Available online at www.sciencedirect.com
SCIENCE DIRECT®
Parallel Computing 31 (2005) 275–289
PARALLEL COMPUTING
www.elsevier.com/locate/parco

Stabilized bordered block diagonal forms for parallel sparse solvers ☆

Iain S. Duff, Jennifer A. Scott *

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

Received 5 June 2004; revised 11 November 2004; accepted 29 December 2004
Available online 29 April 2005

Abstract

One possible approach to the solution of large sparse linear systems is to reorder the system matrix to bordered block diagonal form and then to solve the block system in parallel. We consider the duality between singly bordered and doubly bordered block diagonal forms. The idea of a stabilized doubly bordered block diagonal form is introduced. We show how a stable factorization of a singly bordered block diagonal matrix results in a stabilized doubly bordered block diagonal matrix. We propose using matrix stretching to generate a singly bordered form from a doubly bordered form. Matrix stretching is compared with two alternative methods for obtaining a singly bordered form and is shown to be efficient both in computation time and the quality of the resulting block structure.

Keywords: Large sparse linear systems; Unsymmetric matrices; Ordering; Partitioning; Bordered block diagonal form; Matrix stretching; Parallel processing

☆ This work was supported by the EPSRC grants GR/R46641 and GR/S42170.
* Corresponding author. Tel.: +44 1235 445131; fax: +44 1235 446626.
E-mail address: j.scott@rl.ac.uk (J.A. Scott).

0167-8191/$ - see front matter
doi:10.1016/j.parco.2004.12.008

1. Introduction

Many large scale scientific and engineering computations require the solution of sparse linear systems of equations. In most applications, it is important to solve these systems as accurately and as rapidly as possible. In recent years, as the size of the problems of interest has increased, parallel algorithms and parallel computers have become very important for the efficient solution of these problems. Solving linear systems in parallel involves distributing the data and the computation among the processors. If the problem is well structured this can often be done in a straightforward way but the irregular structure of general large sparse unsymmetric systems can make the distribution of the data and computation difficult. A good ordering of the rows and columns of the matrix can significantly reduce the storage and computation time required to factorize it in parallel. The efficient computation of good orderings that can be used to obtain stable factorizations of unsymmetric problems is a major objective of this paper.

We are concerned with direct methods for solving linear systems of the form

$$Ax = b, \tag{1}$$

where the $n \times n$ matrix $A = \{a_{ij}\}$ is large, sparse and unsymmetric. One possible approach to achieve coarse-grained parallelism is to preorder A to bordered block diagonal form. The block diagonal form leads to subproblems that can be solved independently, leaving an interface problem corresponding to the border that links the subproblems. The interface problem must be solved to complete the solution of the original problem. For the method to work well in a parallel environment, the order of the interface problem should be small compared with n so that there is little communication between the blocks on the diagonal and the interface problem and so that the cost of factorizing the interface problem is significantly less than that of factorizing the blocks on the diagonal. Recently, four solvers that implement direct algorithms based on this coarse-grained parallel approach have been developed for the mathematical software library HSL [13]. These include the frontal solver `HSL_MP43` [20] and `HSL_MP48` [10], which are both designed for solving sparse unstructured unsymmetric systems.

The matrix A is said to be in *doubly bordered block diagonal* (DBBD) form if the rows and columns have been permuted to the form

$$PAQ = A_{DB} = \begin{pmatrix} A_{11} & & & & C_1 \\ & A_{22} & & & C_2 \\ & & \ldots & & . \\ & & & A_{NN} & C_N \\ R_1 & R_2 & \ldots & R_N & E \end{pmatrix}, \tag{2}$$

where the blocks A_{ll} are $n_l \times n_l$ matrices and the border blocks C_l and R_l are $n_l \times p$ and $p \times n_l$ matrices, respectively, with $p \ll n_l$. This form is also sometimes referred to as an *arrowhead* form (see, for example, [11]). For finite-element applications, the DBBD form corresponds to partitioning the underlying finite-element domain

into non-overlapping subdomains; each block A_{ll}, $l = 1, 2, \ldots, N$, corresponds to the interior of a subdomain and the variables in the borders are those that lie on an interface between two or more subdomains. Each column of the column border $(C_1^{\mathrm{T}}, C_2^{\mathrm{T}}, \ldots, C_N^{\mathrm{T}}, E^{\mathrm{T}})^{\mathrm{T}}$ is called a *coupling* (or *linking*) column and each row of the row border $(R_1, R_2, \ldots, R_N, E)$ is a *coupling* (or *linking*) row. The aim is to permute A to DBBD form in such a way that the number p of coupling rows and columns is small, while at the same time ensuring a good balance between the sizes of the blocks A_{ll}, $l = 1, 2, \ldots, N$. Because we are interested in a coarse-grained parallel approach in which the diagonal blocks are factorized in parallel, we assume $N > 1$ and typically $N \leqslant 32$. In general, for larger values of N, p becomes unacceptably large and the solution of the interface problem then dominates the total solution time.

Having ordered A into DBBD form, we can attempt to apply a linear solver to the blocks A_{ll}. However, there are potential problems in doing this. In particular, the A_{ll} may be singular even if A is non-singular. It may be possible to use a direct solver that incorporates threshold-based partial pivoting for stability and moves rows and columns into the border as necessary. This is the approach adopted by the HSL parallel direct solver `HSL_MP42` [19], which is designed for the solution of linear systems from finite-element applications. Unfortunately, the border size may grow unacceptably large. To avoid this problem, we propose using an alternative approach in which A is ordered to *singly bordered block diagonal* (SBBD) form

$$PAQ = A_{SB} = \begin{pmatrix} A_{11} & & & C_1 \\ & A_{22} & & C_2 \\ & & \cdots & \cdot \\ & & A_{NN} & C_N \end{pmatrix}, \tag{3}$$

where the blocks A_{ll} are now rectangular $m_l \times n_l$ matrices with $m_l \geqslant n_l$, and the border blocks C_l are of order $m_l \times p$ ($p \ll n_l$). Again, the objective is to permute A into an SBBD form A_{SB} so that the number of coupling columns is as small as possible while maintaining well balanced blocks A_{ll}, $l = 1, 2, \ldots, N$. Ideally the A_{ll} should be well balanced in terms of factorization time, but at the very least we would plan that they have a similar dimension. With this form, a direct solver can be applied to factorize each of the rectangular blocks A_{ll}. We will show (see Section 2) that this in turn yields a *stabilized* DBBD form, that is, a DBBD form with non-singular blocks on the diagonal that can be stably factorized without the need to delay pivots. In practice, parallel direct solvers such as `HSL_MP43` and `HSL_MP48` do not form the stabilized DBBD form explicitly, but it can be used to provide a modified block Jacobi preconditioner for an iterative scheme on the whole system [8].

This paper is organised as follows. In Section 2, we look at how the SBBD form may be used with a direct solver to obtain a stabilized DBBD form. In Section 3, we discuss the reduction of unsymmetric sparse matrices to SBBD form and, in particular, we propose using a two-phase approach that is fast and yields a narrow border for a set of test problems arising from a range of practical applications that includes

both highly unsymmetric problems and problems with a (nearly) symmetric sparsity structure. Our findings are summarised in Section 4.

All numerical experiments presented in this paper were performed on a single processor of a dual processor Compaq DS20 Alpha workstation, with 3.6 GBytes of RAM. The Fortran codes were compiled using the Compaq Fortran 90 compiler with the optimization flag `-O`; C codes were compiled using the Compaq `cc` compiler with the flag `-O4`.

2. Stabilized DBBD forms

If we are factorizing a block A_{ll} of a doubly bordered block diagonal form (2) with threshold pivoting, then a potential pivot $a_{ij} \in A_{ll}$ must satisfy

$$|a_{ij}| \geqslant u * \max\{\max_k |(A_{ll})_{kj}|, \max_k |(R_l)_{kj}|\}, \tag{4}$$

where $u \in (0,1]$ is the threshold parameter. It is clear that large entries in the row border matrix R_l can prevent potential pivots from being selected. Thus if we want to maintain the doubly bordered structure, we would court instability by failing to satisfy the threshold test. In order to maintain stability, we must move the rows and columns that cannot be eliminated to the borders thus changing the structure of the doubly bordered form. This is exactly the situation that we would encounter if we obtained such a DBBD form from a domain decomposition algorithm unless the matrix had special numerical properties, for example was positive definite. The increase in the border size could adversely affect the a priori data structures, will increase the size of the interface problem, increase the work for the factorization, and reduce the potential for parallelism. Furthermore, not being able to choose a full set of pivots from the blocks on the diagonal (in the extreme case it is possible that no pivots can be stably chosen) may cause problems with load balancing.

If, however, we are factorizing blocks A_{ll} of a singly bordered block diagonal form (3) with threshold pivoting, then the potential pivot $a_{ij} \in A_{ll}$ must satisfy

$$|a_{ij}| \geqslant u * \max_k\{|(A_{ll})_{kj}|\}, \tag{5}$$

(again $u \in (0,1]$ is the threshold parameter) and so, if the matrix A is non-singular, there will always be a numerically satisfactory pivot in column j of A_{ll}. It is thus always possible to factorize the submatrix A_{ll} as the product of an $m_l \times n_l$ lower trapezoidal matrix $\bar{L}_l = \binom{L_l}{\tilde{L}_l}$ (where L_l is an $n_l \times n_l$ lower triangular matrix) and an $n_l \times n_l$ upper triangular matrix, U_l.

We can look at the effect of this factorization on the SBBD matrix by considering the factorization of the augmented system $(A_{ll} \quad C_l)$, which may be written in the form

$$(A_{ll} \quad C_l) = P_l \begin{pmatrix} L_l & \\ \tilde{L}_l & I \end{pmatrix} \begin{pmatrix} U_l & \tilde{U}_l \\ & S_l \end{pmatrix} \begin{pmatrix} Q_l & \\ & I \end{pmatrix}, \tag{6}$$

where P_l and Q_l are permutation matrices of order $m_l \times m_l$ and $n_l \times n_l$, respectively, and S_l is an $(m_l - n_l) \times p_l$ local Schur complement matrix. We note that pivots cannot be chosen from the columns of C_l because these columns have entries in at least one other border block C_j ($j \neq l$). Expressing the factorization in this way enables us to identify a doubly bordered block diagonal form, where the coupling columns are as in the SBBD form and the coupling rows comprise the rows in the $\widetilde{L}_l$ blocks. We observe that this is a special case of the DBBD form (2) in which the set of rows in each border block $R_l = \widetilde{L}_l$ is disjoint from the rows in the other border blocks $R_j = \widetilde{L}_j$ ($j \neq l$). This is part of the reason that the borders of this "stabilized" DBBD form are larger than for the DBBD form obtained directly using a graph partitioning tool such as the METIS package of Karypis and Kumar [16] (see www-users.cs.umn.edu/~karypis/metis/index.html). For example, METIS with $N = 4$ gives doubly bordered block diagonal forms with border sizes of 47 and 68 for problems `bcircuit` and `nopoly`, respectively (see Section 3.2 for details of our test problems). For the stabilized DBBD forms the corresponding border sizes are 94 and 136. However, the main point is that the DBBD form obtained via an SBBD form is stabilized in the sense that pivots can be stably chosen from within the blocks on the diagonal and no further alterations to the DBBD form will be necessary.

3. Ordering to SBBD form

The reduction of sparse unsymmetric matrices to SBBD form has been the subject of a number of recent papers (see, for example, [2,3,14] and the references therein). Hu, Maguire and Blake [14] developed the MONET (Matrix Ordering for minimal NET-cut) algorithm for ordering chemical process engineering problems to SBBD form. MONET is based on applying a multilevel approach to the weighted row connectivity graph of A [18]. Given the original matrix, a series of matrices are generated, each coarser than the preceding one and obtained by merging rows with similiar sparsity patterns. The coarsest matrix is bisected. This bisection is prolonged to the finer matrices and refined using the Kernighan–Lin algorithm [17]. Partitioning into more than two blocks is achieved by recursive bisection. The *net-cut* is the number of columns that lie in the border, that is, the number of coupling columns. The algorithm aims to minimise the net-cut while ensuring a good row balance (that is, each block A_{ll}, $l = 1, 2, \ldots, N$, has a similar number of rows).

HSL offers a Fortran 95 implementation of the MONET algorithm as routine `HSL_MC66`. This code was used by Duff and Scott [10] for preordering in their experiments with the parallel unsymmetric direct solvers `HSL_MP43` and `HSL_MP48`. They found that, for highly unsymmetric problems, such as those that arise in chemical process engineering applications, `HSL_MC66` produces high quality SBBD forms, that is, SBBD forms that are well balanced and have a narrow border. In particular, the numerical experiments reported by Duff and Scott showed that for up to 8 submatrices, the border typically represents less than 5% of the total number of columns. The narrow border and the balanced blocksize allows the parallel solvers to

achieve good speedups when run on a modest number of processors (experiments using up to $N = 16$ blocks on 16 processors were reported on). The main disadvantage of using the MONET algorithm is that it is relatively expensive in terms of CPU time. In general, the CPU cost of computing the singly bordered block form using HSL_MC66 was found to be significantly greater than the cost of the analyse phase of the direct solver applied to the blocks on the diagonal and, for some problems, it dominated the total solution time. Clearly, if a large number of matrices with the same sparsity pattern are to be factorized (such as when the linear solver is used within an iterative scheme for solving a nonlinear system), the ordering cost may be justified as it can be amortized over the repeated factorizations. But in some applications, only a single factorization is required and it may then be essential for the ordering to SBBD form to be performed rapidly so that it does not represent an unacceptable overhead. This is especially important if (as with HSL_MC66) the ordering is performed using a single processor.

The high cost of HSL_MC66 for large problems prompted Hu and Scott [15] to look at developing alternative algorithms that avoid using the row graph of A. Their algorithms are based on computing either a vertex separator or a wide separator of the symmetrized matrix $A + A^{\mathrm{T}}$. A graph partitioning tool such as METIS is used to partition the graph of $A + A^{\mathrm{T}}$ and a vertex separator is extracted from the output. This is optionally widened to a wide separator and used to partition the matrix. Hu and Scott report on a number of variants of the separator approach. In this study, we use the method termed SEP_VS(ND). This employs the multilevel nested dissection routine METIS_NodeND; the METIS output is used to compute an SBBD form using both a vertex separator and a wide separator approach and the one that gives the narrowest border is selected.

Numerical experiments presented in [15] show that for $N \leqslant 8$ the number of coupling columns obtained using the separator methods is generally competitive with results for the MONET algorithm. Furthermore, employing the separator methods is significantly faster than using the MONET algorithm. This makes them useful alternatives, particularly if the required number of factorizations of matrices having the same sparsity pattern is small.

3.1. Two-phase A to A_{SB} algorithm

The separator algorithms of Hu and Scott and the MONET algorithm are one-phase schemes, that is, the SBBD form A_{SB} is computed directly from A. An alternative approach that has been used for LP problems is a two-phase algorithm [2,11]. In the first phase, A is ordered to DBBD form A_{DB}; then, in the second phase, row stretching is used to obtain an SBBD form $\widehat{A}_{SB}$.

Stretching is a sparse matrix preprocessing technique that makes matrices sparser but, at the same time, larger. The idea was first introduced by Grcar [12], who proposed both row and column stretching as an effective way of treating sparse matrices with dense rows or columns before factorization. The technique has since been used by a number of authors, including Alvarado [1], Daydé, Deécamps and Gould [5], Ferris and Horn [11], and Aykannat, Pinar and Çatalyürek [2]. We use row stretch-

ing to associate with A_{DB} a larger square matrix $\widehat{A}_{SB}$ in SBBD form. As an illustration, consider the (5×5) 2-block DBBD matrix given by

$$A_{DB} = \begin{pmatrix} \times & \times & & & \\ \times & \times & & & \\ & & \times & \times & \\ & & \times & \times & \times \\ \times & & \times & & \times \end{pmatrix}. \tag{7}$$

The stretching process has two steps. In the first step, a rectangular matrix is constructed where the entries of the coupling rows (in this example, row 5 is the only coupling row) are split so that each new row has nonzero entries in only one non-border block:

$$\begin{pmatrix} \times & \times & & & \\ \times & \times & & & \\ & & \times & \times & \\ & & \times & \times & \times \\ \times & & & & \times \\ & & \times & & \end{pmatrix}. \tag{8}$$

The second step produces a square matrix by appending new columns to ensure the stretched matrix is structurally non-singular (provided A_{DB} was structurally non-singular). After reordering, the stretched matrix becomes:

$$\widehat{A}_{SB} = \begin{pmatrix} \times & \times & & & & \\ \times & \times & & & & \\ \times & & & & \times & +\sigma \\ & & \times & \times & & \\ & & \times & \times & \times & \\ & & \times & & & -\sigma \end{pmatrix}. \tag{9}$$

Here, $+\sigma$ and $-\sigma$ denote the added entries. More generally, suppose the row border $(R_1, R_2, \ldots, R_N, E)$ of the DBBD form A_{DB} (see (2)) has p rows and let row i be the first row in the border. Assume row i has nonzero entries in blocks $R_{i_1}, R_{i_2}, \ldots, R_{i_k}$ and E $(k \geqslant 2)$. We start the stretching by removing the entries in blocks $R_{i_2}, \ldots, R_{i_k}$ from row i and adding $k-1$ rows to A_{DB} such that row $n+l$ contains the nonzero entries from row i belonging to block $R_{i_{l+1}}, l = 1, \ldots, k-1$. Next, we append $k-1$ columns to the border of A_{DB} so that column $n+l$ has nonzero entries $\pm\sigma_{i_l}$ in rows i and $n+l$ $(l = 1, \ldots, k-1)$. Finally, we permute the rows of the stretched matrix so that row $n+l$ is permuted to be a row of the block $A_{\hat{i}\hat{i}}$, where $\hat{i} = i_{l+1}$. We now have a DBBD form $\widehat{A}_{DB}$ of order $(n+k-1) \times (n+k-1)$ with $p-1$ rows in its border. The process is repeated until finally the original DBBD has been stretched to a SBBD form $\widehat{A}_{SB}$.

This technique of adding rows (and corresponding columns) to make the matrix sparser is also called *row splitting* (see, for example, [11]). An analysis of the numerical properties of stretching is given by Grcar [12]. Assuming the original matrix A is well scaled, the added entries $\pm\sigma_{i_l}$ should be chosen to ensure $\widehat{A}_{SB}$ is also well scaled. Typical values that are used are ± 1.

Note that when solving the system $A_{DB}x = b$, the right-hand side b must also be stretched. In our 5×5 example, if $b = (b_1, b_2, b_3, b_4, b_5)^{\mathrm{T}}$, after stretching and reordering the system to be solved is $\widehat{A}_{SB}\hat{x} = \hat{b}$, where $\hat{x} = (x_1, x_2, x_3, x_4, x_5, y_1)^{\mathrm{T}}$ and $\hat{b} = (b_1, b_2, c_1, b_3, b_4, c_2)^{\mathrm{T}}$. Here y_1 is the new 'stretch' variable and any numbers c_1, c_2 that sum to b_5 can be chosen. Different choices give different values to the new variable y_1 but the original variables are unchanged. This is discussed further in [12].

3.2. One-phase and two-phase results for permuting A to SBBD form

In this section, we present numerical results comparing the performance of the MONET algorithm, the separator algorithm SEP_VS(ND) of Hu and Scott, and the two-phase approach based on row splitting. Our test problems are listed in Table 1. For the coarse-grained parallel approach to be efficient, the test problems need to be reasonably large and so our selected examples are all of order at least 10,000. A † indicates that the problem is included in the University of Florida Sparse Matrix Collection [4]. The remaining problems were supplied by Mark Stadtherr of the University of Notre Dame and Tony Garrett of AspenTech, UK.

The *symmetry index* $s(A)$ of a matrix A is defined to be the number of matched nonzero off-diagonal entries (that is, the number of nonzero entries a_{ij}, $i \neq j$, for which a_{ji} is also nonzero) divided by the total number nz of off-diagonal nonzero entries. Small values of $s(A)$ indicate the matrix is far from symmetric while values close to 1 indicate an almost symmetric sparsity pattern. In Table 1, the test matrices are listed in order of increasing symmetry index. A number of the matrices are highly unsymmetric and have a large number of zero diagonal entries. For such problems, matching orderings generally increase the symmetry index of the resulting reordered matrix (see, for example, Duff and Koster [9]). The HSL routine `MC21` uses a relatively simple algorithm to compute a matching that corresponds to a row permutation of A that puts nonzeros entries onto the diagonal, without considering the numerical values (for details see Duff [6,7]). The numbers in column 5 of Table 1 are the symmetry indices after applying `MC21` to the problems with $s(A) \leqslant 0.9$. We see that, in general, $s(A)$ is increased by applying `MC21` and for some of the highly unsymmetric problems (including `Matrix32406`, `lhr34c` and `lhr71c`) the increase is substantial. There is no guarantee that `MC21` will increase $s(A)$ and, for a number of the problems with a relatively large symmetry index, we find that $s(PA)$ (where P is the permutation returned by `MC21`) is smaller than $s(A)$. These problems have either no zero diagonal entries or a very small number of zeros on the diagonal (for example, `circuit_4` has only 21 zeros on the diagonal) and in such cases changes to $s(A)$ resulting from using `MC21` are largely due to the tie-breaking within `MC21`.

Table 1
Test problems

Identifier	n	nz	$s(A)$	$s(PA)$	Description/Application area
Matrix35640	35640	146880	0.0001	0.0427	Chemical process engineering
bayer01†	57735	277774	0.0002	0.0719	Chemical process engineering
icomp	75724	338711	0.0010	0.0025	Chemical process engineering
Matrix32406	32406	1035989	0.0014	0.2643	Chemical process engineering
lhr34c†	35152	764014	0.0015	0.3294	Chemical process engineering
bayer04†	20545	159082	0.0016	0.0694	Chemical process engineering
lhr71c†	70304	1528092	0.0016	0.3541	Chemical process engineering
poli_large†	15575	33074	0.0035	0.0035	Account of capital links
4cols	11770	43668	0.0159	0.0419	Chemical process engineering
l0cols	29496	109588	0.0167	0.0471	Chemical process engineering
onetone2†	36057	227628	0.1129	0.3600	Harmonic balance method
ethylene-l	10673	80904	0.2973	0.2441	Chemical process engineering
ethylene-2	10353	78004	0.3020	0.2487	Chemical process engineering
circuit_3†	12127	48137	0.7701	0.4093	Circuit simulation
circuit_4†	80209	79566	0.8292	0.7894	Circuit simulation
Zhao2†	33861	166453	0.9225		Electromagnetics
scircuit†	170998	958936	0.9999		Circuit simulation
hcircuit†	105676	513072	0.9999		Circuit simulation
bcircuit†	68902	375558	1.0000		Circuit simulation
garon2†	13535	390607	1.0000		2D Navier Stokes
pesa†	11738	79566	1.0000		Unknown
venkat50†	62424	1717792	1.0000		2D Euler
nopoly†	10774	70842	1.0000		Unknown

n, nz denote the order of A and the number of matrix entries, respectively. $s(A)$ denotes the symmetry index and $s(PA)$ is the symmetry index after ordering with MC21. Problems marked † are available from the University of Florida Sparse Matrix Collection.

For the test problems in the top half of the table with an unsymmetric sparsity pattern up to and including problem onetone2, we let $B = PA$ and for the remaining problems we set $B = A$. Hu and Scott [15] report that if A has a small symmetry index the vertex separator methods yield narrower borders when applied to B instead of A; for highly unsymmetric problems with many zeros on the diagonal, the reduction in border size by preordering with MC21 can be substantial. Thus, in the remainder of this section, the separator methods are applied to B (and, when used, the time for MC21 is included in the time needed to compute the SBBD form). Note that the MONET algorithm is applied to A because it is designed particularly for highly unsymmetric problems; experiments have shown that applying it to B generally results in wider borders.

For the stretching methods, in the first phase we must permute A to DBBD form A_{DB}. For symmetrically structured matrices this is done by computing a vertex separator V_s of the adjacency graph of A; the coupling rows and columns of A_{DB} correspond to the vertices belonging to V_s. In our experiments we use the METIS package and, in particular, the multilevel nested dissection routine METIS_NodeND. We note that minor modifications were needed in order to extract the vertex separator information. The output from the modified METIS_NodeND is an array part of

length n. If `part(i) = 0` then vertex $i \in V_s$; otherwise, if `part(i) = k` ($1 \leqslant k \leqslant N$), vertex i belongs to the kth block of A_{DB}.

For problems with an unsymmetric sparsity pattern we report the results of applying `METIS_NodeND` to the graph of the matrix $A + A^{\mathrm{T}}$ and to the bipartite graph of A. When applied to $A + A^{\mathrm{T}}$, if vertex $i \in V_s$ then the ith row and column of A are moved to the row and column borders, respectively.

The bipartite graph of A consists of two distinct sets $\mathscr{R}$ and $\mathscr{C}$ of n vertices each and undirected edges joining the vertices in $\mathscr{R}$ with those in $\mathscr{C}$. The set $\mathscr{R}$ is associated with the rows of A and the set $\mathscr{C}$ with the columns. There is an edge (r_i,c_j) if and only if $r_i \in \mathscr{R}$, $c_j \in \mathscr{C}$ and a_{ij} is nonzero. It is straightforward to show that the bipartite graph is equivalent to the adjacency graph of the $2n \times 2n$ matrix

$$\begin{pmatrix} 0 & A \\ A^{\mathrm{T}} & 0 \end{pmatrix}. \tag{10}$$

`METIS_NodeND` is applied to this graph. If vertex $i \leqslant n$ belongs to V_s then row i of A is permuted to the row border, while if $i > n$ belongs to V_s, column $j = i - n$ is permuted to the column border.

Once A has been permuted to a DBBD form A_{DB}, stretching is used to obtain an SBBD form $\widehat{A}_{SB}$ of order $\hat{n} \times \hat{n}$. We are concerned that the difference between $\hat{n}$ and n (the size of the original matrix A) should be small relative to n. This is important because we now have a larger (stretched) linear system of the form

$$\widehat{A}_{SB}\hat{x} = \hat{b} \tag{11}$$

to solve in place of the original system (1). In Table 2, n is compared with $\hat{n}$ for each of our test examples with $N = 8$ blocks. The size of $\hat{n}$ depends on the method used to permute A to DBBD form. The results of applying `METIS_NodeND` to $A + A^{\mathrm{T}}$ ($\hat{n}_1$), to the graph of $B + B^{\mathrm{T}}$ ($\hat{n}_2$) and to the bipartite graph ($\hat{n}_3$) are given in Table 2. A blank entry in the $\hat{n}_2$ column indicates the problem was not preordered using `MC21` (that is, $B = A$), and there is a blank in the $\hat{n}_3$ column for the symmetric matrices because the bipartite graph is not used for these problems. For the problems with a symmetric pattern (problems `bcircuit` onwards), the results are for applying METIS to A. The figures in parentheses are the percentage increases in the order of the system. Comparing the $\hat{n}_1$ and $\hat{n}_2$ columns, we see that for the highly unsymmetric problems the number of rows and columns added during the stretching process is significantly reduced by preordering using `MC21`. If we compare the use of $B + B^{\mathrm{T}}$ with that of the bipartite graph, we note that, in general, the former results in a slightly smaller stretched matrix. However, for each problem in the test set and $N = 8$, both approaches lead to an increase in the matrix size of less than 6%. Further results for a subset of our test problems run with $N = 4$ and $N = 16$ are given in Table 3. In each case, METIS is applied to $B + B^{\mathrm{T}}$ (with $B = A$ for the matrices below problem `onetone2`). We see that as N increases so too does the order $\hat{n}$ of the stretched matrix but the increases are generally modest. In particular, with $N = 16$ the increase in matrix size is less than 10% for all our test problems (including those not reported on in Table 3) and for most problems the increase is less than 3%. These results are

Table 2
The order of the stretched matrix $\hat{A}_{SB}$ ($N = 8$)

Identifier	n	$\hat{n}_1$	$\hat{n}_2$	$\hat{n}_3$
Matrix35640	35640	50361 (41.3)	35981 (0.96)	36605 (2.72)
bayer01	57735	61940 (7.28)	57916 (0.31)	57963 (0.39)
icomp	75724	75907 (0.24)	75864 (0.18)	75868 (0.19)
Matrix32406	32406	35823 (10.5)	33212 (2.48)	33241 (2.57)
lhr34c	35152	40662 (15.7)	35558 (1.15)	35670 (1.47)
bayer04	20545	23159 (12.7)	20770 (1.09)	20846 (1.46)
lhr71c	70304	74296 (5.68)	70764 (0.65)	70690 (0.55)
poli_large	15575	15927 (2.26)	15927 (2.26)	16059 (3.10)
4cols	11770	11857 (7.39)	11841 (0.60)	11865 (0.81)
l0cols	29496	29622 (0.43)	29602 (0.36)	29621 (0.42)
onetone2	36057	36377 (0.89)	36680 (1.72)	36458 (1.11)
ethylene-l	10673	10722 (0.46)		10730 (0.53)
ethylene-2	10353	10435 (0.79)		10437 (0.81)
circuit_3	12127	12197 (0.33)		12212 (0.70)
circuit_4	80209	81131 (1.15)		80924 (0.89)
Zhao2	33861	35696 (5.42)		35506 (4.85)
scircuit	170998	171327 (0.19)		171293 (0.17)
hcircuit	105676	105900 (0.21)		105937 (0.25)
bcircuit	68902	69099 (0.26)		
garon2	13535	14302 (5.67)		
pesa	11738	11934 (1.67)		
venkat50	62424	63544 (1.79)		
nopoly	10774	10925 (1.40)		

The figures in parentheses are the percentage increases in the order of the system.

Table 3
The order of the stretched matrix $\hat{A}_{SB}$ for different N

Identifier	n	$\hat{n}$		
		$N = 4$	$N = 8$	$N = 16$
Matrix35640	35640	35811 (0.48)	35981 (0.96)	36692 (2.95)
bayer01	57735	57829 (0.16)	57916 (0.31)	58022 (0.50)
lhr71c	70304	70411 (0.15)	70764 (0.65)	71129 (1.17)
l0cols	29496	29534 (0.13)	29602 (0.36)	29688 (0.65)
onetone2	36057	36313 (0.73)	36680 (1.72)	36940 (2.45)
circuit_4	80209	80677 (0.58)	81131 (1.15)	81877 (2.08)
Zhao2	33861	34761 (2.66)	35696 (5.42)	37153 (9.72)
venkat50	62424	63096 (1.08)	63544 (1.79)	64356 (3.09)
nopoly	10774	10842 (0.63)	10925 (1.40)	11011 (2.20)

The figures in parentheses are the percentage increases in the order of the system.

encouraging since such a small increase is unlikely to add a significant overhead to the time needed to solve the linear system.

We now compare the border sizes of the two-phase approach with those obtained using the HSL_MC66 implementation of the MONET algorithm and the separator method SEP_VS(ND) of Hu and Scott. Default setting are used for all HSL_MC66

control parameters. For the two-phase approach we apply METIS to the graph of $B + B^T$ and, for matrices with an unsymmetric sparsity pattern, to the bipartite graph. Results are given in Table 4 for $N = 8$. We highlight in bold the narrowest borders and those that are within 5% of the best. The results show that the two-phase approach works well, both for the problems with an unsymmetric structure and those that are symmetric. In particular, for the (nearly) symmetric problems it consistently outperforms the SEP_VS(ND) method, and there are only a small number of the highly unsymmetric problems for which HSL_MC66 produces significantly smaller borders. For the circuit simulation problems, the two-phase approach succeeds in producing much narrower borders than the one-phase algorithms. For problem circuit_4, both SEP_VS(ND) and HSL_MC66 give borders that are too wide for the coarse-grained parallel approach to work well; the solution time will be dominated by the time needed to solve the interface problem.

Comparing the two versions of the two-phase approach, using the bipartite graph for the unsymmetric problems seems to produce slightly narrower borders than using the graph of $B + B^T$. However, the reordering times given in Table 5 show that using the bipartite graph is generally the more expensive approach. This is because it involves applying METIS to a graph with $2n$ vertices while the graph of $B + B^T$

Table 4
The size of the border in the 8-block SBBD form computed using the SEP_VS(ND) method, HSL_MC66, and the two-phase approach

Identifier	n	SEP_VS(ND)	HSL_MC66	Two-phase	
				$B + B^T$	Bipartite
Matrix35640	35640	1599	**1367**	1514	1476
bayer01	57735	458	**254**	431	385
icomp	75724	332	**229**	**239**	409
Matrix32406	32406	1756	3514	1886	**1469**
lhr34c	35152	941	**792**	1062	**767**
bayer04	20545	404	542	547	**367**
lhr71c	70304	883	990	1163	**832**
poli_large	15575	2200	713	**473**	**492**
4cols	11770	263	233	**204**	268
l0cols	29496	313	**279**	**263**	361
onetone2	36057	1596	1745	1812	**1310**
ethylene-1	10673	190	217	215	**168**
ethylene-2	10353	201	217	**95**	191
circuit_3	12127	2015	1920	**114**	143
circuit_4	80209	19305	16417	1107	**962**
Zhao2	33861	3132	**2773**	3209	3036
scircuit	170998	1274	4353	**581**	649
hcircuit	105676	2350	2138	370	**154**
bcircuit	68902	495	951	**394**	
garon2	13535	2043	2308	**1516**	
pesa	11738	469	446	**391**	
venkat50	62424	2492	2536	**2232**	
nopoly	10774	310	337	**302**	

Table 5
The CPU time (in seconds) to compute the 8-block SBBD form using the SEP_VS(ND) method, HSL_MC66, and the two-phase approach

Identifier	SEP_VS(ND)	HSL_MC66	Two-phase	
			$B + B^T$	Bipartite
Matrix35640	1.57	6.93	1.54	2.02
bayer01	2.06	6.33	2.25	2.19
icomp	1.64	5.71	1.67	2.82
Matrix32406	5.27	324	5.45	9.48
lhr34c	3.12	9.64	3.30	3.47
bayer04	0.85	3.70	0.86	1.09
lhr71c	7.58	19.3	7.51	7.16
poli_large	0.22	0.21	0.23	0.30
4cols	0.66	0.91	0.66	0.85
l0cols	2.14	2.45	2.12	2.96
onetone2	1.35	8.12	1.51	1.34
ethylene-1	0.25	2.59	0.25	0.38
ethylene-2	0.24	3.07	0.21	0.35
circuit_3	0.20	51.6	0.20	0.35
circuit_4	3.29	2121	3.74	6.46
Zhao2	0.65	2.44	0.65	1.61
scircuit	2.83	54.7	2.70	7.78
hcircuit	1.23	16.1	1.19	2.91
bcircuit	0.95	5.34	0.97	
garon2	0.27	2.35	0.34	
pesa	0.14	0.75	0.14	
venkat50	0.82	6.14	1.26	
nopoly	0.12	0.68	0.12	

has only n vertices. The difference in CPU times is significantly less for the unsymmetric problems that are preordered for the $B + B^T$ approach using MC21 since employing MC21 incurs an overhead. There is little difference between the SEP_VS(ND) and two-phase $B + B^T$ times and both are significantly faster than HSL_MC66.

Finally, we consider the row imbalance. We define the row imbalance to be the difference between the maximum submatrix row dimension and the average submatrix row dimension, divided by the average submatrix row dimension, expressed as a percentage. That is, if m_l is the row dimension of A_{ll}, then

$$\text{row imbalance} = \frac{\max\{m_l\} - n/N}{n/N} \times 100.$$

For our stretching approach, the row imbalance can increase significantly with N. For example, for problem bayer04 the row balance is 4.2% for $N = 4$, 8.8% for $N = 8$ and 13.3% for $N = 16$. For HSL_MC66, the row imbalance increases more slowly with N and for our test examples, the row imbalance was less than 5% for $N \leqslant 16$. We remark that HSL_MC66 has a parameter that may be set by the user to control the size of the row imbalance and so, if the user is willing to allow a greater

imbalance, it may be possible to obtain a narrower border. We also note that the real metrics for load balancing should be the amount of work and time required by the factorization and this is not necessarily proportional to the submatrix row dimension.

We conclude that the two-phase approach is very successful at efficiently computing SBBD forms with a narrow border. The method replaces the original linear system by a larger one, but in our tests the increase in the order of the linear system was modest. However, we may need to investigate in the future whether it is possible to improve the row imbalance as the number of blocks is increased.

4. Concluding remarks

We have shown a duality between doubly and singly bordered block diagonal forms and, on the one hand, have used this in an efficient algorithm to produce an SBBD form and, on the other hand, have used a stable factorization of the SBBD form to define a stabilized DBBD form.

The stabilized form can either be used to effect a direct solution of the system of equations, see for example Duff and Scott [10], or as the basis for a block iterative technique, either using a Schur complement approach or a variant of block Jacobi [8].

We have also shown how matrix stretching can be used to generate a singly bordered form from a doubly bordered one. Numerical results have been presented for this approach and compared with using the MONET algorithm (`HSL_MC66`) and with methods based on vertex separators. The results show that matrix stretching is significantly faster than MONET and leads to only a modest increase in the order of the matrix. Moreover, matrix stretching produces border sizes that are competitive with MONET for highly unsymmetric problems and are significantly narrower for problems with a (nearly) symmetric sparsity pattern.

Acknowledgment

We are grateful to Yifan Hu of Wolfram Research for modifying `METIS_NodeND` to output vertex separator information and for commenting on a draft of this report. We would also like to thank our colleague John Reid and two anonymous referees for helpful and constructive comments.

References

[1] F.L. Alvarado, Matrix enlarging methods and their application, BIT 37 (1997) 473–505.

[2] C. Aykanat, A. Pinar, Ü.V. Çatalyürek, Permuting sparse rectangular matrices into singly-bordered block-diagonal form for parallel solution of LP problems, Technical Report BU-CE-0203, Computer Engineering Department, Bilkent Univeristy, 2002.

[3] K.V. Camarda, M.A. Stadtherr, Frontal solvers for process engineering: local row ordering strategies, Comput. Chem. Eng. 22 (1998) 333–341.
[4] T. Davis, University of Florida Sparse Matrix Collection. *NA Digest*, 97(23), 1997. Full details from www.cise.ufl.edu/research/sparse/matrices/.
[5] M.J. Daydé, J.P. Décamps, N.I.M Gould, Solution of unassembled linear systems using block stretching: preliminary experiments, Technical Report TR/APO/97/3, ENSEEIHT-IRIT, Toulouse, 1997.
[6] I.S. Duff, Algorithm 575, permutations for a zero-free diagonal, ACM Trans. Math. Software 7 (1981) 387–390.
[7] I.S. Duff, On algorithms for obtaining a maximum transversal, ACM Trans. Math. Software 7 (1981) 315–330.
[8] I.S Duff, G.H. Golub, F. Kwok, J.A. Scott, Combining direct and iterative methods to solve partitioned linear systems, Technical Report, Rutherford Appleton Laboratory, in press. Presented by Kwok at SIAM Meeting on Parallel Processing, San Francisco, February 2004.
[9] I.S. Duff, J. Koster, The design and use of algorithms for permuting large entries to the diagonal of sparse matrices, SIAM J. Matrix Anal. Appl. 20 (1999) 889–901.
[10] I.S. Duff, J.A. Scott, A parallel direct solver for large sparse highly unsymmetric linear systems, ACM Trans. Math. Software 30 (2004) 95–117.
[11] M.C. Ferris, J.D. Horn, Partitioning mathematical programs for parallel solution, Math. Program. 80 (1998) 35–62.
[12] J.F. Grcar, Matrix stretching for linear equations, Technical Report SAND90-8723, Sandia National Laboratories, 1990.
[13] HSL, A collection of Fortran codes for large-scale scientific computation, 2004. See http://hsl.rl.ac.uk/.
[14] Y.F. Hu, K.C.F. Maguire, R.J. Blake, A multilevel unsymmetric matrix ordering for parallel process simulation, Comput. Chem. Eng. 23 (2000) 1631–1647.
[15] Y.F. Hu, J.A. Scott, Ordering techniques for singly bordered block diagonal forms for unsymmetric parallel sparse direct solvers, Technical Report RAL-TR-2003-020, Rutherford Appleton Laboratory, 2003.
[16] G. Karypis, V. Kumar, METIS: A software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices—version 4.0, 1998.
[17] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell Syst. Technol. J. 49 (1970) 291–308.
[18] B.H. Mayoh, A graph technique for inverting certain matrices, Math. Comput. 19 (1965) 644–646.
[19] J.A. Scott, A parallel solver for finite element applications, Int. J. Numer. Meth. Eng. 50 (2001) 1131–1141.
[20] J.A. Scott, Two-stage ordering for unsymmetric parallel row-by-row frontal solvers, Comput. Chem. Eng. 25 (2001) 323–332.