**RAL-TR-2003-020**

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

Yifan Hu[1] and Jennifer A. Scott[2,3,4]

**Abstract**

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 in parallel. 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, Maguire and Blake (2000) 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 $A^T + A$ (or $B + B^T$, 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.

**Keywords:** large sparse linear systems, unsymmetric matrices, ordering, parallel processing.

---

[1] Wolfram Research, Inc., 100 Trade Center Drive, Champaign, IL61820, USA.
Email: yifanhu@wolfram.com

[2] Computational Science and Engineering Department, Rutherford Appleton Laboratory, Chilton, Oxfordshire, OX11 0QX, England, UK.
Email: j.a.scott@rl.ac.uk

[3] Current reports available from "http://www.numerical.rl.ac.uk/reports/reports.html".

[4] The work of this author was supported by the EPSRC grant GR/R46641.

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

October 3, 2003.

# 1 Introduction

One possible approach to the problem of rapidly solving very large sparse linear systems of equations

$$Ax = b$$

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` (Scott, 2001) and `HSL_MP48` (Duff and Scott, 2002) from the HSL mathematical software library (HSL, 2002) employ this technique for solving unsymmetric systems. To use these codes, the matrix $A$ must be preordered to singly bordered block diagonal (SBBD) form

$$\begin{pmatrix} A_{11} & & & & C_1 \\ & A_{22} & & & C_2 \\ & & \cdots & & . \\ & & & A_{NN} & C_N \end{pmatrix}, \tag{1.1}$$

where the rectangular blocks on the diagonal $A_{ll}$ are $m_l \times n_l$ matrices with $m_l \geq n_l$, and the border blocks $C_l$ are $m_l \times k$ with $k \ll n_l$.

`HSL_MP43` uses the frontal solver `MA42` (Duff and Scott, 1996) 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` (Duff and Reid, 1993). 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 to 16). In particular, the codes have been used to successfully solve highly unsymmetric linear systems arising from chemical process engineering applications (Scott, 2001, Duff and Scott, 2002). However, the effectiveness of the approach is dependent upon being able to obtain 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 problem of reordering chemical process engineering problems to SBBD form has been addressed by the MONET algorithm of Hu, Maguire and Blake (2000). 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 $\mathcal{G}_{AA^T}$ of $A$. Computing and working with the pattern of $AA^T$ is costly, because $AA^T$ may contain many more nonzero 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 $A^TA$) 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 (2002) on ordering the columns of sparse unsymmetric matrices to reduce fill-in during sparse $LU$ factorizations with partial pivoting. George and Ng (1988) 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 $A^TA$. Furthermore, for a large class of matrices, for every entry in the Cholesky factor of $A^TA$ there is a pivot sequence $P$ that causes that entry of $U$ to be nonzero (Gilbert and Ng, 1993). Thus, unsymmetric direct solvers often preorder the columns of $A$ using a permutation $Q$ that attempts to reduce the fill in the Cholesky factor of $Q^TA^TAQ$. The main challenge is to find a fill-minimizing permutation without computing $A^TA$ or its sparsity pattern. One approach to this problem is the column approximate minimum degree ordering algorithm (COLAMD) of Davis, Gilbert, Larimore and Ng (2000). Brainman and Toledo propose adopting an earlier idea of Gilbert and Schreiber (1982). Their method finds vertex separators in $\mathcal{G}_{A^TA}$ by finding *wide separators* in $\mathcal{G}_{A^T+A}$. 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 organised 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 $\mathcal{G}_{A^T+A}$ then, in Section 5, we propose computing SBBD forms directly from the vertex separators in $\mathcal{G}_{A^T+A}$. Vertex separtors are computed using the graph partitioning routine `METIS_PartGraphRecursive` from the well-known METIS package (Karypis and Kumar, 1998) 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 our highly unsymmetric test matrices and 8 blocks, we obtain borders within a factor of two of the MONET results. Furthermore, for matrices with a more symmetric structure, we generally achieve higher quality orderings, with the border columns representing only a small percentage of the total number of columns. The new orderings and the MONET orderings are used in Section 7 with the parallel solver `HSL_MP48`. We find that, even if the border is wider, the overall cost of reordering and then solving a single linear system is often less for the new algorithms than for MONET.

## 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. A † indicates that the problem is included in the University of Florida Sparse Matrix Collection (Davis, 1997). 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 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. The test matrices are listed in order of increasing symmetry index in Table 2.1.

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 a symmetric (or nearly symmetric) sparsity structure.

| Identifier | $n$ | $nz$ | $s(A)$ | Description/Application area |
|---|---|---|---|---|
| `Matrix35640` | 35640 | 146880 | 0.0001 | Chemical process engineering |
| `bayer01`† | 57735 | 277774 | 0.0002 | Chemical process engineering |
| `icomp` | 75724 | 338711 | 0.0010 | Chemical process engineering |
| `Matrix32406` | 32406 | 1035989 | 0.0014 | Chemical process engineering |
| `lhr34c`† | 35152 | 764014 | 0.0015 | Chemical process engineering |
| `bayer04`† | 20545 | 159082 | 0.0016 | Chemical process engineering |
| `lhr71c`† | 70304 | 1528092 | 0.0016 | Chemical process engineering |
| `poli_large`† | 15575 | 33074 | 0.0035 | Account of capital links |
| `4cols` | 11770 | 43668 | 0.0159 | Chemical process engineering |
| `10cols` | 29496 | 109588 | 0.0167 | Chemical process engineering |
| `onetone2`† | 36057 | 227628 | 0.1129 | Harmonic balance method |
| `ethylene-1` | 10673 | 80904 | 0.2973 | Chemical process engineering |
| `ethylene-2` | 10353 | 78004 | 0.3020 | Chemical process engineering |
| `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 |
| `wang3`† | 26064 | 177168 | 1.0000 | 3D diode semiconductor device |

Table 2.1: Test problems. $n$, $nz$ 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.

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* $v_1, v_2, ..., v_n$, 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 $\{1, 2, ..., n\}$ 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 $(v_{i_1}, v_{i_2}, ..., v_{i_{k+1}})$ where $(v_{i_j}, v_{i_{j+1}}) \in E$ for $1 \leq j \leq k$. 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 nonzero. If $A$ has a symmetric sparsity pattern, $\mathcal{G}(A)$ is undirected.

Row and column graphs were first introduced by Mayoh (1965). The *column graph* $\mathcal{G}_{A^TA}$ of $A$ is defined to be the undirected graph of the symmetric matrix $A^T * A$, 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 nonzero entry and the corresponding edge is included in the column graph). The vertices of $\mathcal{G}_{A^TA}$ 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 nonzero. The *row graph* $\mathcal{G}_{AA^T}$ is defined analogously as the undirected graph of $A * A^T$. Column (row) permutations of $A$ correspond to relabelling the vertices of the column (row) graph.

A subset $E_s \subset E$ 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 (1965) showed that, given a vertex separator in the column graph $\mathcal{G}_{A^TA}$, the matrix can be reordered to SBBD form. Suppose $S$ is a vertex separator in $\mathcal{G}_{A^TA}$ and let

$VC_1, VC_2, \ldots, VC_N$ be the subsets of columns of $A$ that correspond to the $N$ components of $\mathcal{G}_{A^TA}$ once $S$ and its incident edges have been removed. Then each row of $A$ has nonzero 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 nonzero entry in a column of $VC_i$ is ordered ahead of any row with a nonzero entry in a column of $VC_j$.

Thus, if we have a vertex separator in $\mathcal{G}_{A^TA}$, we can reorder $A$ to the required form. However, as already noted, computing $A^TA$ is expensive and we would like to find a vertex separator without forming $A^TA$ or its sparsity pattern. One possible approach is to use wide separators, a term coined by Gilbert and Schreiber (1982). 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 (2002) give the following result.

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

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 $\mathcal{G}_{A^TA}$ is a wide separator in $\mathcal{G}_{A^T+A}$.*

# 4 Computing SBBDs via wide separators

Theorem 1 provides a means of computing a vertex separator in $\mathcal{G}_{A^TA}$ without forming $A^TA$; the problem is reduced to computing a wide separator in the undirected graph $\mathcal{G}_{A^T+A}$. 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 (1998) (see www-users.cs.umn.edu/~karypis/metis/index.html). In particular, we use the routine `METIS_PartGraphRecursive` to partition $\mathcal{G}_{A^T+A}$ 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 (Dulmage and Mendelsohn, 1958; see also Pothen and Fan, 1990). 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, 1998 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, ..., 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 $\mathcal{G}_{A^T+A}$, 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 $W_1 = S \bigcup \{i | i \in V_1, (i,j) \in E \text{ for some } j \in S\}$ and $W_2 = S \bigcup \{i | i \in V_2, (i,j) \in E \text{ for some } j \in S\}$ 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 (2002) 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**

*Initialise* $W_s \Leftarrow S$

*For each vertex* $j \in S$

> *Let* $n_l$ *be number of neighbours* $i$ *of* $j$ *that belong to* $V_l$ *but not to* $W_s$ $(l = 1, 2)$*.*
>
> *If* $n_1 \le n_2$ *set* $W_s \Leftarrow W_s \bigcup \{i | i \in V_1, i \notin W_s, (i,j) \in E\}$*;*
> *otherwise set* $W_s \Leftarrow W_s \bigcup \{i | i \in V_2, i \notin W_s, (i,j) \in E\}$.

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.

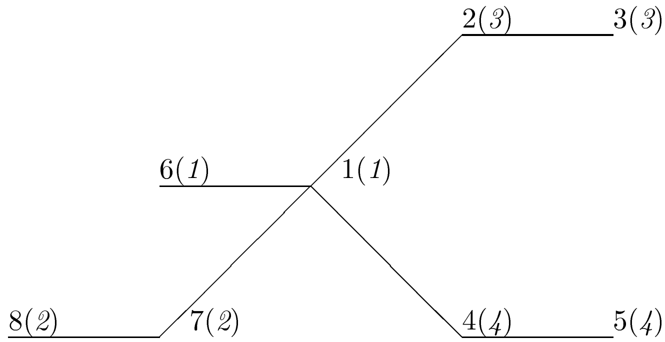


Figure 4.1: Figure illustrating that the union of the wide separators of bisections may not be a wide separator with regard to the overall partition of the graph. The numbers in brackets are the subgraph to which the vertex belongs after recursive bisection.

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 may not be a true wide separator of the original graph. For example, consider the graph $\mathcal{G}$ with eight vertices given in Figure 4.1. 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 4.1. 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 can be easily remedied when ordering to SBBD form, by bringing the few offending columns into the border, in a manner similar to Algorithm SBBD_Vertex Separator of Section 5.

In Table 4.1 we give the size of the border in the 8-block SBBD form obtained by computing the wide separator in $\mathcal{G}_{A^T+A}$ 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 (2002)

III: the above Wide Separator Algorithm.

The smallest border (and those within 5 per cent of the smallest) are highlighted in bold. For comparison, the size $|S|$ of the vertex separator computed using `METIS_PartGraphRecursive` and the Tůma software is given in column 3. We see that

| Identifier | $n$ | $\|S\|$ | Method | | |
|---|---|---|---|---|---|
| | | | I | II | III |
| `Matrix35640` | 35640 | 19888 | 33599 | **29998** | **29920** |
| `bayer01` | 57735 | 12264 | 19342 | 18609 | **17280** |
| `icomp` | 75724 | 289 | 427 | **401** | 441 |
| `Matrix32406` | 32406 | 13357 | 19836 | 18166 | **16989** |
| `lhr34c` | 35152 | 11943 | 24432 | 21394 | **19376** |
| `bayer04` | 20545 | 6630 | 10200 | 9939 | **9242** |
| `lhr71c` | 70304 | 12171 | 23799 | 20296 | **18864** |
| `poli_large` | 15575 | 199 | **665** | 1386 | 695 |
| `4cols` | 11770 | 305 | 524 | **498** | **477** |
| `10cols` | 29496 | 343 | 536 | 536 | **485** |
| `onetone2` | 36057 | 1981 | 3256 | 3897 | **2352** |
| `ethylene-1` | 10673 | 308 | 705 | 737 | **628** |
| `ethylene-2` | 10353 | 302 | 641 | 678 | **535** |
| `Zhao2` | 33861 | 1435 | **3051** | **3059** | **3014** |
| `scircuit` | 170998 | 449 | **1297** | 2071 | **1292** |
| `hcircuit` | 105676 | 509 | **1482** | 4219 | 2595 |
| `bcircuit` | 68902 | 279 | **632** | 702 | **631** |
| `garon2` | 13535 | 756 | **2059** | **2059** | **2059** |
| `pesa` | 11738 | 213 | **438** | **438** | **445** |
| `wang3` | 26064 | 2544 | **5069** | **4912** | **4904** |

Table 4.1: The size of the border in the 8-block SBBD form computed using wide separators in $\mathcal{G}_{A^T+A}$. $|S|$ denotes the size of the vertex separator.

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 $G_{A^TA}$ and wide separators in $G_{A+A^T}$. Therefore we would expect that if $A$ can be preordered to a more symmetric matrix $B$, then vertex separators in $G_{A^TA}$ should be "captured" better by wide separators in $G_{B+B^T}$ (note that row and column permutations do not change the graph $G_{A^TA}$, 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, 1999). Permuting a large number of nonzero off-diagonal entries onto the diagonal reduces the number of unmatched nonzero off-diagonal entries, which in turn increases the symmetry index. Furthermore, if $A$ is permuted to a matrix $B$ with a nonzero diagonal, the following theorem proves that every vertex separator in $\mathcal{G}_{B^TB}$ is a vertex separator in $\mathcal{G}_{B^T+B}$.

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

**Proof** Let $S$ be a vertex separator in $\mathcal{G}_{B^TB}$ such that the removal of $S$ and its incident edges breaks the graph into $N$ components. Let $V_1, V_2, \dots, V_N$ be the subsets of the vertices corresponding to the $N$ components. Suppose for contradiction that $S$ is not a separator in $\mathcal{G}_{B^T+B}$. Then there exists a path in $\mathcal{G}_{B^T+B}$ 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 $\mathcal{G}_{B^T+B}$, the $(i,j)$ entry of $B^T + B$ is nonzero. Therefore, either $b_{ij} \neq 0$ or $b_{ji} \neq 0$. Since $b_{ii} \neq 0$ and $b_{jj} \neq 0$, it follows that $(B^TB)_{ij} = \sum_q b_{qi}b_{qj}$ is nonzero. Thus $(i,j)$ must an edge in $\mathcal{G}_{B^TB}$ and $i \leftrightarrow j$ is a path in $\mathcal{G}_{B^TB}$ that does not pass through $S$, a contradiction.

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

The HSL routine `MC21` uses a relatively simple algorithm to compute a matching that corresponds to a row permutation of $A$ that puts nonzeros 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 (1981*a*, 1981*b*). In Table 4.2, 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 2.1 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 4.3, we show the size of the border in the 8-block SBBD form obtained by computing the wide separator in $\mathcal{G}_{B^T+B}$ using Methods I, II and III. The best results (and those within 5 per cent 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 examples. For the remaining problems, we see that Method III remains the method of choice and, comparing the results with those in Table 4.1, it is clear that preordering $A$ can lead to a dramatic reduction in the border

| Identifier | $n$ | Diagonal zeros | Symmetry index Before | After |
|---|---|---|---|---|
| Matrix35640 | 35640 | 35639 | 0.0001 | 0.0427 |
| bayer01 | 57735 | 57733 | 0.0002 | 0.0719 |
| icomp | 75724 | 0 | 0.0010 | 0.0025 |
| Matrix32406 | 32406 | 32366 | 0.0014 | 0.2643 |
| lhr34c | 35152 | 35050 | 0.0015 | 0.3294 |
| bayer04 | 20545 | 20545 | 0.0016 | 0.0694 |
| lhr71c | 70304 | 70100 | 0.0016 | 0.3541 |
| poli_large | 15575 | 0 | 0.0035 | 0.0035 |
| 4cols | 11770 | 0 | 0.0159 | 0.0419 |
| 10cols | 29496 | 0 | 0.0167 | 0.0471 |
| onetone2 | 36057 | 26967 | 0.1129 | 0.3600 |
| ethylene-1 | 10673 | 0 | 0.2973 | 0.2441 |
| ethylene-2 | 10353 | 0 | 0.3020 | 0.2487 |
| Zhao2 | 33861 | 0 | 0.9225 | 0.9225 |
| scircuit | 170998 | 84 | 0.9999 | 0.9995 |
| hcircuit | 105676 | 48 | 0.9999 | 0.9852 |

Table 4.2: Structural symmetry before and after permuting the rows of $A$ using the MC21 ordering.

| Identifier | $n$ | $\|S\|$ | Method I | II | III |
|---|---|---|---|---|---|
| Matrix35640 | 35640 | 1313 | **1949** | 2221 | 2038 |
| bayer01 | 57735 | 247 | **437** | 545 | **432** |
| icomp | 75724 | 299 | **411** | **427** | **412** |
| Matrix32406 | 32406 | 1504 | 2470 | 3168 | **2336** |
| lhr34c | 35152 | 769 | 1505 | 1959 | **1346** |
| bayer04 | 20545 | 390 | **621** | **621** | **612** |
| lhr71c | 70304 | 918 | 1458 | 1772 | **1378** |
| 4cols | 11770 | 211 | 369 | 354 | **294** |
| 10cols | 29496 | 275 | 447 | 446 | **384** |
| onetone2 | 36057 | 1434 | **2832** | 3391 | **2825** |
| ethylene-1 | 10673 | 248 | 612 | 570 | **484** |
| ethylene-2 | 10353 | 239 | 565 | 513 | **487** |
| scircuit | 170998 | 444 | **1255** | 1856 | **1237** |
| hcircuit | 105676 | 458 | **1051** | 1361 | **1052** |

Table 4.3: The size of the border in the 8-block SBBD form computed using wide separators in $\mathcal{G}_{B^T+B}$. $|S|$ denotes the size of the vertex separator.

size. In particular, for lhr71c the border size is reduced from 18864 to 1378 columns, and for Matrix35640 the percentage of border columns is cut to less than 6%.

| Identifier | I | | II | | III | |
|---|---|---|---|---|---|---|
| | 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 |

Table 4.4: The row differences for the 8-block SBBD form computed using wide separators in $\mathcal{G}_{A^T+A}$ (denoted by "Before") and $\mathcal{G}_{B^T+B}$ (denoted by "After").

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 4.4, we give the row differences for the 8-block SBBD forms computed using Methods I, II and III applied to $\mathcal{G}_{A^T+A}$ and $\mathcal{G}_{B^T+B}$. The columns labelled "Before" are computed using wide separators in $\mathcal{G}_{A^T+A}$ and those labelled "After" use $\mathcal{G}_{B^T+B}$. We see that, for a number of the highly unsymmetric problems, the row imbalance when wide separators are computed using $\mathcal{G}_{A^T+A}$ 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 4.3 and 4.4 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 4.1 and 4.3). Since for matrices with a nonzero diagonal and unsymmetric sparsity pattern a vertex separator in $\mathcal{G}_{A^TA}$ is not necessarily a wide separator in $\mathcal{G}_{A^T+A}$ but, by Theorem 3, is a vertex separator in $\mathcal{G}_{A^T+A}$, it may be advantageous to try and compute the SBBD directly from the vertex separator in $\mathcal{G}_{A^T+A}$ (or $\mathcal{G}_{B^T+B}$), without computing wide separators.

Suppose $S$ is a vertex separator in $\mathcal{G}_{A^T+A}$. Let $VC_1, VC_2, \ldots VC_N$ be the subsets of columns of $A$ that correspond to the $N$ components of $\mathcal{G}_{A^T+A}$ 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 nonzero entries. Let $row_i$ and $VC_j$ denote the $i$th row and $j$th column of $A$. Suppose $row_i$ has a nonzero entry in $VC_j$. If $VC_j$ belongs to $S$ then we do nothing. Otherwise, $VC_j$ must belong to one of the subsets $VC_l$ ($1 \le l \le N$). If $row_i$ has not yet been assigned to a partition, we assign $row_i$ to partition $l$. Otherwise, $row_i$ belongs to partition $k$ for some $k \neq l$ ($row_i$ has entries belonging to more than one of the sets $VC_l$). In this case, we move column $VC_j$ into the set $S$. Once all the rows have been considered, the only rows that are still unassigned are those which have all their nonzero 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 $block(i)$ denotes the partition in the SBBD form to which row $row_i$ is assigned, the above algorithm can be summarised as follows.

**Algorithm: SBBD_Vertex Separator**

1. *Set $S$ be a vertex separator in $\mathcal{G}_{A^T+A}$. Initialise $block(1:n) = 0$.*
2. *For each $row_i$, consider the columns $VC_j$ of its nonzero entries.*
   *If $VC_j \in VC_l$ then*
   *If $block(row_i) = 0$, set $block(row_i) = l$*
   *else remove column $VC_j$ from $VC_l$ and add to $S$.*
3. *Once all rows considered, assign any rows for which $block(row_i) = 0$ equally between the $N$ partitions.*

In practice, once all the rows have been assigned to a partition, we check that there are no redundant columns in the set $S$, that is, columns with entries in only one partition. If $VC_k \in S$ has nonzero entries only in rows belonging to partition $m$, $VC_k$ is removed from $S$ and added to $VC_m$.

In Table 5.1 results are presented for this vertex separator method (which we refer 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 of Hu et al., 2000). Results are give 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 $B^T + B$ generally led to wider borders.

With $N = 2$, both Methods III and VS achieve narrower borders than `HSL_MC66` for a large proportion of the test examples and for some examples (including `Matrix32406`

| Identifier | $n$ | Number of blocks | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $N=2$ | | | $N=4$ | | | $N=8$ | | |
| | | III | VS | MC66 | III | VS | MC66 | III | VS | MC66 |
| Matrix35640 | 35640 | 490 | 438 | **344** | 1008 | 1031 | **704** | 2038 | 1957 | **1367** |
| bayer01 | 57735 | 90 | **76** | 71 | 234 | 198 | **135** | 432 | 353 | **254** |
| icomp | 75724 | **41** | **43** | 55 | 213 | 191 | **134** | 412 | 363 | **229** |
| Matrix32406 | 32406 | **170** | **167** | 1215 | **1112** | **1121** | 2539 | 2336 | **2179** | 3514 |
| lhr34c | 35152 | 509 | 518 | **94** | 965 | 966 | **354** | 1346 | 1316 | **792** |
| bayer04 | 20545 | **87** | 109 | 182 | **306** | 326 | 369 | 612 | **539** | **542** |
| lhr71c | 70304 | **154** | **147** | 198 | 775 | 769 | **392** | 1378 | 1351 | **990** |
| poli_large | 15575 | **303** | 421 | 394 | **568** | 874 | **582** | **695** | 1023 | **713** |
| 4cols | 11770 | 33 | 34 | **30** | 95 | **70** | 106 | 294 | **222** | **233** |
| 10cols | 29496 | **32** | 33 | **30** | 167 | 142 | **123** | 384 | 300 | **279** |
| onetone2 | 36057 | 469 | **268** | **254** | 1967 | 1545 | **1204** | 2825 | 2097 | **1745** |
| ethylene-1 | 10673 | **38** | **39** | 75 | 202 | 142 | **111** | 484 | 320 | **217** |
| ethylene-2 | 10353 | **27** | **27** | 50 | 151 | **97** | 133 | 487 | 299 | **217** |
| Zhao2 | 33861 | 666 | 742 | **641** | 1794 | 1975 | **1688** | 3014 | 3320 | **2773** |
| scircuit | 170998 | **58** | 78 | 2551 | **594** | 745 | 3753 | **1237** | 1551 | 4353 |
| hcircuit | 105676 | **191** | **191** | 591 | **364** | **375** | 891 | **1052** | 1154 | 2138 |
| bcircuit | 68902 | **3** | **3** | 563 | **142** | 183 | 737 | **631** | 872 | 951 |
| garon2 | 13535 | **542** | **556** | 682 | **1100** | **1116** | 1543 | **2059** | **2054** | 2308 |
| pesa | 11738 | **81** | **80** | 127 | **192** | **196** | 245 | **445** | **456** | **446** |
| wang3 | 26064 | **1740** | **1740** | **1740** | **3355** | **3315** | **3310** | **4904** | **4840** | **4813** |

Table 5.1: The size of the border in the SBBD form computed using the wide separator Method III, the vertex separator Method VS, and HSL_MC66.

and the circuit problems), the improvements are significant. As $N$ increases, the size of the border grows. The results in Table 5.1 suggest that the increase in border size is less for HSL_MC66 than for the other approaches. In particular, for $N = 8$, HSL_MC66 returns the smallest borders for the majority of the highly unsymmetric examples in the top half of the table (and although not reported on here, we found that this trend continues for $N = 16$). However, comparing the final two columns of the table, we see that with $N = 8$, Method VS produces a border that is generally less than 1.5 times the size of the HSL_MC66 border and, for the (nearly) symmetrically structured problems, both Methods III and VS perform as well as, or better than, HSL_MC66.

As $N$ increases, the vertex separator approach (VS) often outperforms the wide separator approach (III) when used on the unsymmetric problems. But for the (nearly) symmetrically structured problems given towards the end of the table, Method III generally results in narrower borders. For a given problem and given $N$ it is not possible to predict which method will give the narrowest border. The main cost involved in computing the separator-based orderings is that of preordering using MC21 (where appropriate) and then computing an edge separator using METIS_PartGraphRecursive. However, this needs to be done only once; we can then run each of the separator methods and choose the one that yields the narrowest border. Our findings suggest that Method II is generally poorer than the other methods and so we propose running Methods I, III, and VS and selecting the best ordering; we will call this the SEP_VS Method.

## 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 $\mathcal{G}_{A+A^T}$ (or $\mathcal{G}_{B+B^T}$) 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 2, 4 and 8 blocks are given in Table 6.1. 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.

| Identifier | $n$ | Number of blocks | | | | | |
|---|---|---|---|---|---|---|---|
| | | $N=4$ | | | $N=8$ | | |
| | | III(ND) | VS(ND) | `MC66` | III(ND) | VS(ND) | `MC66` |
| `Matrix35640` | 35640 | 857 | 856 | **704** | 1779 | 1792 | **1367** |
| `bayer01` | 57735 | 328 | 339 | **135** | 592 | 571 | **254** |
| `icomp` | 75724 | 328 | 333 | **134** | 392 | 371 | **229** |
| `Matrix32406` | 32406 | **1103** | 1153 | 2539 | **1840** | 1914 | 3514 |
| `lhr34c` | 35152 | 442 | 458 | **354** | 1015 | 1028 | **792** |
| `bayer04` | 20545 | **216** | 233 | 369 | 527 | **508** | 542 |
| `lhr71c` | 70304 | **398** | **398** | **392** | **1006** | 1035 | **990** |
| `poli_large` | 15575 | 1159 | 1388 | **582** | 2577 | 2649 | **713** |
| `4cols` | 11770 | 91 | **74** | 106 | 311 | 265 | **233** |
| `10cols` | 29496 | **128** | **128** | **123** | 322 | 306 | **279** |
| `onetone2` | 36057 | 917 | **843** | 1204 | 2052 | **1721** | **1745** |
| `ethylene-1` | 10673 | **73** | 93 | 111 | 324 | **211** | **217** |
| `ethylene-2` | 10353 | **76** | **78** | 133 | 298 | **208** | **217** |
| `Zhao2` | 33861 | 1858 | 1871 | **1688** | 3417 | 3427 | **2773** |
| `scircuit` | 170998 | **1067** | **1067** | 3753 | **1751** | **1761** | 4353 |
| `hcircuit` | 105676 | 996 | 1022 | **891** | 2574 | 2687 | **2138** |
| `bcircuit` | 68902 | **127** | **127** | 737 | **679** | 701 | 951 |
| `garon2` | 13535 | **1226** | **1233** | 1543 | **2099** | **2106** | 2308 |
| `pesa` | 11738 | **191** | **195** | 245 | 493 | 493 | **446** |
| `wang3` | 26064 | **3118** | **3105** | 3310 | **4395** | **4352** | 4813 |

Table 6.1: The size of the border in the SBBD form computed using the nested dissection-based methods III(ND) and VS(ND), and `HSL_MC66`.

Because there is no clear winner between III(ND) and VS(ND) and the main cost is using `MC21` to preorder (where appropriate) and then computing the vertex separator using `METIS_NodeND`, we again propose running both methods and selecting the one that gives the smallest border. We will refer to this as the SEP_VS(ND) Method.

In Table 6.2 the border sizes for SEP_VS and SEP_VS(ND) are compared with `HSL_MC66`. We see that for some problems, including `bayer04`, the `lhr` and the `ethylene` examples, using the nested dissection method leads to the narrowest borders. But for

| Identifier | $n$ | Number of blocks | | | | | |
|---|---|---|---|---|---|---|---|
| | | $N = 4$ | | | $N = 8$ | | |
| | | SEP_VS | SEP_VS(ND)) | MC66 | SEP_VS | SEP_VS(ND)) | MC66 |
| Matrix35640 | 35640 | 1008 | 856 | **704** | 1957 | 1779 | **1367** |
| bayer01 | 57735 | 198 | 328 | **135** | 353 | 571 | **254** |
| icomp | 75724 | 191 | 328 | **134** | 363 | 371 | **229** |
| Matrix32406 | 32406 | **1121** | **1103** | 2539 | 2179 | **1840** | 3514 |
| lhr34c | 35152 | 965 | 442 | **354** | 1316 | 1015 | **792** |
| bayer04 | 20545 | 306 | **216** | 369 | 539 | **508** | 542 |
| lhr71c | 70304 | 769 | **398** | **392** | 1351 | **1006** | **990** |
| poli_large | 15575 | **568** | 1159 | **582** | **695** | 2577 | **713** |
| 4cols | 11770 | **70** | **74** | 106 | **222** | 265 | **233** |
| 10cols | 29496 | 142 | **128** | **123** | 300 | 306 | **279** |
| onetone2 | 36057 | 1545 | **843** | 1204 | 2097 | **1721** | **1745** |
| ethylene-1 | 10673 | 142 | **73** | 111 | 320 | **211** | **217** |
| ethylene-2 | 10353 | 97 | **76** | 133 | 299 | **208** | **217** |
| Zhao2 | 33861 | 1794 | 1858 | **1688** | 3014 | 3417 | **2773** |
| scircuit | 170998 | **594** | 1067 | 3753 | **1237** | 1751 | 4353 |
| hcircuit | 105676 | **364** | 996 | 891 | **1052** | 2574 | 2138 |
| bcircuit | 68902 | 142 | **127** | 737 | **631** | 679 | 951 |
| garon2 | 13535 | **1100** | 12263 | 1543 | **2054** | **2099** | 2308 |
| pesa | 11738 | **192** | **191** | 245 | **445** | 493 | **446** |
| wang3 | 26064 | 3315 | **3105** | 3310 | 4840 | **4352** | 4813 |

Table 6.2: The size of the border in the SBBD form computed using the SEP_VS and SEP_VS(ND), and HSL_MC66.

other problems (notably poli_large, scircuit, and hcircuit) nested dissection gives much poorer results. We also find that the row differences are significantly larger for the SEP_VS(ND) method. For example, for bayer04 with 8 blocks, the row difference for Method SEP_VS(ND) is 12.5% compared with 1.2% for Method SEP_VS. Similarly, for ethylene-2 the row differences are 15.6% and 2.9% for SEP_VS(ND) and SEP_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 7.1, we compare the run times for Methods SEP_VS and SEP_VS(ND) with those for HSL_MC66. For SEP_VS and SEP_VS(ND), the times include (where appropriate) the time taken to run MC21 and to permute $A$ using the MC21 ordering prior to computing the SBBD form. The times also include the METIS time plus the total time for the suborderings (so that for SEP_VS the times for Methods I, III, and VS are summed). All timings are CPU times in seconds.

| Identifier | Number of blocks | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $N = 2$ | | | $N = 4$ | | | $N = 8$ | | |
| | SEP_VS | SEP_VS (ND) | MC66 | SEP_VS | SEP_VS (ND) | MC66 | SEP_VS | SEP_VS (ND) | MC66 |
| Matrix35640 | 0.45 | 1.78 | 2.02 | 0.59 | 1.70 | 4.31 | 0.71 | 1.57 | 6.93 |
| bayer01 | 0.63 | 2.58 | 2.54 | 0.82 | 2.09 | 4.51 | 0.99 | 2.13 | 6.33 |
| icomp | 0.45 | 1.24 | 2.06 | 0.71 | 1.44 | 3.93 | 0.92 | 1.68 | 5.71 |
| Matrix32406 | 1.41 | 5.05 | 98.8 | 1.89 | 5.53 | 237 | 2.42 | 5.51 | 324 |
| lhr34c | 0.88 | 3.36 | 3.38 | 1.22 | 3.60 | 6.49 | 1.47 | 3.30 | 9.64 |
| bayer04 | 0.23 | 0.77 | 1.50 | 0.32 | 0.83 | 2.71 | 0.42 | 0.88 | 3.70 |
| lhr71c | 1.99 | 7.89 | 7.98 | 2.56 | 7.52 | 13.2 | 3.23 | 7.95 | 19.3 |
| poli_large | 0.09 | 0.12 | 0.08 | 0.17 | 0.16 | 0.14 | 0.21 | 0.22 | 0.21 |
| 4cols | 0.08 | 0.91 | 0.34 | 0.09 | 0.87 | 0.64 | 0.14 | 0.67 | 0.91 |
| 10cols | 0.21 | 2.91 | 0.89 | 0.26 | 2.64 | 1.64 | 0.36 | 2.27 | 2.45 |
| onetone2 | 0.92 | 1.32 | 3.57 | 0.99 | 1.29 | 5.68 | 1.10 | 1.53 | 8.12 |
| ethylene-1 | 0.10 | 0.21 | 1.51 | 0.13 | 0.23 | 1.94 | 0.17 | 0.26 | 2.59 |
| ethylene-2 | 0.09 | 0.20 | 1.50 | 0.12 | 0.22 | 2.62 | 0.17 | 0.26 | 3.07 |
| Zhao2 | 0.16 | 0.60 | 0.70 | 0.26 | 0.61 | 1.53 | 0.37 | 0.67 | 2.44 |
| scircuit | 1.25 | 2.39 | 26.6 | 1.34 | 2.66 | 44.7 | 2.24 | 3.12 | 54.7 |
| hcircuit | 0.65 | 0.92 | 6.06 | 0.73 | 1.13 | 11.2 | 1.21 | 1.30 | 16.1 |
| bcircuit | 0.32 | 0.72 | 2.03 | 0.34 | 0.84 | 3.82 | 0.67 | 1.01 | 5.34 |
| garon2 | 0.19 | 0.17 | 0.80 | 0.19 | 0.23 | 1.54 | 0.39 | 0.29 | 2.35 |
| pesa | 0.05 | 0.10 | 0.25 | 0.05 | 0.12 | 0.48 | 0.11 | 0.14 | 0.75 |
| wang3 | 0.15 | 0.42 | 0.80 | 0.16 | 0.48 | 1.52 | 0.30 | 0.54 | 2.37 |

Table 7.1: The times (in seconds) to compute the SBBD form using the SEP_VS and SEP_VS(ND) Methods and HSL_MC66.

On many problems, SEP_VS is more than twice as fast as SEP_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 7.2 for solving a single linear system once it is in SBBD form using the direct solver HSL_MP48. These timings are for a subset of our test problems with $N = 8$ run on both processors of our Compaq DS20. They are elapsed times in seconds, measured using the `MPI` timer `MPI_WTIME` on the host processor.

| Identifier | Ordering method | | | |
|---|---|---|---|---|
| | SEP_VS | | MC66 | |
| `Matrix35640` | 27.6 | (12.4) | 12.8 | (4.80) |
| `bayer01` | 1.81 | (0.06) | 1.70 | (0.03) |
| `icomp` | 0.41 | (0.002) | 0.41 | (0.001) |
| `lhr34c` | 9.97 | (2.12) | 7.63 | (0.67) |
| `lhr71c` | 27.8 | (2.44) | 22.5 | (0.95) |
| `4cols` | 0.17 | (0.02) | 0.15 | (0.02) |
| `10cols` | 0.61 | (0.03) | 0.55 | (0.03) |
| `ethylene-1` | 0.30 | (0.01) | 0.25 | (0.01) |
| `scircuit` | 17.4 | (3.00) | 43.1 | (31.6) |
| `bcircuit` | 2.90 | (0.30) | 2.89 | (0.50) |
| `garon2` | 24.8 | (11.8) | 25.6 | (14.7) |

Table 7.2: `HSL_MP48` times for solving a single linear system after ordering to SBBD form ($N = 8$). The numbers in parentheses are the times for analysing and factorizing the interface problems.

The results in Table 7.2 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 SEP_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 (Table 7.1), several factorizations of matrices having the same pattern are needed to justify the extra cost of ordering using `HSL_MC66`.

# 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 $A^T + A$. 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 highly unsymmetric problems, as the number of blocks increases, the border is generally larger for the new methods than for the existing MONET algorithm of Hu et al. (2000). However, for more symmetrically structured examples, the separator methods often 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, even though the border may be wider, 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.

# 9 Acknowledgements

We are very grateful to Mirek Tůma 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.

# References

C. Ashcraft and J.W.H. Liu. Applications of the Dulmage-Mendelsohn decomposition and network flow to graph bisection improvement. *SIAM J. Matrix Analysis and Applications*, **19**, 325–354, 1998.

I. Brainman and S. Toledo. Nested-dissection orderings for sparse LU with partial pivoting. *SIAM J. Matrix Analysis and Applications*, **23**, 998–1012, 2002.

T. Davis. University of Florida Sparse Matrix Collection. *NA Digest*, **97**(23), 1997. Full details from www.cise.ufl.edu/∼davis/sparse/.

T.A. Davis and J.R. Gilbert and S.I. Larimore and E.G. Ng. A column approximate minimum degree ordering algorithm. Technical Report TR-00-005, Department of Computer and Information Science and Engineering, University of Florida.

I.S. Duff. Algorithm 575, permutations for a zero-free diagonal. *ACM Trans. Mathematical Software*, **7**, 387–390, 1981*a*.

I.S. Duff. On algorithms for obtaining a maximum transversal. *ACM Trans. Mathematical Software*, **7**, 315–330, 1981*b*.

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

I.S. Duff and J.K. Reid. MA48, a Fortran code for direct solution of sparse unsymmetric linear systems of equations. Report RAL-93-072, Rutherford Appleton Laboratory, 1993.

I.S. Duff and J.A. Scott. The design of a new frontal code for solving sparse unsymmetric systems. *ACM Trans. Mathematical Software*, **22**(1), 30–45, 1996.

I.S. Duff and J.A. Scott. A parallel direct solver for large sparse highly unsymmetric linear systems. Technical Report RAL-TR-2002-033, Rutherford Appleton Laboratory, 2002.

A. Dulmage and N. Mendelsohn. Coverings of bipartite graphs. *Canad. J. Math.*, **10**, 517–534, 1958.

A. George and E. Ng. On the complexity of sparse QR and LU factorization on finite-element matrices. *SIAM J. Scientific and Statistical Computing*, **9**, 849–861, 1988.

J.R. Gilbert and E. Ng. Predicting structure in nonsymmetric sparse matrix factorizations. *in* A. George, J. Gilbert and J. Liu, eds, 'Graph Theory and Sparse Matrix Computation'. Springer-Verlag, New York, 1993.

J.R. Gilbert and R. Schreiber. Nested dissection with partial pivoting. *in* 'Sparse Matrix Symposium 1982: Program and Abstracts, Fairfield Glade, TN', 1982.

HSL. A collection of Fortran codes for large scale scientific computation, 2002. Full details from www.cse.clrc.ac.uk/nag/hsl/.

Y.F. Hu, K.C.F. Maguire, and R.J. Blake. A multilevel unsymmetric matrix ordering for parallel process simulation. *Computers in Chemical Engineering*, **23**, 1631–1647, 2000.

G. Karypis and V. Kumar. METIS: A software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices - version 4.0, 1998.

B.H. Mayoh. A graph technique for inverting certain matrices. *Mathematics of Computation*, **19**, 644–646, 1965.

A. Pothen and C-J. Fan. Computing the block triangular form of a sparse matrix. *ACM Trans. Mathematical Software*, **16**, 303–324, 1990.

J.A. Scott. The design of a portable parallel frontal solver for chemical process engineering problems. *Computers in Chemical Engineering*, **25**, 1699–1709, 2001.