NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2005; **12**:877–894
Published online 10 March 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nla.427

Ordering techniques for singly bordered block diagonal forms for unsymmetric parallel sparse direct solvers

Yifan Hu[1,‡] and Jennifer Scott[2,*,†]

[1]*Wolfram Research Inc., 100 Trade Center Drive, Champaign, IL 46530, U.S.A.*
[2]*Computational Science and Engineering Department, Rutherford Appleton Laboratory, Chilton, Oxfordshire OX11 0QX, U.K.*

SUMMARY

The solution of large sparse linear systems of equations is one of the cornerstones of scientific computation. In many applications it is important to be able to solve these systems as rapidly as possible. One approach for very large problems is to reorder the system matrix to bordered block diagonal form and then to solve the block system using a coarse-grained parallel approach. In this paper, we consider the problem of efficiently ordering unsymmetric systems to singly bordered block diagonal form. Algorithms such as the MONET algorithm of Hu *et al.* (Comput. Chem. Eng. 23 (2000) 1631) that depend upon computing a representation of AA^{T} can be prohibitively expensive when a single (or small number of) matrix factorizations are required. We therefore work with the graph of $A^{\mathrm{T}}+A$ (or $B^{\mathrm{T}}+B$, where B is a row permutation of A) and propose new reordering algorithms that use only vertex separators and wide separators of this graph. Numerical experiments demonstrate that our methods are efficient and can produce bordered forms that are competitive with those obtained using MONET. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: large sparse linear systems; unsymmetric matrices; ordering; parallel processing

1. INTRODUCTION

One possible approach to the problem of rapidly solving very large sparse $n \times n$ linear systems of equations

$$Ax = b$$

*Correspondence to: Jennifer Scott, Computational Science and Engineering Department, Rutherford Appleton Laboratory, Chilton, Oxfordshire OX11 0QX, U.K.
†E-mail: j.a.scott@rl.ac.uk
‡E-mail: yifanhu@wolfram.com

Contract/grant sponsor: EPSRC; contract/grant number: GR/R46641

Received 20 October 2003
Revised 31 August 2004
Accepted 15 September 2004

is to partition A into a number of loosely connected blocks and then apply an efficient sparse direct solver to the blocks in parallel. Solving the interface problem that connects the blocks completes the solution. The recent solvers `HSL_MP43` [1] and `HSL_MP48` [2] from the HSL mathematical software library [3] employ this technique for solving unsymmetric systems. To use these codes, the matrix A must be preordered to singly bordered block diagonal (SBBD) form

$$\begin{pmatrix} A_{11} & & & & C_1 \\ & A_{22} & & & C_2 \\ & & \cdots & & \cdot \\ & & & A_{NN} & C_N \end{pmatrix} \tag{1}$$

where, for $l = 1, 2, \ldots, n$, the rectangular blocks on the diagonal A_{ll} are $m_l \times n_l$ matrices with $m_l \geqslant n_l$, and the border blocks C_l are $m_l \times k$ with $k \ll n_l$.

`HSL_MP43` uses the frontal solver `MA42` [4] to perform a partial LU factorization of the diagonal blocks, while `HSL_MP48` employs a modified version of the well-known general purpose sparse direct solver `MA48` [5]. The interface problem is solved on a single processor using `MA42` and `MA48`, respectively. Experimentation on a number of different parallel platforms has demonstrated that both `HSL_MP43` and `HSL_MP48` can be significantly faster than their serial counterparts when the number N of blocks is small (typically in the range 4–16). In particular, the codes have been used to successfully solve highly unsymmetric linear systems arising from chemical process engineering applications [1, 2]. However, the effectiveness of the approach is dependent upon being able to obtain efficiently a SBBD form in which the blocks are of a similar size and, most importantly, the number of columns k in the border is small compared to n, the order of A. This is because the interface problem, which is generally denser than the original matrix, is currently solved using a serial solver and so it needs to be small enough that its solution does not cause a bottleneck within the parallel solver. The ordering to SBBD form itself is also performed sequentially.

The problem of reordering chemical process engineering problems to SBBD form has been addressed by the MONET algorithm of Hu *et al.* [6]. HSL offers an implementation of the MONET algorithm as routine `HSL_MC66`. This code was used by Duff and Scott for preordering in their experiments with `HSL_MP43` and `HSL_MP48`. They found that, for highly unsymmetric problems, `HSL_MC66` produces well-balanced SBBD forms (each submatrix A_{ll}, $l = 1, 2, \ldots, N$, has a similar number of rows) and, for up to 8 submatrices, the border typically represents less than 5% of the total number of columns. However, in terms of CPU time, the MONET algorithm is relatively expensive. In general, the CPU cost of ordering A to SBBD form was found to be significantly greater than the cost of the analyse phase of the direct solver on the diagonal blocks and, for some problems, it can dominate the total solution time. Clearly, if a large number of matrices with the same sparsity pattern are to be factorized, 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.

A key reason why the MONET algorithm is expensive is because it works with the sparsity pattern of AA^{T}. More precisely, it applies a multilevel recursive bisection algorithm combined with Kernighan–Lin refinement to the row graph $\mathcal{G}_{AA^{\mathrm{T}}}$ of A. Computing and working with the pattern of AA^{T} is costly, because AA^{T} may contain many more non-zero entries than A, particularly when A contains one or more relatively dense columns. Thus we would like to derive a cheaper algorithm that avoids computing AA^{T} (and $A^{\mathrm{T}}A$) but produces SBBD forms of similar quality to those obtained using the MONET algorithm.

Our starting point is the recent paper of Brainman and Toledo [7] on ordering the columns of sparse unsymmetric matrices to reduce fill-in during sparse LU factorizations with partial pivoting. George and Ng [8] showed that for every row permutation P, the fill of the LU factors of PA is essentially contained in the fill of the Cholesky factor of $A^{\mathrm{T}}A$. Furthermore, for a large class of matrices, for every entry in the Cholesky factor of $A^{\mathrm{T}}A$ there is a pivot sequence P that causes that entry of U to be non-zero [9]. Thus, unsymmetric direct solvers are often used to preorder the columns of A using a permutation Q that attempts to reduce the fill in the Cholesky factor of $Q^{\mathrm{T}}A^{\mathrm{T}}AQ$. The main challenge is to find a fill-minimizing permutation without computing $A^{\mathrm{T}}A$ or its sparsity pattern. One approach to this problem is the column approximate minimum degree ordering algorithm (COLAMD) of Davis *et al.* [10]. Brainman and Toledo propose adopting an earlier idea of Gilbert and Schreiber [11]. Their method finds vertex separators in $\mathcal{G}_{A^{\mathrm{T}}A}$ by finding *wide separators* in $\mathcal{G}_{A^{\mathrm{T}}+A}$. They present some encouraging results which motivated us to consider whether a similar approach might be used to order matrices A with an unsymmetric sparsity pattern to SBBD form more rapidly than the MONET algorithm.

This paper is organized as follows. In Section 2, we introduce the test problems and computing environment used for our numerical experiments. Basic concepts from graph theory and some results on SBBD forms and wide separators are recalled in Section 3. In Section 4, we consider how SBBD forms may be generated via wide separators in $\mathcal{G}_{A^{\mathrm{T}}+A}$ then, in Section 5, we propose computing SBBD forms directly from the vertex separators in $\mathcal{G}_{A^{\mathrm{T}}+A}$. Vertex separators are computed using the graph partitioning routine `METIS_PartGraphRecursive` from the well-known METIS package [12] and, in Section 6, using the nested dissection routine `METIS_NodeND`. Numerical results compare the proposed approaches with the MONET algorithm. These show that the new algorithms are significantly faster than MONET and, for up to 8 blocks, for many of our test problems we obtain orderings that are competitive in quality with the MONET orderings. The new orderings and the MONET orderings are used in Section 7 with the parallel solver `HSL_MP48`. We find that the overall cost of reordering and then solving the linear system is often less for the new algorithms.

2. TEST PROBLEMS AND COMPUTING ENVIRONMENT

In this section, we introduce the test problems that will be used throughout this paper to illustrate the performance of the ordering algorithms. For the coarse-grained parallel approach to be efficient, the test problems need to be reasonably large and so we have selected problems that are all of order at least 10 000. In Table I $*$ (asterisk) indicates that the problem is included in the University of Florida Sparse Matrix Collection [13]. The remaining problems were supplied by Mark Stadtherr of the University of Notre Dame and Tony Garrett of AspenTech, U.K. The *symmetry index* $s(A)$ of a matrix A is defined to be the number of matched non-zero off-diagonal entries (that is, the number of non-zero entries a_{ij}, $i \neq j$, for which a_{ji} is also non-zero) divided by the total number of off-diagonal non-zero entries. Small values of $s(A)$ indicate the matrix is far from symmetric while values close to 1 indicate an

Table I. Test problems.

Identifier	n	nz	$s(A)$	Description/application area
`Matrix35640`	35 640	146 880	0.0001	Chemical process engineering
`bayer01`*	57 735	277 774	0.0002	Chemical process engineering
`icomp`	75 724	338 711	0.0010	Chemical process engineering
`Matrix32406`	32 406	1035 989	0.0014	Chemical process engineering
`lhr34c`*	35 152	764 014	0.0015	Chemical process engineering
`bayer04`*	20 545	159 082	0.0016	Chemical process engineering
`lhr71c`*	70 304	1528 092	0.0016	Chemical process engineering
`poli_large`*	15 575	33 074	0.0035	Account of capital links
`4cols`	11 770	43 668	0.0159	Chemical process engineering
`10cols`	29 496	109 588	0.0167	Chemical process engineering
`onetone2`*	36 057	227 628	0.1129	Harmonic balance method
`ethylene-1`	10 673	80 904	0.2973	Chemical process engineering
`ethylene-2`	10 353	78 004	0.3020	Chemical process engineering
`Zhao2`*	33 861	166 453	0.9225	Electromagnetics
`scircuit`*	170 998	958 936	0.9999	Circuit simulation
`hcircuit`*	105 676	513 072	0.9999	Circuit simulation
`bcircuit`*	68 902	375 558	1.0000	Circuit simulation
`garon2`*	13 535	390 607	1.0000	2D Navier–Stokes
`pesa`*	11 738	79 566	1.0000	Unknown
`wang3`*	26 064	177 168	1.0000	3D diode semiconductor device

n, nz denote the order of the system and the number of matrix entries, respectively. $s(A)$ denotes the symmetry index. Problems marked $*$ are available from the University of Florida Sparse Matrix Collection.

almost symmetric sparsity pattern. The test matrices are listed in order of increasing symmetry index.

Note that a significant proportion of our test problems originate from chemical process simulation. We chose these because they have a highly unsymmetric sparsity pattern and it is for such problems that the HSL parallel solvers `HSL_MP43` and `HSL_MP48` and the ordering routine `HSL_MC66` are primarily designed. We have, however, also included a number of problems from a variety of other application areas, many of which have a greater degree of symmetry. In particular, the problems towards the end of the table have unsymmetric values but have a symmetric (or nearly symmetric) sparsity structure.

All numerical experiments presented in this paper were performed on a dual processor Compaq DS20 Alpha server, 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`. Default settings were used for all `HSL_MC66`, `HSL_MP48`, and METIS control parameters.

3. GRAPHS AND SEPARATORS

3.1. Graph notation and definitions

It is convenient to recall some basic concepts from graph theory.

A *graph* $\mathcal{G}$ is defined to be a pair (V, E), where V is a finite set of *vertices* $v_1, v_2, \ldots, v_n$, and E is a set of *edges*, where an edge is a pair (v_i, v_j) of distinct vertices of V. If no distinction is made between (v_i, v_j) and (v_j, v_i) the graph is *undirected.* An *ordering* (or *labelling*) of a graph $\mathcal{G}$ with n vertices is a bijection of $\{1,2,\ldots,n\}$ onto V. Two vertices v_i and v_j in V are said to be *adjacent* (or *neighbours*) if $(v_i, v_j) \in E$. The edge (v_i, v_j) is *incident* to vertex v_i and to vertex v_j. A *path of length* k in $\mathcal{G}$ is an ordered set of distinct vertices $(v_{i_1}, v_{i_2}, \ldots, v_{i_{k+1}})$ where $(v_{i_j}, v_{i_{j+1}}) \in E$ for $1 \leqslant j \leqslant k$. Two vertices are *connected* if there is a path joining them. An undirected graph $\mathcal{G}$ is *connected* if each pair of distinct vertices is connected. Otherwise, $\mathcal{G}$ is disconnected and consists of two or more *connected components.*

A labelled graph $\mathcal{G}(A)$ with n vertices can be associated with any square matrix $A = \{a_{ij}\}$ of order n. Two vertices i and j $(i \neq j)$ are adjacent in the graph if and only if a_{ij} is non-zero. If A has a symmetric sparsity pattern, $\mathcal{G}(A)$ is undirected.

Row and column graphs were first introduced by Mayoh [14]. The *column graph* $\mathcal{G}_{A^{\mathrm{T}}A}$ of A is defined to be the undirected graph of the symmetric matrix $A^{\mathrm{T}} * A$, where $*$ denotes matrix multiplication without taking cancellations into account (so that, if an entry is zero as a result of numerical cancellation, it is considered as a non-zero entry and the corresponding edge is included in the column graph). The vertices of $\mathcal{G}_{A^{\mathrm{T}}A}$ are the columns of A and two columns i and j $(i \neq j)$ are adjacent if and only if there is at least one row k of A for which a_{ki} and a_{kj} are both non-zero. The *row graph* $\mathcal{G}_{AA^{\mathrm{T}}}$ is defined analogously as the undirected graph of $A * A^{\mathrm{T}}$. Column (row) permutations of A correspond to relabelling the vertices of the column (row) graph.

A subset $E_{\mathrm{s}} \subset E$ of edges of an undirected graph $\mathcal{G} = (V, E)$ is an *edge separator* if removing E_{s} leaves $\mathcal{G}$ disconnected. Edges in the edge separator are called the *cut edges*. A subset $S \subset V$ of vertices is a *vertex separator* (or *attachment set*) if the removal of S and its incident edges disconnects an otherwise connected graph or connected component.

A *vertex cover* of a graph $\mathcal{G} = (V, E)$ is a subset of V, such that each edge in E is incident to at least one vertex in the cover. The minimum vertex cover is the smallest such cover.

3.2. Separators and SBBD forms

Mayoh [14] showed that, given a vertex separator in the column graph $\mathcal{G}_{A^{\mathrm{T}}A}$, the matrix can be reordered to SBBD form. Suppose S is a vertex separator in $\mathcal{G}_{A^{\mathrm{T}}A}$ and let $VC_1, VC_2, \ldots, VC_N$ be the subsets of columns of A that correspond to the N components of $\mathcal{G}_{A^{\mathrm{T}}A}$ once S and its incident edges have been removed. Then each row of A has non-zero entries in columns of at most one VC_i and thus the columns of A can be ordered into SBBD form as follows:

1. All the columns in the VC_i are ordered before the columns corresponding to the vertices in S.
2. For $i < j$, all the columns in VC_i are ordered before the columns in VC_j.
3. For $i < j$, a row with a non-zero entry in a column of VC_i is ordered ahead of any row with a non-zero entry in a column of VC_j.

Thus, if we have a vertex separator in $\mathcal{G}_{A^{\mathrm{T}}A}$, we can reorder A to the required form. However, as already noted, computing $A^{\mathrm{T}}A$ is expensive and we would like to find a vertex separator without forming $A^{\mathrm{T}}A$ or its sparsity pattern. One possible approach is to use wide separators, a term coined by Gilbert and Schreiber [11]. If $V_1, V_2, \ldots, V_N$ are the subsets of the vertices V corresponding to the N components of $\mathcal{G} = (V, E)$ after the removal of the vertex separator S and its incident edges, any path between $i \in V_k$ and $j \in V_l$ $(k \neq l)$ must pass through at least one vertex in S. A vertex set is a *wide separator* if every path between $i \in V_k$ and $j \in V_l$ passes through a sequence of two vertices in S (one after the other along the path).

Brainman and Toledo [7] give the following result.

Theorem 1
A wide separator in $\mathcal{G}_{A^{\mathrm{T}}+A}$ is a vertex separator in $\mathcal{G}_{A^{\mathrm{T}}A}$.

Moreover, if A is symmetric they also show the converse result.

Theorem 2
If A has a symmetric sparsity pattern with no zeros on the diagonal, then a vertex separator in $\mathcal{G}_{A^{\mathrm{T}}A}$ is a wide separator in $\mathcal{G}_{A^{\mathrm{T}}+A}$.

4. COMPUTING SBBDS VIA WIDE SEPARATORS

Theorem 1 provides a means of computing a vertex separator in $\mathcal{G}_{A^{\mathrm{T}}A}$ without forming $A^{\mathrm{T}}A$; the problem is reduced to computing a wide separator in the undirected graph $\mathcal{G}_{A^{\mathrm{T}}+A}$. If we have an edge separator E_{s}, a wide separator can be found by choosing the endpoints of each edge in E_{s}. Alternatively, a wide separator may be found by widening a vertex separator S.

In our numerical experiments, edge separators are computed using the well-known graph partitioning code METIS of Karypis and Kumar [12] (see www-users.cs.umn.edu/~karypis/metis/index.html). In particular, we use the routine `METIS_PartGraphRecursive` to partition $\mathcal{G}_{A^{\mathrm{T}}+A}$ into N parts using a multilevel recursive bisection algorithm. The objective of this partitioning is to minimize the number of edges that are cut by the partitioning. Vertex separators may be extracted from the METIS output using Dulmage–Mendelsohn-type decompositions ([15]; see also Pothen and Fan [16]). Essentially, once an edge separator has been computed, the bipartite graph induced by the cut edges is generated. A vertex separator then corresponds to a minimum vertex cover in this bipartite graph (see, for example Ashcraft and Liu [17] and the references therein).

The software that we use to postprocess the METIS output was provided by Mirek Tůma of the Academy of Sciences of the Czech Republic. Assuming $N = 2^k$ for some k, the Tůma code is run k times, each time generating a minimum cover for the bipartite graph given by the METIS edge separators. These k cover sets are then unified to give the required vertex separator. For example, consider $k = 3$. Denoting the partition numbers by $000, 001, 010, 011, \ldots, 111$, three separate vertex covers are computed based on the difference in individual bits in their binary representations:

Partitions 000, 001, 010, 011 versus 100, 101, 110, 111.
Partitions 000, 001, 100, 101 versus 010, 011, 110, 111.
Partitions 000, 110, 010, 100 versus 001, 011, 101, 111.

Unifying these three vertex covers yields a vertex separator for the partitioned graph with 8 partitions.

Having obtained a vertex separator S in $\mathcal{G}_{A^{\mathrm{T}}+A}$, we need to widen it to a wide separator W_{s}. Suppose the subset $S \subset V$ of vertices of an undirected graph $\mathcal{G} = (V, E)$ is a vertex separator such that the removal of S and its incident edges breaks the graph into two components $\mathcal{G}_1 = (V_1, E_1)$ and $\mathcal{G}_2 = (V_2, E_2)$. It is clear that the sets $W_1 = S \cup \{i | i \in V_1, (i,j) \in E$

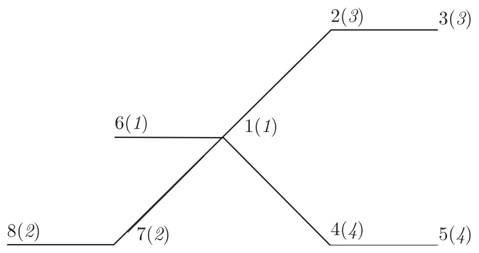

Figure 1. Figure illustrating that the union of the wide separators of bisections may not be a wide separator with regard to the overall partition of the graph. The numbers in brackets are the subgraph to which the vertex belongs after recursive bisection.

for some $j \in S\}$ and $W_2 = S \cup \{i | i \in V_2, (i, j) \in E$ for some $j \in S\}$ are wide separators in $\mathcal{G}$. In other words, the vertex separator may be widened by adding to it all the vertices that are adjacent to the separator in one of the subgraphs. In their paper, Brainman and Toledo [7] select the smaller of W_1 and W_2 as their wide separator. We have performed experiments using this choice but, in an attempt to obtain a smaller wide separator (and hence an SBBD form with a narrower border), we propose the following method which widens S by adding vertices from both V_1 and V_2.

Algorithm. Wide separator

```
Initialize $W_s \Leftarrow S$
For each vertex $j \in S$
    Let $n_1$ (respectively, $n_2$) be the number of neighbours of $j$ that belong to $V_1$ (respectively,
      $V_2$) but not to $W_s$.
    If $n_1 \leqslant n_2$ set $W_s \Leftarrow W_s \cup \{i | i \in V_1, i \notin W_s, (i, j) \in E\}$;
      otherwise set $W_s \Leftarrow W_s \cup \{i | i \in V_2, i \notin W_s, (i, j) \in E\}$.
```

Thus for each $j \in S$ we add up how many neighbours it has belonging to V_1 that are not already in W_s and, similarly, how many belong to V_2 but not to W_s. We then add to the set of vertices W_s the smaller of these two sets of neighbours. There is no guarantee that the final wide separator computed in this way will be smaller than that obtained using the simpler method of Brainman and Toledo but, as we shall see in our numerical experiments, in general this approach does yield SBBD forms with narrower borders.

We note that when either the above algorithm or the Brainman and Toledo algorithm is applied recursively to the subgraphs of bisections, the union of the wide separators of each of the bisections need not be a true wide separator of the original graph. For example, consider the graph $\mathcal{G}$ with eight vertices given in Figure 1. Assume that the first bisection gives two subgraphs $\mathcal{G}_1$ and $\mathcal{G}_2$, with $\mathcal{G}_1$ having vertices $\{1, 6, 7, 8\}$, and $\mathcal{G}_2$ vertices $\{2, 3, 4, 5\}$. A wide separator for this bisection is $WS_0 = \{1, 6, 7\}$. $\mathcal{G}_1$ and $\mathcal{G}_2$ are then bisected again so that each vertex belongs to one of the four subgraphs, as show by numbers in brackets in Figure 1. A wide separator for the bisection of $\mathcal{G}_1$ is $WS_1 = \{1, 6\}$, and the bisection of $\mathcal{G}_2$ has a null wide separator $WS_2 = \{\}$. However, the union of the three wide separators, $W_s = WS_0 \cup WS_1 \cup WS_2 = \{1, 6, 7\}$, is not a true wide separator with regard to the quadrasection, because there is a path from vertex 2 (in domain 3) to vertex 4 (in domain 4), passing through only one vertex (vertex 1) in W_s. This, however, does not happen very often in practice and

Table II. The size of the border in the 8-block SBBD form computed using wide separators in $\mathcal{G}_{A^{\mathrm{T}}+A}$.

			Method		
Identifier	n	$\|S\|$	I	II	III
`Matrix35640`	35 640	19 888	33 599	**29 998**	**29 920**
`bayer01`	57 735	12 264	19 342	18 609	**17 280**
`icomp`	75 724	289	427	**401**	441
`Matrix32406`	32 406	13 357	19 836	18 166	**16 989**
`lhr34c`	35 152	11 943	24 432	21 394	**19 376**
`bayer04`	20 545	6630	10 200	9939	**9242**
`lhr71c`	70 304	12 171	23 799	20 296	**18 864**
`poli_large`	15 575	199	**665**	1386	695
`4cols`	11 770	305	524	**498**	**477**
`10cols`	29 496	343	536	536	**485**
`onetone2`	36 057	1981	3256	3897	**2352**
`ethylene-1`	10 673	308	705	737	**628**
`ethylene-2`	10 353	302	641	678	**535**
`Zhao2`	33 861	1435	**3051**	**3059**	**3014**
`scircuit`	170 998	449	**1297**	2071	**1292**
`hcircuit`	105 676	509	**1482**	4219	2595
`bcircuit`	68 902	279	**632**	702	**631**
`garon2`	13 535	756	**2059**	**2059**	**2059**
`pesa`	11 738	213	**438**	**438**	**445**
`wang3`	26 064	2544	**5069**	**4912**	**4904**

$|S|$ denotes the size of the vertex separator.

can be easily remedied when ordering to SBBD form, by bringing the few offending columns into the border.

In Table II we give the size of the border in the 8-block SBBD form obtained by computing the wide separator in $\mathcal{G}_{A^{\mathrm{T}}+A}$ using METIS followed by the three approaches discussed above, namely:

I: choose both endpoints of each edge in the edge separator E_s,
II: the method of Brainman and Toledo [7],
III: the above wide separator algorithm.

The smallest border (and those within 5% of the smallest) are highlighted in bold.

For comparison, the size $|S|$ of the vertex separator computed using `METIS_PartGraph Recursive` and the Tüma software is given in column 3. We see that Method III usually produces narrower borders than Method II and, although there are a number of problems (notably `hcircuit`) for which Method I produces the narrowest border, Method III appears to be the best method overall. However, we also observe that for some of the very unsymmetric problems in the top half of the table, the size of the vertex separator and the border is large; in particular, for problems `Matrix35640` and `Matrix32406` the percentages of the columns lying in the border are 84 and 52%, respectively. For these problems, the border is too large for the coarse-grained parallel approach discussed in Section 1 to be effective.

Theorem 2 tells us that for a symmetric matrix A, there is a one to one correspondence between vertex separators in $\mathcal{G}_{A^{\mathrm{T}}A}$ and wide separators in $\mathcal{G}_{A^{\mathrm{T}}+A}$. Therefore we would expect that if A can be preordered to a more symmetric matrix B, then vertex separators in $\mathcal{G}_{A^{\mathrm{T}}A}$ should be 'captured' better by wide separators in $\mathcal{G}_{B^{\mathrm{T}}+B}$ (note that row and column permutations do not change the graph $\mathcal{G}_{A^{\mathrm{T}}A}$, other than vertex renumbering.) This leads us to consider preordering A in an attempt to increase the symmetry index prior to ordering to SBBD form.

4.1. Preordering using maximal matchings

It is well-known that matching orderings can increase the symmetry index of the resulting reordered matrix, particularly in cases where A is very sparse with a large number of zeros on the diagonal (see, for example Duff and Koster [18]). Permuting a large number of non-zero off-diagonal entries onto the diagonal reduces the number of unmatched non-zero off-diagonal entries, which in turn increases the symmetry index. Furthermore, if A is permuted to a matrix B with a non-zero diagonal, the following theorem proves that every vertex separator in $\mathcal{G}_{B^{\mathrm{T}}B}$ is a vertex separator in $\mathcal{G}_{B^{\mathrm{T}}+B}$.

Theorem 3
If B has no zeros on the diagonal, then a vertex separator in $\mathcal{G}_{B^{\mathrm{T}}B}$ is a vertex separator in $\mathcal{G}_{B^{\mathrm{T}}+B}$.

Proof
Let S be a vertex separator in $\mathcal{G}_{B^{\mathrm{T}}B}$ such that the removal of S and its incident edges breaks the graph into N components. Let $V_1, V_2, \ldots, V_N$ be the subsets of the vertices corresponding to the N components. Suppose for contradiction that S is not a separator in $\mathcal{G}_{B^{\mathrm{T}}+B}$. Then there exists a path in $\mathcal{G}_{B^{\mathrm{T}}+B}$ between $i_1 \in V_k$ and $j_1 \in V_l$ $(k \neq l)$ that does not pass through a vertex in S. There must be a pair of adjacent vertices i and j along the path such that $i \in V_k$ and $j \in V_l$. Since i and j are adjacent in $\mathcal{G}_{B^{\mathrm{T}}+B}$, the (i,j) entry of $B^{\mathrm{T}}+B$ is non-zero. Therefore, either $b_{ij} \neq 0$ or $b_{ji} \neq 0$. Since $b_{ii} \neq 0$ and $b_{jj} \neq 0$, it follows that $(B^{\mathrm{T}}B)_{ij} = \sum_q b_{qi}b_{qj}$ is non-zero. Thus (i,j) must an edge in $\mathcal{G}_{B^{\mathrm{T}}B}$ and $i \leftrightarrow j$ is a path in $\mathcal{G}_{B^{\mathrm{T}}B}$ that does not pass through S, a contradiction. $\square$

It can be shown by example that if B has an unsymmetric sparsity pattern with no zeros on the diagonal, a vertex separator in $\mathcal{G}_{B^{\mathrm{T}}B}$ is not necessarily a wide separator in $\mathcal{G}_{B^{\mathrm{T}}+B}$.

The HSL routine MC21 uses a relatively simple algorithm to compute a matching that corresponds to a row permutation of A that puts non-zeros entries onto the diagonal, without considering the numerical values. The method used by MC21 is a simple depth-first search with look-ahead; the algorithm is described by Duff [19, 20]. In Table III, the symmetry index of our test problems is given before and after reordering with MC21; we also give the number of zero diagonal entries before permuting the matrix (in each case, there are no zero entries on the diagonal after permuting). We omit the final four test examples from Table I because they have a symmetric sparsity pattern with no zeros on the diagonal so that MC21 is not needed (it returns the identity permutation).

In the remainder of our discussion, we let $B = PA$ be the permuted matrix after employing MC21. In Table IV, we show the size of the border in the 8-block SBBD form obtained by computing the wide separator in $\mathcal{G}_{B^{\mathrm{T}}+B}$ using Methods I, II and III. The best results (and those within 5% of the best) are highlighted. Again, the final 4 test problems are not included. We also omit poli_large and Zhao2 because MC21 returns the identity permutation for these two

Table III. Structural symmetry before and after permuting the rows of A using the MC21 ordering.

			Symmetry index	
Identifier	n	Diagonal zeros	Before	After
Matrix35640	35 640	35 639	0.0001	0.0427
bayer01	57 735	57 733	0.0002	0.0719
icomp	75 724	0	0.0010	0.0025
Matrix32406	32 406	32 366	0.0014	0.2643
lhr34c	35 152	35 050	0.0015	0.3294
bayer04	20 545	20 545	0.0016	0.0694
lhr71c	70 304	70 100	0.0016	0.3541
poli_large	15 575	0	0.0035	0.0035
4cols	11 770	0	0.0159	0.0419
10cols	29 496	0	0.0167	0.0471
onetone2	36 057	26 967	0.1129	0.3600
ethylene-1	10 673	0	0.2973	0.2441
ethylene-2	10 353	0	0.3020	0.2487
Zhao2	33 861	0	0.9225	0.9225
scircuit	170 998	84	0.9999	0.9995
hcircuit	105 676	48	0.9999	0.9852

Table IV. The size of the border in the 8-block SBBD form computed using wide separators in $\mathcal{G}_{B^{T}+B}$.

			Method		
Identifier	n	$\|S\|$	I	II	III
Matrix35640	35 640	1313	**1949**	2221	2038
bayer01	57 735	247	**437**	545	**432**
icomp	75 724	299	**411**	**427**	**412**
Matrix32406	32 406	1504	2470	3168	**2336**
lhr34c	35 152	769	1505	1959	**1346**
bayer04	20 545	390	**621**	**621**	**612**
lhr71c	70 304	918	1458	1772	**1378**
4cols	11 770	211	369	354	**294**
10cols	29 496	275	447	446	**384**
onetone2	36 057	1434	**2832**	3391	**2825**
ethylene-1	10 673	248	612	570	**484**
ethylene-2	10 353	239	565	513	**487**
scircuit	170 998	444	**1255**	1856	**1237**
hcircuit	105 676	458	**1051**	1361	**1052**

$|S|$ denotes the size of the vertex separator.

examples. For the remaining problems, we see that Method III remains the method of choice and, comparing the results with those in Table II, it is clear that preordering A can lead to a dramatic reduction in the border size. In particular, for lhr71c the border size is reduced from 18 864 to 1378 columns, and for Matrix35640 the percentage of border columns is cut from 84% to less than 6%.

Table V. The row differences for the 8-block SBBD form computed using wide separators in $\mathcal{G}_{A^{\mathrm{T}}+A}$ (denoted by 'Before') and $\mathcal{G}_{B^{\mathrm{T}}+B}$ (denoted by 'After').

	I		II		III	
Identifier	Before	After	Before	After	Before	After
`Matrix35640`	4.02	0.43	121.82	1.89	16.75	1.55
`bayer01`	8.81	0.17	27.18	0.26	9.52	0.26
`icomp`	0.17	0.13	0.17	0.15	0.16	0.21
`Matrix32406`	12.57	1.88	65.19	6.32	34.20	1.51
`lhr34c`	7.56	0.86	47.79	5.05	27.67	1.73
`bayer04`	22.04	0.39	43.42	0.78	22.59	0.66
`lhr71c`	6.57	0.32	12.31	0.80	5.99	0.30
`4cols`	0.68	0.61	1.29	1.16	2.04	1.70
`10cols`	0.52	0.46	0.52	1.06	1.00	0.92
`onetone2`	1.69	1.18	10.07	3.99	7.23	1.09
`ethylene-1`	2.62	2.40	4.65	3.60	2.32	1.72
`ethylene-2`	1.93	2.40	4.71	2.70	1.62	2.32
`scircuit`	0.03	0.03	0.43	0.14	0.11	0.17
`hcircuit`	0.14	0.03	2.01	0.26	0.99	0.70

For many of our test examples, preordering using `MC21` not only reduces the border size but also improves the row balance. For a given SBBD form, we define the percentage *row difference* to be

$$(m_{\max} - n/N)/(n/N) * 100$$

where N is the number of blocks and $m_{\max}$ is the largest number of rows in a block. Thus the row difference compares the size of the largest block with the average block size. A small row difference implies the blocks are of a similar size and this is what is meant by a good row balance. In Table V, we give the row differences for the 8-block SBBD forms computed using Methods I, II and III applied to $\mathcal{G}_{A^{\mathrm{T}}+A}$ and $\mathcal{G}_{B^{\mathrm{T}}+B}$. The columns labelled 'Before' are computed using wide separators in $\mathcal{G}_{A^{\mathrm{T}}+A}$ and those labelled 'After' use $\mathcal{G}_{B^{\mathrm{T}}+B}$. We see that, for a number of the highly unsymmetric problems, the row imbalance when wide separators are computed using $\mathcal{G}_{A^{\mathrm{T}}+A}$ is very poor, particularly if Method II (the Brainman and Toledo method) is used. But if we reorder using `MC21` the row balance improves significantly for these examples. In particular, the row difference for Method III is always less than 2.5%.

5. COMPUTING SBBDS WITHOUT COMPUTING WIDE SEPARATORS

The results in Tables IV and V are encouraging since, for many examples, we are now obtaining good row balance and border sizes that should not lead to the interface problem causing a significant bottleneck when the ordering is used with a parallel direct solver such as `HSL_MP43` or `HSL_MP48`. However, for some problems the wide separator is much larger than the vertex separator (given in column 3 of Tables II and IV). Since for matrices with a non-zero diagonal and unsymmetric sparsity pattern a vertex separator in $\mathcal{G}_{A^{\mathrm{T}}A}$ is not necessarily a wide separator in $\mathcal{G}_{A^\mathrm{T}+A}$ but, by Theorem 3, is a vertex separator in $\mathcal{G}_{A^\mathrm{T}+A}$, it may be advantageous to try and compute the SBBD directly from the vertex separator in $\mathcal{G}_{A^\mathrm{T}+A}$ (or $\mathcal{G}_{B^\mathrm{T}+B}$), without computing wide separators.

Suppose S is a vertex separator in $\mathcal{G}_{A^\mathrm{T}+A}$. Let $VC_1, VC_2, \ldots, VC_N$ be the subsets of columns of A that correspond to the N components of $\mathcal{G}_{A^\mathrm{T}+A}$ once S and its incident edges have been removed. Each row has to be assigned to a partition. We do this by considering the rows in turn and, for each row, examine the column indices of its non-zero entries. Let row_i and col_j denote the ith row and jth column of A. We first add up the number of entries $n_{i,k}$ in row_i that belong to the subset VC_k $(1 \leqslant k \leqslant N)$. If $n_{i,l} = \max\{n_{i,k} : 1 \leqslant k \leqslant N\}$, we assign row_i to the partition l. We then move all columns col_j in row_i that do not belong to VC_l into the set S. Once all the rows have been considered, the only rows that are still unassigned are those which have all their non-zero entries in S. Such rows are assigned equally to the N partitions. The final set S is the set of border columns. In this way, A is ordered into SBBD form.

If $\mathrm{block}(i)$ denotes the partition in the SBBD form to which row_i is assigned, the above algorithm can be summarized as follows.

Algorithm. SBBD_vertex separator

```
1. Set $S$ be a vertex separator in $\mathcal{G}_{A^\mathrm{T}+A}$. Initialize $\mathrm{block}(1:n) = 0$.
2. For each $\mathrm{row}_i$, consider the columns $\mathrm{col}_j$ of its non-zero entries;
   add up the number $n_{i,k}$ of entries belonging to each $VC_k$.
    If $n_{i,l} = \max\{n_{i,k} : 1 \leqslant k \leqslant N\}$ then
      set $\mathrm{block}(\mathrm{row}_i) = l$;
      remove all $\mathrm{col}_j \in VC_k$ for each $k \neq l$ from $VC_k$ and add to $S$.
3. Once all rows considered, assign any rows for which $\mathrm{block}(\mathrm{row}_i) = 0$ equally between
   the $N$ partitions.
4. If for some $k$, $\mathrm{col}_k \in S$ has non-zero entries only in rows belonging to partition $m$, $\mathrm{col}_k$
   is removed from $S$ and added to $VC_m$.
```

To limit the row imbalance, at step 2, row_i is only assigned to partition l if the number of rows in partition l is less than a given fraction of the total number of rows n. In our tests, we only allow a row to be assigned to partition l if either $n_{i,k} = 0$ for all $k \neq l$ or the total number of rows assigned to l is less than $\mathrm{Imbal} * n/N$, where we set the imbalance parameter Imbal to 1.2. If partition l is already too large, then $\mathrm{block}(\mathrm{row}_i)$ is set to l_1, where subset VC_{l_1} has the next largest number of entries in row_i.

In Table VI results are presented for this vertex separator method (which we refer to as Method VS) and are compared with the best wide separator method from Section 4 (Method III) and with HSL_MC66 (the HSL implementation of the MONET algorithm [6]). Results are given for 2, 4, and 8 partitions. We use MC21 to preorder the problems for which A has an unsymmetric structure prior to calling Methods III and VS but not before calling HSL_MC66. We do not preorder for HSL_MC66 because this code is designed particularly for highly unsymmetric problems and experiments using $B^\mathrm{T} + B$ generally led to wider borders.

Comparing the vertex separator approach (VS) with the wide separator approach (III) we see that, for the (nearly) symmetric problems there is little to choose between the two but

Table VI. The size of the border in the SBBD form computed using the wide separator Method III, the vertex separator Method VS, and `HSL_MC66`.

		Number of blocks								
		$N=2$			$N=4$			$N=8$		
Identifier	n	III	VS	`MC66`	III	VS	`MC66`	III	VS	`MC66`
`Matrix35640`	35 640	490	388	**344**	1008	965	**704**	2038	1682	**1367**
`bayer01`	57 735	90	**69**	71	234	169	**135**	432	296	**254**
`icomp`	75 724	41	**36**	55	213	171	**134**	412	325	**229**
`Matrix32406`	32 406	170	**134**	1215	1112	**894**	2539	2336	**1750**	3514
`lhr34c`	35 152	509	474	**94**	965	936	**354**	1346	1202	**792**
`bayer04`	20 545	**87**	**91**	182	306	**254**	369	612	**463**	542
`lhr71c`	70 304	514	**135**	198	775	706	**392**	1378	1290	**990**
`poli_large`	15 575	**303**	**301**	394	**568**	**543**	582	695	**652**	713
`4cols`	11 770	33	34	**30**	95	**70**	106	294	**217**	233
`10cols`	29 496	**32**	33	**30**	167	142	**123**	384	**290**	**279**
`onetone2`	36 057	469	**233**	254	1967	1333	**1204**	2825	**1797**	**1745**
`ethylene-1`	10 673	**38**	**38**	75	202	129	**111**	484	290	**217**
`ethylene-2`	10 353	**27**	**28**	50	151	**93**	133	487	293	**217**
`Zhao2`	33 861	666	665	**641**	1794	1790	**1688**	3014	2979	**2773**
`scircuit`	170 998	**58**	**58**	2551	**594**	**600**	3753	**1237**	**1230**	4353
`hcircuit`	105 676	**191**	**190**	591	**364**	**365**	891	**1052**	**1052**	2138
`bcircuit`	68 902	**3**	**3**	563	**142**	**141**	737	**631**	**618**	951
`garon2`	13 535	**542**	**542**	682	**1100**	**1100**	1543	**2059**	**2031**	2308
`pesa`	11 738	**81**	**79**	127	**192**	**192**	245	**445**	**444**	**446**
`wang3`	26 064	**1740**	**1740**	**1740**	**3355**	**3316**	**3310**	**4904**	**4863**	**4813**

for the unsymmetric examples in the top half of the table, the former generally yields the narrower borders. For both methods, the row imbalance with $N=8$ blocks was less than 2.5% for each of the test problems. Compared with `HSL_MC66`, we see that Method VS produces wider borders for some of the unsymmetric problems but for others the converse is true. For the symmetrically structured examples, Methods III and VS outperform `HSL_MC66`.

6. NESTED DISSECTION VERTEX SEPARATORS

As well as providing routines for partitioning graphs into equal parts, METIS has routines for computing fill-reducing orderings for sparse matrices. These use a multilevel nested dissection algorithm. The nested dissection algorithm is based on computing a vertex separator of the graph of the matrix. Thus an alternative approach for ordering A to SBBD form is to use the multilevel nested dissection routine `METIS_NodeND` to compute a vertex separator in $\mathcal{G}_{A^T+A}$ (or $\mathcal{G}_{B^T+B}$) and to either widen it using the wide separator algorithm of Section 4 or use it in the SBBD_vertex separator algorithm (see Section 5). We refer to these as Methods III(ND) and VS(ND), respectively. Results for 4 and 8 blocks are given in Table VII. Again, `MC21` is used to preorder the problems for which A has an unsymmetric structure prior to using Methods III(ND) and VS(ND). We remark that it was necessary to modify routine `METIS_NodeND` in order to extract the vertex separator information. The

Table VII. The size of the border in the SBBD form computed using the nested dissection-based methods III(ND) and VS(ND).

		Number of blocks			
		$N = 4$		$N = 8$	
Identifier	n	III(ND)	VS(ND)	III(ND)	VS(ND)
`Matrix35640`	35 640	857	**764**	1756	**1599**
`bayer01`	57 735	**328**	339	584	**458**
`icomp`	75 724	328	**268**	386	**332**
`Matrix32406`	32 406	**1103**	**1112**	1840	**1756**
`lhr34c`	35 152	442	**412**	1015	**941**
`bayer04`	20 545	216	**180**	530	**404**
`lhr71c`	70 304	398	**345**	1004	**883**
`poli_large`	15 575	1159	**1116**	2567	**2200**
`4cols`	11 770	91	**73**	311	**263**
`10cols`	29 496	128	**110**	**322**	**313**
`onetone2`	36 057	917	**740**	1980	**1596**
`ethylene-1`	10 673	95	**86**	322	**190**
`ethylene-2`	10 353	**76**	**74**	293	**201**
`Zhao2`	33 861	1858	**1720**	3392	**3132**
`scircuit`	170 998	1067	**769**	1751	**1274**
`hcircuit`	105 676	996	**815**	2574	**2350**
`bcircuit`	68 902	127	**75**	679	**495**
`garon2`	13 535	**1226**	**1219**	**2078**	**2050**
`pesa`	11 738	**191**	**181**	492	**471**
`wang3`	26 064	**3118**	**3137**	**4370**	**4376**

results show that, in general, the VS(ND) method produces narrower borders than the III(ND) method.

In Table VIII the border sizes for VS and VS(ND) are compared with those for `HSL_MC66`. We see that for some problems, including `bayer04` and the `ethylene` examples, using the nested dissection method leads to the narrowest borders. But for other problems (notably `poli_large` and `hcircuit`) nested dissection gives much poorer results. We also find that the row differences are significantly larger for the VS(ND) method. For example, for `bayer04` with 8 blocks, the row difference for Method VS(ND) is 12.4% compared with 0.5% for Method VS. Similarly, for `ethylene-2` the row differences are 15.5 and 1.7% for VS(ND) and VS, respectively. Thus the smaller borders appear to be at the cost of greater row imbalances.

7. TIMINGS AND `HSL_MP48` RESULTS

One of the main motivations for this study was the need to preorder matrices to SBBD more rapidly than using the `HSL_MC66` implementation of the MONET algorithm. In Table IX, we compare the run times for Methods VS and VS(ND) with those for `HSL_MC66`. For VS and VS(ND), the times include the METIS time and the time taken to run `MC21` and to permute A using the `MC21` ordering prior to computing the SBBD form. All timings are CPU times in seconds.

Table VIII. The size of the border in the SBBD form computed using the VS and VS(ND) methods, and HSL_MC66.

		Number of blocks					
		$N=4$			$N=8$		
Identifier	n	VS	VS(ND)	MC66	VS	VS(ND)	MC66
Matrix35640	35 640	965	764	**704**	1682	1599	**1367**
bayer01	57 735	169	268	**135**	296	458	**254**
icomp	75 724	171	264	**134**	325	332	**229**
Matrix32406	32 406	**894**	1112	2539	**1750**	**1756**	3514
lhr34c	35 152	936	412	**354**	1202	941	**792**
bayer04	20 545	254	**180**	369	463	**404**	542
lhr71c	70 304	706	**345**	392	1290	**993**	**990**
poli_large	15 575	**543**	1116	582	**652**	2200	713
4cols	11 770	**70**	**73**	106	**217**	263	**233**
10cols	29 496	142	**110**	123	**290**	313	**279**
onetone2	36 057	1333	**740**	1204	1797	**1596**	1745
ethylene-1	10 673	129	**86**	111	290	**190**	217
ethylene-2	10 353	93	**74**	133	293	**201**	217
Zhao2	33 861	1790	**1720**	**1688**	2979	3132	**2773**
scircuit	170 998	**600**	769	3753	**1230**	**1274**	4353
hcircuit	105 676	**365**	815	891	**1052**	2350	2138
bcircuit	68 902	141	**75**	737	618	**495**	951
garon2	13 535	**1100**	1219	1543	**2031**	**2050**	2308
pesa	11 738	192	**181**	245	**444**	471	**446**
wang3	26 064	3343	**3137**	3310	4863	**4376**	4813

On many problems, VS is more than twice as fast as VS(ND) and is significantly faster than HSL_MC66. In fact, for a number of examples, the HSL_MC66 timings are prohibitively expensive when compared with the times given in Table X for solving a single linear system once it is in SBBD form using the direct solver HSL_MP48. These timings, which are for a subset of our test problems with $N=8$ run on both processors of our Compaq DS20, are designed to illustrate the relative costs of reordering and solving the resulting linear system; further results for HSL_MP48 on a variety of computing platforms and different numbers of processors are presented in Duff and Scott [2]. The reported timings are elapsed times in seconds, measured using the MPI timer MPI_WTIME on the host processor.

The results in Table X demonstrate clearly the importance of having a narrow border. For those problems with a relatively wide border (including Matrix35640, scircuit and garon2) the time taken for analysing and factorizing the interface problem represents a significant proportion of the total solution time. If the number of processors is increased, this will result in a significant bottleneck and poor speed-ups. However, the results also show that our new approaches can be successful in obtaining good SBBD forms, that is, SBBD forms leading to a small interface problem and that are competitive with those found with the MONET algorithm. For a number of examples (such as the lhr problems), the HSL_MP48 time using the SBBD form computed by the VS method is greater than that reported for the HSL_MC66 SBBD form but, if the time required to compute the SBBD form is taken into consideration, several factorizations of matrices having the same pattern are needed to justify the extra cost of ordering using HSL_MC66.

Table IX. The times (in seconds) to compute the SBBD form using the VS and VS(ND) methods and `HSL_MC66`.

	Number of blocks								
	$N=2$			$N=4$			$N=8$		
Identifier	VS	VS(ND)	`MC66`	VS	VS(ND)	`MC66`	VS	VS(ND)	`MC66`
`Matrix35640`	0.43	1.88	2.02	0.55	1.77	4.31	0.67	1.62	6.93
`bayer01`	0.57	2.51	2.54	0.76	2.04	4.51	0.93	2.08	6.33
`icomp`	0.39	1.28	2.06	0.63	1.47	3.93	0.85	1.68	5.71
`Matrix32406`	1.25	5.15	98.8	1.75	5.61	237	2.20	5.44	324
`lhr34c`	0.70	3.34	3.38	0.97	3.56	6.49	1.22	3.23	9.64
`bayer04`	0.19	0.77	1.50	0.28	0.84	2.71	0.36	0.85	3.70
`lhr71c`	1.63	8.03	7.98	2.21	7.63	13.2	2.76	7.89	19.3
`poli_large`	0.08	0.12	0.08	0.14	0.17	0.14	0.20	0.21	0.21
`4cols`	0.07	0.95	0.34	0.09	0.87	0.64	0.13	0.66	0.91
`10cols`	0.18	3.07	0.89	0.26	2.67	1.64	0.34	2.15	2.45
`onetone2`	0.91	1.23	3.57	0.93	1.28	5.68	1.04	1.37	8.12
`ethylene-1`	0.07	0.20	1.51	0.11	0.22	1.94	0.15	0.26	2.59
`ethylene-2`	0.07	0.20	1.50	0.10	0.22	2.62	0.15	0.24	3.07
`Zhao2`	0.12	0.54	0.70	0.22	0.59	1.53	0.30	0.67	2.44
`scircuit`	0.95	2.26	26.6	1.44	2.61	44.7	1.83	2.97	54.7
`hcircuit`	0.49	0.88	6.06	0.75	1.04	11.2	1.01	1.29	16.1
`bcircuit`	0.24	0.68	2.03	0.40	0.79	3.82	0.55	0.97	5.34
`garon2`	0.12	0.14	0.80	0.21	0.20	1.54	0.29	0.25	2.35
`pesa`	0.04	0.09	0.25	0.06	0.11	0.48	0.09	0.13	0.75
`wang3`	0.10	0.40	0.80	0.18	0.46	1.52	0.25	0.52	2.37

Table X. Timings for ordering to SBBD form and then using `HSL_MP48` to solve a single linear system ($N=8$).

	VS			`MC66`		
Identifier	Ordering	`MP48`		Ordering	`MP48`	
`Matrix35640`	0.67	16.7	(5.01)	6.93	12.8	(4.80)
`bayer01`	0.93	1.58	(0.04)	6.33	1.70	(0.03)
`icomp`	0.85	0.48	(0.002)	5.71	0.41	(0.001)
`lhr34c`	1.22	10.8	(1.27)	9.64	7.63	(0.67)
`lhr71c`	2.76	26.3	(1.08)	19.3	22.5	(0.95)
`4cols`	0.13	0.15	(0.02)	0.91	0.15	(0.02)
`10cols`	0.34	0.61	(0.03)	2.45	0.55	(0.03)
`ethylene-2`	0.15	0.38	(0.01)	3.07	0.37	(0.01)
`scircuit`	1.83	18.4	(2.59)	54.7	43.1	(31.6)
`bcircuit`	0.55	2.93	(0.47)	5.34	2.89	(0.50)
`garon2`	0.29	29.4	(11.8)	2.35	25.6	(14.7)

The numbers in parentheses are the times for analysing and factorizing the interface problems.

All the reordering algorithms discussed in this paper are serial algorithms. But, even using the faster VS ordering algorithms, the time taken to reorder to SBBD form can still be greater than the time needed to solve the resulting linear system using `HSL_MP48`. Thus in the future we would like to develop reordering algorithms that can be run in parallel.

8. CONCLUDING REMARKS

New algorithms that avoid using either the row or column graph of the matrix have been proposed for ordering an unsymmetric matrix A to SBBD form. The new methods use either vertex separators or wide separators of the symmetrized matrix $A^{\mathrm{T}} + A$. In general, if A has a highly unsymmetric sparsity pattern with a large number of zeros on the diagonal, SBBD forms with better row balance and narrower borders are achieved by first applying a maximal matching ordering to A to improve its symmetry. For many of our test problems, the new methods are competitive with the existing MONET algorithm of Hu *et al.* [6]. In particular, for symmetrically structured examples, the separator methods generally lead to narrower border sizes. Furthermore, the new methods are much faster than the MONET algorithm. This makes them useful alternatives when the required number of factorizations of matrices having the same sparsity pattern is small because the overall cost of reordering A and then solving a single linear system (or small number of systems) using a parallel direct solver can be faster for the new algorithms than for MONET.

ACKNOWLEDGEMENTS

We are very grateful to Mirek Tüma of the Academy of Sciences of the Czech Republic for providing us with his code for computing vertex separators from the `METIS_PartGraphRecursive` output. Many thanks also to Iain Duff at the Rutherford Appleton Laboratory for helpful discussions and comments on a draft of this paper. The work of J. A. Scott was supported by the EPSRC Grant GR/R46641.

REFERENCES

1. Scott JA. The design of a portable parallel frontal solver for chemical process engineering problems. *Computers in Chemical Engineering* 2001; **25**:1699–1709.
2. Duff IS, Scott JA. A parallel direct solver for large sparse highly unsymmetric linear systems. *ACM Transactions on Mathematical Software* 2004; **30**:95–117.
3. HSL. A collection of Fortran codes for large scale scientific computation, 2002. Full details from http://www.cse.clrc.ac.uk/nag/hsl/
4. Duff IS, Scott JA. The design of a new frontal code for solving sparse unsymmetric systems. *ACM Transactions on Mathematical Software* 1996; **22**(1):30–45.
5. Duff IS, Reid JK. MA48, a Fortran code for direct solution of sparse unsymmetric linear systems of equations. *Report RAL-93-072*, Rutherford Appleton Laboratory, 1993.
6. Hu YF, Maguire KCF, Blake RJ. A multilevel unsymmetric matrix ordering for parallel process simulation. *Computers in Chemical Engineering* 2000; **23**:1631–1647.
7. Brainman I, Toledo S. Nested-dissection orderings for sparse LU with partial pivoting. *SIAM Journal on Matrix Analysis and Applications* 2002; **23**:998–1012.
8. George A, Ng E. On the complexity of sparse QR and LU factorization on finite-element matrices. *SIAM Journal on Scientific and Statistical Computing* 1988; **9**:849–861.
9. Gilbert JR, Ng E. Predicting structure in nonsymmetric sparse matrix factorizations. In *Graph Theory and Sparse Matrix Computation*, George A, Gilbert J, Liu J (eds). Springer: New York, 1993.
10. Davis TA, Gilbert JR, Larimore SR, Ng EG. A column approximate minimum degree ordering algorithm. *Technical Report TR-00-005*, Department of Computer and Information Science and Engineering, University of Florida.
11. Gilbert JR, Schreiber R. Nested dissection with partial pivoting. *Sparse Matrix Symposium 1982*: *Program and Abstracts*, Fairfield Glade, TN, 1982.
12. Karypis G, Kumar V. METIS: a software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices—version 4.0, 1998.
13. Davis T. University of Florida Sparse Matrix Collection. *NA Digest* 1997; **97**(23). Full details from http://www.cise.ufl.edu/∼davis/sparse/
14. Mayoh BH. A graph technique for inverting certain matrices. *Mathematics of Computation* 1965; **19**:644–646.
15. Dulmage A, Mendelsohn N. Coverings of bipartite graphs. *Canadian Journal of Mathematics* 1958; **10**:517–534.

16. Pothen A, Fan CJ. Computing the block triangular form of a sparse matrix. *ACM Transactions on Mathematical Software* 1990; **16**:303–324.
17. Ashcraft C, Liu JWH. Applications of the Dulmage–Mendelsohn decomposition and network flow to graph bisection improvement. *SIAM Journal on Matrix Analysis and Applications* 1998; **19**:325–354.
18. Duff IS, Koster J. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM Journal on Matrix Analysis and Applications* 1999; **20**:889–901.
19. Duff IS. Algorithm 575, permutations for a zero-free diagonal. *ACM Transactions on Mathematical Software* 1981; **7**:387–390.
20. Duff IS. On algorithms for obtaining a maximum transversal. *ACM Transactions on Mathematical Software* 1981; **7**:315–330.