILU(0)

matrix	Number of iterations								
	1 partition	2 partitions		4 partitions		8 partitions		16 partitions	
		dropping		dropping		dropping		dropping	
		no	yes	no	yes	no	yes	no	yes
sherman2	1	20	4	43	4	69	11	92	11
jan99jac040	1	27	14	45	31	92	42	115	70
perrel	1	11	8	19	8	22	13	34	21
wang3	1	45	39	57	50	63	63	88	88
bayer01	1	13	10	14	10	16	15	23	26
venkat25	1	32	32	42	42	52	52	69	68

Table 5.8: Number of iterations of GMRES(50): MA48

matrix	elapsed times (sec)								
	1 partition	2 partitions		4 partitions		8 partitions		16 partitions	
		dropping		dropping		dropping		dropping	
		no	yes	no	yes	no	yes	no	yes
sherman2	0.02	0.03	0.03	0.07	0.02	0.05	0.07	0.10	0.04
jan99jac040	0.41	0.33	0.22	0.29	0.23	0.18	0.21	0.22	0.25
perrel	0.39	0.17	0.18	0.22	0.10	0.08	0.10	0.10	0.14
wang3	1.75	1.02	0.97	0.55	0.52	0.30	0.35	0.28	0.30
bayer01	12.09	4.43	5.14	1.37	1.56	0.75	0.93	0.46	0.56
venkat25	21.15	11.87	11.88	4.67	4.76	1.97	1.94	1.01	0.98

Table 5.9: Elapsed times of GMRES(50): ILU(0)

matrix	elapsed times (sec)								
	1 partition	2 partitions		4 partitions		8 partitions		16 partitions	
		dropping		dropping		dropping		dropping	
		no	yes	no	yes	no	yes	no	yes
sherman2	0.11	0.05	0.14	0.08	0.04	0.05	0.04	0.12	0.04
jan99jac040	1.18	0.20	0.35	0.17	0.11	0.16	0.18	0.20	0.21
perrel	14.10	4.14	4.17	1.22	1.06	0.40	0.41	0.22	0.19
wang3	141.92	21.57	23.91	4.11	4.46	1.00	0.98	0.40	0.45
bayer01	1.01	0.51	0.55	0.29	0.20	0.13	0.15	0.11	0.13
venkat25	63.54	25.93	26.22	10.09	12.10	4.26	3.95	1.87	1.58

Table 5.10: Elapsed times of GMRES(50): MA48

matrix	Number of iterations								
	1 partition	2 partitions		4 partitions		8 partitions		16 partitions	
		dropping		dropping		dropping		dropping	
		no	yes	no	yes	no	yes	no	yes
sherman2	17	27	17	47	17	61	19	77	19
jan99jac040	45	59	48	68	54	82	67	87	81
perrel	12	13	12	19	13	23	14	34	22
wang3	69	83	82	90	87	94	94	99	99
bayer01	90	93	93	94	94	98	98	111	111
venkat25	149	154	154	160	160	166	166	179	179

Table 5.11: Number of iterations of full GMRES: ILU(0)

from chemical process simulation. In order to maximise the size of the entries within the diagonal blocks, we apply a dropping strategy and partition the matrix using the same partitioning as obtained for this sparsified matrix. This strategy can considerably reduce the increase in the number of iterations if the number of partitions is increased, as is shown by our experimental results for the economic model `jan99jac040` and the chemical process simulation matrix `bayer01`. An undesirable side-effect of our dropping strategy, however, is that the communication volume is increased, since not all the entries in the matrix are taken into account by the partitioner. Finally we construct the preconditioner by decomposing the diagonal blocks into triangular factors. To this end we apply either the sparse direct solver MA48 or we make an incomplete ILU(0) decomposition.

The block diagonal preconditioners constructed from partitioning the equilibrated matrices give quite satisfactory results for the large matrices, especially the matrices from semiconductor device simulation and unstructured 2D Euler equations. Since the computational complexity of MA48 is $\mathcal{O}(n^{1+\delta})$, we gain more speedup for matrices `wang3` and `venkat25` when solving the preconditioning equation with MA48. Although the speedup for MA48 is higher than for ILU(0), the latter usually outperforms the former in term of total computing time.

Our approach for constructing an efficient block-diagonal preconditioner is quite flexible and can also be applied when other algorithms for scaling, partitioning or decomposition are preferred. Each of the algorithms in the chain can be replaced by another method of choice, so the same approach can be followed with MONDRIAAN or HMETIS instead of PATOH as the partitioner, or with ILU(t) to make an incomplete decomposition of the diagonal blocks.

Acknowledgements We thank Luc Giraud for his helpful advice in the early stages of this work and for his comments and suggestions on earlier versions of this paper. We thank Daniel Ruiz for interesting discussions on many aspects of this work.

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.

Axelsson, O. (1996), *Iterative solution methods*, Cambridge University Press, New York.

Çatalyürek, Ü. V. and Aykanat, C. (1995), 'Hypergraph model for mapping repeated sparse matrix-vector product computations onto multicomputers', *Proceedings of International Conference on High Performance Computing, HiPC'95, Goa, India.*

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

Çatalyürek, Ü. V. and Aykanat, C. (1999*b*), PaToH, a multilevel hypergraph partitioning tool for decomposing sparse matrices and partitioning vlsi circuits, Technical Report BU-CEIS-9902, Bilkent University.

Davis, T. A. (2004), 'University of Florida sparse matrix collection, `http://www.cise.ufl.edu/research/sparse/matrices/`'.

Duff, I. S. and Koster, J. (1999), 'The design and use of algorithms for permuting large entries to the diagonal of sparse matrices', *SIAM J. Matrix Analysis and Applications* **20**(4), 889–901.

Duff, I. S. and Koster, J. (2001), 'On algorithms for permuting large entries to the diagonal of a sparse matrix', *SIAM J. Matrix Analysis and Applications* **22**(4), 973–996.

Duff, I. S. and Reid, J. K. (1996), 'The design of MA48, a code for the direct solution of sparse unsymmetric linear systems of equations', *ACM Trans. Math. Softw.* **22**(2), 187–226.

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

Fiduccia, C. M. and Mattheyses, R. M. (1982), 'A linear-time heuristic for improving network partition', *Proc. 19th IEEE Design Automation Conference* pp. 175–181.

Foster, I. (1995), *Designing and Building Parallel Programs*, Addison-Wesley.

Frayssé, V., Giraud, L. and Kharraz-Aroussi, H. (1998), On the influence of the orthogonalization scheme on the parallel performance of GMRES, *in* D. Pritchard and J. Reeve, eds, 'EUROPAR'98 Parallel Processing', Vol. 1470, Springer, pp. 751–762.

Frayssé, V., Giraud, L., Gratton, S. and Langou, J. (2003), A set of GMRES routines for real and complex arithmetics on high performance computers, Technical Report TR/PA/03/03, CERFACS, Toulouse, France.

Giraud, L., Langou, J. and Rozložník, M. (2002), On the round-off error analysis of the Gram-Schmidt algorithm with reorthogonalization, Technical Report TR/PA/02/33, CERFACS, Toulouse, France.

Hendrickson, B. (1998), Graph partitioning and parallel solves: Has the emporer no clothes?, *in* A. Ferreira, ed., 'Solving Irregularly Structured Problems in Parallel, 5th International Symposium, Irregular '98, Berkeley, CA, 1998', Lecture Notes in Computer Science 1457, Springer Verlag, New York, pp. 218–225.

Hendrickson, B. and Kolda, T. G. (1998), Partitioning sparse rectangular matrices for parallel computations of Ax and $A^T x$, *in* B. Kågström, ed., 'Applied Parallel Computing in Large Scale Scientific and Industrial Problems, 4th International Workshop, PARA'98, Umeå, Sweden, 1998', Lecture Notes in Computer Science 1541, Springer Verlag, New York, pp. 239–247.

Hendrickson, B. and Kolda, T. G. (2000*a*), 'Graph partitioning models for parallel computing', *Parallel Computing* **26**, 1519–1534.

Hendrickson, B. and Kolda, T. G. (2000*b*), 'Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing', *SIAM J. Scientific Computing* **21**(6), 2048–2072.

HSL (2004), 'HSL 2004: A collection of Fortran codes for large scale scientific computation'. `www.cse.clrc.ac.uk/nag/hsl`.

Karypis, G. (1998), Multilevel hypergraph partitioning, Technical Report TR# 02-25, University of Minnesota.

Karypis, G. and Kumar, V. (1998), hMeTiS, A hypergraph partitioning package version, 1.5.3, Technical Report, University of Minnesota.

Kernighan, B. W. and Lin, S. (1970), 'An efficient heuristic procedure for partitioning graphs', *The Bell System Technical Journal* **49**, 291–307.

Olschowka, P. and Neumaier, A. (1996), 'A new pivoting strategy for gaussian elimination', *Linear Algebra and its Applications* **240**, 131–151.

Riyavong, S. (2003*a*), Experiments on parallel matrix-vector product, Working Notes WN/PA/03/127, CERFACS, Toulouse, France.

Riyavong, S. (2003*b*), Experiments on sparse matrix partitioning, Working Notes WN/PA/03/32, CERFACS, Toulouse, France.

Saad, Y. (1990), SPARSKIT: A basic tool kit for sparse matrix computations, Technical Report RIACS-90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA.

Saad, Y. (2003), *Iterative Methods for Sparse Linear Systems, 2nd edition*, SIAM, Philadelpha, PA.

Saad, Y. and Schultz, M. H. (1986), ‘GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems’, *SIAM J. Scientific and Statistical Computing* **7**, 856–869.

van der Vorst, H. (2003), *Iterative Krylov Methods for Large Linear systems*, Cambridge University Press, Cambridge.

Vastenhouw, B. and Bisseling, R. H. (2004), ‘A two-dimensional data distribution method for parallel sparse matrix-vector multiplication’, *SIAM Review*.