Matrix Ordering Strategies for Process Engineering: Graph Partitioning Algorithms for Parallel Computation

Kyle V. Camarda*
Department of Chemical Engineering
University of Illinois
600 S. Mathews Avenue
Urbana, IL 61801 USA

Mark A. Stadtherr†
Department of Chemical Engineering
University of Notre Dame
Notre Dame, IN 46556 USA

(August 1998)
(revised, March 1999)

Keywords: Simulation, Optimization, Design, Sparse Matrices, Parallel Computation, Graph Partitioning

*Currently at Department of Chemical Engineering, Pennsylvania State University, University Park, PA 16802 U.S.A.

†Author to whom all correspondence should be addressed. Phone: (219)631-9318; Fax: (219)631-8366; E-mail: markst@nd.edu

Abstract

The solution of large-scale chemical process simulation and optimization problems using parallel computation requires algorithms that can take advantage of multiprocessing when solving the large, sparse matrices that arise. Parallel algorithms require that the matrices be partitioned in order to distribute computational work across processors. One way to accomplish this is to reorder the matrix into a bordered block-diagonal form. Since this structure is not always obtained from the equation generation routine, an algorithm to reorder the rows and columns of the coefficient matrix is needed. We describe here a simple graph partitioning algorithm that creates a bordered block-diagonal form that is suitable for use with parallel algorithms for the solution of the highly asymmetric sparse matrices arising in process engineering applications. The method aims to create a number of similarly sized diagonal blocks while keeping the size of the interface matrix, which may represent a bottleneck in the parallel computation, reasonably small. Results on a wide range of test problems indicate that the reordering algorithm is able to find such a structure in most cases, and requires much less reordering time than previously used graph partitioning methods.

1 Introduction

The simulation and optimization, offline or online, of industrial-scale chemical processes is a computationally intense problem that may involve several hundreds of thousands of equations and variables. The use of parallel computing technology provides the potential not only to solve such problems faster, but also to further expand the problem size threshold. As many of the original algorithms were designed with serial computers in mind, new computational strategies need to be developed to better exploit advanced computer architectures. An example along these lines is given by Zitney *et al.* (1995), who showed that the time required to solve a large dynamic simulation problem was reduced from 18 hours to 21 minutes of CPU time on a CRAY C90 supercomputer by the use of a different sparse linear equation solving strategy (Zitney and Stadtherr, 1993) and other improvements.

In most industrial-scale process engineering problems, the solution of large, sparse systems of linear equations is the single most computationally intensive step, requiring as much as 90% of the total simulation time in some cases (Zitney, 1992; Zitney, Camarda and Stadtherr, 1994). Reducing the time required to solve these systems can thus have a very significant effect on the total computation time. One approach to parallelizing the solution of the linear equation system is to use a problem decomposition corresponding to a bordered block-diagonal matrix structure, and to factorize the diagonal blocks in parallel. Two proven solvers that use this approach are the Harwell routine MA-52, and the Cray Research routine PFAMP (Mallya *et al.*, 1997a), with the latter designed specifically for process engineering problems and the former for finite element problems. Both of these solvers are designed for vector multiprocessing machines, and thus use a frontal solver to factorize the diagonal blocks. However, any factorization scheme can be used on the diagonal blocks and thus the bordered block-diagonal decomposition can be used in connection with other multiprocessing architectures.

In general, obtaining the desired bordered block-diagonal structure requires a matrix reordering step. While many algorithms exist to determine such reorderings for symmetric matrices, and these can generally be extended to structurally unsymmetric matrices if the degree of asymmetry is not high, few have been developed for matrices with a very high degree of structural asymmetry, such as process engineering matrices. In this paper we describe a simple new algorithm for determining row and column permutations to bordered block-diagonal form for highly asymmetric sparse matrices. The algorithm is based on the graph partitioning concepts developed by Coon and Stadtherr (1995), and represents a simplification of their min-net-cut (MNC) methods. Results on a wide variety of problems indicate that the new reordering algorithm creates matrix structures suitable for parallel computation, and does so much more efficiently than previous methods.

2 Background

Consider the solution of a linear equation system $Ax = b$, where A is a large sparse $n \times n$ matrix and x and b are column vectors of length n. While iterative methods can be used to solve such systems, the reliability of such methods is questionable in the context of process simulation (Cofer and Stadtherr, 1996), since in this context A does not have desirable properties such as symmetry (numerical or structural) or positive definiteness. Thus we concentrate here on direct methods. Generally such methods can be interpreted as an LU factorization scheme in which A is factored $A = LU$, where L is a lower triangular matrix and U is an upper triangular matrix. Thus, $Ax = (LU)x = L(Ux) = b$, and the system can be solved by a simple forward substitution to solve $Ly = b$ for y, followed by a back substitution to find the solution vector x from $Ux = y$.

Direct parallel solution methods aim to find elements of L and U by factoring multiple partitions of A simultaneously, with the partitions typically representing diagonal blocks in some type of bordered block-diagonal form, or perhaps a bordered block-triangular form (e.g., Gallivan *et al.*, 1996). The reordering algorithm given here is for the case in which a bordered block-diagonal form is used. For example, consider a matrix in row-bordered block-diagonal form:

$$A = \left[\begin{array}{cccc} A_{11} & & & \\ & A_{22} & & \\ & & \ddots & \\ & & & A_{NN} \\ \hline S_1 & S_2 & \cdots & S_N \end{array}\right].$$

Here the diagonal blocks A_{ii} are $m_i \times n_i$ and in general can be rectangular. For process simulation problems, the equation generator could take advantage of the unit-stream nature of a flowsheet to create a matrix in this form, as described by Westerberg and Berna (1978). In this case, each A_{ii} would represent the model equations for a particular unit, while the border contains connection equations and design specifications. However, not all commercial simulators generate a matrix with this structure, and even if they did it is not necessarily going to be the most attractive structure for purposes of parallel computing. Thus there is a need for matrix reordering. One reordering scheme that exploits the unit-stream structure of the problem is that of Abbott *et. al.* (1997). While not designed specifically for parallel computation, it is capable of producing useful orderings for this purpose (Mallya *et al.*, 1997b). Still, in most commercial software, information about the unit-stream structure is not available to the sparse matrix solver, and thus a more general reordering approach is needed.

The basic idea of the parallel factorization is to partially factor each of the A_{ii}, with each such task being performed independently and in parallel. Since the A_{ii} are in general rectangular, some variables in the block may not be eliminated. For reasons of numerical stability, certain equations may also not be eliminated. These combine with the border equations to form an interface matrix, which is factored after completion of the parallel factorization of the A_{ii}. After factorization of the diagonal blocks, we have (Mallya *et al.*, 1997a):

$$\left[\begin{array}{cccc|c} L_1U_1 & & & & U_1' \\ & L_2U_2 & & & U_2' \\ & & \ddots & & \vdots \\ & & & L_NU_N & U_N' \\ \hline L_1' & L_2' & \cdots & L_N' & F \end{array}\right].$$

Here F is the interface matrix. Note that, in general it will contain contributions from the factorization of each diagonal block. After factorization of F to complete the LU factorization of A, the triangular solves which must be done in order to compute a solution vector can be completed in parallel using the same block structure used in computing the L and U factors. Factorization of the diagonal blocks and of the interface matrix can be done using any technique that accommodates pivoting, and the same technique need not be used on all blocks. Note that instead of beginning with a row-bordered form as above, we could also begin with a column-bordered form, as seen in an example below, or a row-and-column-bordered form. Pivoting within the diagonal blocks will result in appropriate rows and/or columns being put into the borders (corresponding to the interface matrix).

The diagonal blocks may be factored on a single processor, or on multiple processors using a smaller task granularity than in the bordered block-diagonal decomposition. Likewise, depending on its size and sparsity, the interface matrix may be factored on a single processor or on multiple processors using a small task granularity. In any case, however, the factorization of the interface matrix represents a synchronization point in the solution algorithm, and so may represent a bottleneck. Thus, it is desirable in seeking a reordering that the size of the interface problem be kept reasonably small. Of course, it is also desirable that the reordering produce diagonal blocks of a number and size that lead to good load balancing on the target architecture. The desired reordering may differ depending on the target architecture; thus, the reordering algorithm must be tunable and have some flexibility. Our experience with the MNC orderings indicated that, while this approach is tunable to some extent, it did not provide the desired flexibility. This was one motivating factor in the development of the reordering method described here.

3 Reordering Methods

For matrices that are structurally *symmetric* or nearly so, there are various approaches that can be used to try to get an appropriate matrix reordering (e.g., Fiduccia and Mattheyses, 1982; Schweikert and Kernighan, 1972; Kernighan and Lin, 1970; Leiserson and Lewis, 1989; O'Neil and Szyld, 1990; Karypis and Kumar, 1995; Choi and Szyld, 1996). These are generally based on solving *undirected* graph partitioning, bisection or min-cut problems, often in the contexts of nested dissection applied to finite element problems, block preconditioners for iterative linear solvers, or various other divide-and-conquer problems. Such methods are applied to a structurally *asymmetric* matrix A by applying them to the structure of the symmetric matrix $A + A^T$. This may provide satisfactory results if the degree of asymmetry is low. However, when the degree of asymmetry is very high, as in the case of process engineering problems, the approach cannot be expected to always yield good results (Mallya *et al.*, 1997b), as the number of additional nonzeros in $A + A^T$, indicating dependencies that are nonexistent in the problem, may be large, nearly as large as the number of nonzeros indicating actual dependencies. In order to represent such highly asymmetric matrix structures, the ordering algorithm developed here is based on the partitioning of a *bipartite* graph model of the sparse matrix, as developed in the next section.

3.1 Graph Model

An $n \times n$ structurally unsymmetric matrix A can be represented by an ordered directed graph $\vec{G} = (X, \vec{E}, a)$, where X is the vertex set, $\vec{E}$ is the edge set, and a is an ordering of the vertices. The vertices in this representation correspond to the locations of diagonal elements of A, and the edges correspond to the locations of nonzero off-diagonal elements [see Coon and Stadtherr (1995) for a schematic of such a graph and the corresponding matrix]. This representation requires the existence of a zero-free diagonal, which can be found using a maximum transversal algorithm such as the one given by Duff (1981). An important concept is that of a *disconnecting set* B, which is a set of edges whose removal from $\vec{G}$ leaves two disjoint subgraphs $\vec{G}_1$ and $\vec{G}_2$. A disconnecting set B of $\vec{G}$ is *minimal* if no proper subset of B disconnects $\vec{G}$. The identification of a minimal disconnecting subset is equivalent to partitioning the corresponding sparse matrix into two diagonal blocks, with the off-diagonal blocks containing a minimal number of nonzeros. This is an NP-complete problem and so, especially for large problems, it is usually necessary to seek an approximately minimal disconnecting subset using heuristic or stochastic approaches.

Another useful representation for an unsymmetric sparse matrix is a directed bipartite graph $\vec{G} = (R, C, \vec{E}(M))$, consisting of a row vertex set R, a column vertex set C, a complete matching M corresponding to a zero-free diagonal, and an edge set $\vec{E}$ such that a vertex in C is out-adjacent to a vertex in R if an off-diagonal nonzero element exists in the corresponding row and column, and a vertex in R is out-adjacent to a vertex in C for each diagonal element of the matrix [see Coon and Stadtherr (1995) for a schematic of such a graph and the corresponding matrix]. A key idea is that of a *net*, which can be viewed as a column vertex together with all the vertices which are adjacent to it. For a given column, these adjacent vertices correspond to a set of rows, all of which have a nonzero element in the given column. A net is said to be cut with respect to a partitioning into subgraphs if any two of its vertices belong to different subgraphs after the partitioning; this corresponds to a column having nonzero elements on both sides of a matrix row partition. If a disconnecting set for the graph of the matrix is found such that the number of nets cut by the partition is minimized, this corresponds to a bordered block-diagonal form with two diagonal blocks and borders of minimum size (the cut nets correspond to border columns). The problem of finding a disconnecting set for a bipartite graph which minimizes the number of nets cut is known to be NP-complete, so algorithms attempt to select a good approximation to such a partitioning using heuristics.

3.2 The Min-Net-Cut Approach

This algorithm represented a generalization of previous graph partitioning strategies, and was developed by Coon and Stadtherr (1995) for dealing with unsymmetric process simulation matrices. The directed bipartite graph model described above was used. The kernel of the MNC methods is the identification of a disconnecting set which bisects the bipartite graph $\vec{G}$ into two partitions $\vec{G}_1$ and $\vec{G}_2$, such that the number of nets cut is approximately minimized. This disconnecting set is determined by repeatedly choosing vertex pairs to move or swap across the partition boundary in order to reduce the number of nets cut by the partitioning. First, vertices (both column and row) are ranked according to their *gain*, that is, the decrease in the number of nets cut when this vertex is moved to the other partition. Two heuristics are used to choose vertex pairs: the *free match* criterion and the *free swap* criterion. The free match criterion pairs a row vertex with its matching column vertex in the same partition and moves both to the other partition. This criterion can only find symmetric permutations of row and column vertices, since it does not consider alternate matchings. Searching all alternate matchings is too time consuming, however, and so the free swap criterion searches a cycle of length four in the directed bipartite graph for a free vertex, and exchanges it with the vertex with the highest gain from the other partition. The matching can then be easily reconstructed. This partitioning procedure is applied recursively to the subgraphs $\vec{G}_1$ and $\vec{G}_2$ until a stopping criterion is reached. This yields a bordered block-diagonal form with a structured border, a form which can also be interpreted as a nested block-tridiagonal form whose off-diagonal blocks have few nonzero columns. The complete details of MNC are given by Coon and Stadtherr (1995), who describe a family of methods based on this approach, differing in the order in which the heuristics are applied and other implementation details.

Since the MNC methods begin with a complete matching (nonzero diagonal) and seek to maintain a complete matching during the reordering, the diagonal blocks they produce also have a complete matching, and thus are structurally nonsingular. This does not guarantee numerical nonsingularity, however, and so in factoring the diagonal blocks some pivoting strategy must be used, with rows and columns that cannot be used in pivots moved to the borders. Since a pivoting strategy must be used anyway, starting with structurally nonsingular diagonal blocks is not strictly necessary. There may be some disadvantages to starting with structurally singular blocks, since this will likely mean an increase in the number of rows and columns that must be moved out to the borders during pivoting, thus further increasing the size of the interface problem. However, removing the complete matching constraint may allow one to find a bordered block-diagonal form that has a smaller border in the first place. In the reordering algorithm described here we remove the complete matching constraint, resulting in an algorithm that is much simpler and faster than the original MNC approach. Furthermore, because the new algorithm is more flexible than MNC, since the number of possible row moves is much greater, a larger number of partitions can often be created. This new MNC-type ordering is referred to as GPA-SUM (**G**raph-**P**artitioning **A**lgorithm for **S**parse **U**nsymmetric **M**atrices.

3.3 Graph-Partitioning Algorithm for Sparse Unsymmetric Matrices (GPA-SUM)

This algorithm is essentially a simplification and generalization of the MNC ordering of Coon and Stadtherr (1995). As with the MNC ordering, the kernel of the algorithm attempts to identify a disconnecting subset that minimizes the number of nets cut by the bisection. However, since the GPA-SUM algorithm seeks to create a bordered block-diagonal form with diagonal blocks that are not necessarily nonsingular, simplifications can be made to the original MNC algorithm.

The first major difference between GPA-SUM and MNC is the removal of the complete matching constraint. Finding a partitioning without this constraint is equivalent to partitioning an *undirected* bipartite graph $G(R, C, E)$ in which each edge signifies one nonzero element of the unsymmetric matrix A. Removing this constraint means that no initial zero-free diagonal need be found. For large matrices, finding a maximum transversal can be a quite significant computational expense. With the removal of the matching constraint, the core problem can be treated as one of bisecting the row vertices only, with columns assigned to partitions after row partitioning has been completed. We therefore redefine a cut net as simply one whose row vertices lie in more than one partition (without regard to its column vertex). This greatly simplifies the computation of the gain of a row vertex, which is one of the most time consuming steps of the MNC approach.

GPA-SUM also has a simpler algorithm for choosing row vertices to move across a partition boundary. The gain of each row vertex, that is the change in the number of nets cut when this row is moved across the partition boundary, is computed and stored in a linked list structure. This structure can be searched very efficiently in order to find the vertex with the highest gain, and the corresponding row is then moved into the other partition. These moves continue until no more rows exist which can improve the number of cut nets, or until the ratio of the sizes of the two partitions fails to meet a specific tolerance. The ratio of the order of the two newly created partitions is controlled by the parameter *minratio*, as in MNC, since keeping the blocks approximately equal in size can be important for load balancing when solving using parallel methods. The optimal value of this parameter depends on the relative costs of communication and computation on a specific machine architecture.

Once the rows have been bisected into two partitions, column assignments can be made. A column is assigned to a partition if all of the nonzero elements in that column are located in that partition. This corresponds to an uncut net in the bipartite graph. Columns corresponding to cut nets will become part of the column border. As discussed previously, it is generally important to try to limit the number of cut columns created by the partitioning, perhaps even at the expense of creating fewer diagonal blocks, since this has a direct impact on the size of the interface problem. The parameter *cutratio* allows the number of cut columns created by GPA-SUM to be limited, by putting a tolerance on the ratio of cut columns to total columns. Since this parameter is only checked globally after a full pass through the recursive algorithm, the actual percentage of cut columns will tend to be somewhat higher than the specified value, and so this parameter should be set lower than the actual percentage of cut columns desired. In the MNC methods, the *cutratio* parameter is used somewhat differently, being applied locally to each partition created, rather than globally. This has effect of often making the actual global percentage of cut columns significantly less than the specified *cutratio*.

Like MNC, GPA-SUM recursively partitions the subgraphs created by each partitioning, using a breadth-first recursion. The procedure used for choosing row vertices to move across the partition boundary in GPA-SUM can create too many partitions for efficient solution on a given architecture if stopping criteria similar to those used in the MNC algorithm are used. Thus, a parameter *maxpart* was introduced to limit the total number of partitions created.

A basic outline of the steps involved in the GPA-SUM ordering is as follows:

Begin k-th level of recursion:

1. Attempt bisection of each (unlocked) partition found on the $(k-1)$-th level of recursion. Do for each such partition:

   (a) Create linked lists, define the initial partition point and disconnecting edge set, and compute gains for each row vertex in the partition.

   (b) Using the linked lists, find the (unlocked) row vertex in each partition that has the highest positive gain. Exchange this pair of vertices between partitions. If one partition has no (unlocked) row vertices with positive gain, then in the other partition, choose the single (unlocked) row vertex with the highest positive gain and move it across the partition boundary. Temporarily lock any row vertices moved so that they are not eligible to be moved again while processing the current partition. If neither partition has any (unlocked) row vertices with positive gain, then go to (e).

   (c) Check *minratio*; if too many single vertex moves made, go to (e).

(d) Update linked lists to account for the moved vertices. Return to (b).

(e) If no row vertices moved during this partitioning step, lock this partition. No attempts will be made on subsequent levels of recursion to bisect this partition.

(f) If bisection successful (row vertices were moved) and column ordering desired, assign columns to partitions and determine which nets have been cut.

2. Check *cutratio* and *maxpart*. If either parameter is reached or exceeded, or if all partitions have been locked, then end. Otherwise, unlock row vertices that were temporarily locked in 1(b), and continue recursive partitioning with $k = k + 1$.

As a small example to illustrate how the GPA-SUM algorithm works, consider the matrix of Figure 1. In this figure, the row vertex gains are listed after each row. A positive row vertex gain indicates that if this row vertex is moved to the other partition, then the total number of nets cut will decrease by this value. Conversely, a negative row vertex gain indicates that the number of nets cut will increase if this row is moved across the partition boundary. For instance, row 1 has a gain of 1 because moving row vertex 1 across the partition boundary causes net 1 to become uncut, while net 10 remains cut, for a net change of one less cut net. In this example, the row gains show that an exchange of rows 4 and 10 can be made which will decrease the number of nets cut more than any other exchange. After this exchange, the row vertex gains are recomputed for the affected rows (all of the rows in this small example must be recomputed, but in the bipartite graph corresponding to a large, sparse matrix, very few row vertex gains must actually be updated).

The result of exchanging row vertices 4 and 10 is shown in Figure 2. Here the row vertex gains for rows 10 and 4 are not computed since these rows are locked, that is, they are not allowed to move again during this bisection step. A second exchange can now be made which will decrease the number of nets cuts, namely row 5 and row 9. Again, all affected row gains must be recomputed after this exchange, leading to the matrix of Figure 3. Since no more row vertices can be moved which will decrease the number of nets cut, the algorithm next assigns uncut columns to the appropriate partitions, and designates cut columns as belonging to the border, as indicated in Figure 4 by marking the nonzeros in columns 5 and 6 with a B. The algorithm then moves to the second ($k = 2$) level of the recursive partitioning.

For each of the partitions created in the previous level of recursion ($k = 1$), another bisection is now attempted. Beginning with the first (topmost) partition in Figure 3, an initial partition point is defined and row vertex gains computed as shown in Figure 4. It should be emphasized that, in computing the gains, the nets cut on previous levels of recursion are not considered. Row vertices 1 and 9 have the highest gains (ties are broken by whichever row vertex appears first in the linked list, which in this case is the row with the lower row index), and so these two rows are interchanged. After this exchange, the row gains for the first partition are recomputed, resulting in the structure of Figure 5. At this point, it is clear that no exchanges of rows are possible in the first partition. However, row 3 can be moved across the upper partition boundary as a single row move, with the result shown in Figure 6. After this, no further moves can be made in the first partition and column 10 is still cut so it is designated a border column.

Next the second (bottommost) partition is processed, with the initial partition point and row vertex gains as shown in Figure 6. Here, an exchange of rows 5 and 12 is found, leading to the structure in Figure 7. Looking at the recomputed row gains for the second partition, it is clear that no further row moves are possible, and so the columns are then assigned to partitions.

At this point if *cutratio* $= 0.25$ or *maxpart* $= 4$, the recursion would stop since these tolerances have been reached. If a next level of recursion was attempted, there would be no possible row moves in any partition, causing all partitions to be locked and thus end the recursion. The final structure is shown in Figure 8 as a column-bordered block-diagonal form. Columns 5, 6 and 10 will belong to the interface problem; the rows belonging to the interface problem will be determined by pivoting within each of the four partitions. This example clearly shows how a block-diagonal structure is formed by the GPA-SUM algorithm. While 3 of the 12 nets in this case are cut (compared to 10 for the original matrix), larger examples often give orderings with a much smaller percentage of cut nets and more diagonal blocks.

4 Results and Discussion

A large number of test problems were used in this study to test the capabilities of the GPA-SUM ordering. Table 1 gives information about each test matrix, including the order n, the number of nonzeros, the symmetry ratio, the matrix source, and a brief problem description. The symmetry ratio s is the number of matched off-diagonal nonzeros (i.e., nonzeros $a_{ij}, i \neq j$, for which a_{ji} is also nonzero) divided by the total number of off-diagonal nonzeros. Thus $s = 1$ indicates a completely symmetric pattern, while $s = 0$ shows complete asymmetry. It can be seen that, except for the last two problems, which are not process engineering examples, these matrices have a very high degree of asymmetry, almost complete asymmetry in many cases. With five exceptions, all the matrices arise from applications in process engineering, using SEQUEL (Zitney and Stadtherr, 1988), SPEEDUP (Aspen Technology, Inc.) and ASPEN PLUS (Aspen Technology, Inc.), as well as three distillation problems (*west**) from the Harwell/Boeing collection. The other five problems are also from the Harwell/Boeing collection, and represent a variety of applications outside of process engineering.

Each matrix was reordered using the GPA-SUM and MNC orderings, with selected results summarized in Table 2. The MS variant of MNC, a top performer in the study by Coon and Stadtherr (1995), was used. The results of interest include:

1. The number of diagonal blocks identified (NB).

2. The percentage of nets cut (%NC). This is the number of columns in the border expressed as a percentage of total number of columns, and is directly related to the size of the interface problem. Typically, the order of the interface problem will become somewhat larger during the numerical solution process, due to pivoting in the diagonal blocks, as described above. Because the diagonal blocks created by GPA-SUM may be structurally singular, while those created by MNC are always structurally nonsingular, the increase in the size of the interface problem due to pivoting may be larger when the GPA-SUM ordering is used.

3. The reordering time (RT). This is given as CPU seconds on a CRAY J-90.

4. The relative size of the diagonal blocks. This is expressed in terms of the maximum block ratio (MBR), which is the ratio of the order of the largest block to the mean block order (n/NB).

The relative importance of the structural results depends on the target machine architecture as well as on the details of the parallel solver. For example, if the parallel solver is designed to solve the interface problem on a single processor, then the percentage of nets cut will likely be a critical concern. On the other hand, if the interface problem can be solved effectively in parallel, then its size will be of less concern. GPA-SUM is designed to be tunable, so that structures suitable for different machine architectures and parallel solvers can be created. Thus, we do not attempt to evaluate the reorderings in the context of any particular solver or architecture.

Both GPA-SUM and MNC have the tuning parameters *cutratio* (CR), which adjusts the allowable number of cut nets and thus controls the size of the interface matrix, and *minratio* (MR), which controls the relative sizes of the vertex sets after partitioning. The GPA-SUM results are shown for two sets of parameter settings: the first is designed to keep the interface matrix small, while the second is designed to generate a larger number of partitions. As noted by Coon and Stadtherr (1995), the performance of MNC is not very sensitive to values of the tuning parameters. Since, as explained above, the *cutratio* parameter is used differently in GPA-SUM and MNC, the values used are not directly comparable. In MNC, the actual fraction of nets cut is generally significantly less than *cutratio*, while in GPA-SUM it is often significantly higher than *cutratio*. Thus, the *cutratio* values of 0.20 in MNC and 0.07 in one set of GPA-SUM runs can be regarded as roughly comparable, and do tend to yield roughly similar results in terms of percent nets cut.

The results in Table 2 show the trade-off between the number of diagonal blocks created by the algorithm and the number of nets cut by the partitioning, and demonstrate how by varying *cutratio* and *minratio* this trade-off can be tuned for a particular target architecture or parallel solver. For nearly every matrix in the test set, matrix structures with more or fewer diagonal blocks could be created by GPA-SUM from just these two sets of tuning parameters. For example, in the *extr1b_** series of matrices, either 8 or 16 diagonal blocks could be obtained using these parameter values. By further increasing the *cutratio* parameter, even more partitions (and a larger interface problem) can be created, and by reducing *cutratio* even fewer partitions (and a smaller interface matrix) can be achieved. This is significantly different from the MNC algorithm, which is insensitive to the values of the tuning parameters, and which is only able to create structures with a small number of partitions on these problems. By relaxing the matching constraint in MNC, the GPA-SUM approach is able to identify a much wider range of structures of interest for parallel computation.

As expected, eliminating the need to initialize and maintain a complete matching provides a significant running time advantage for GPA-SUM. The results in Table 2 show that GPA-SUM runs 20-80 times faster than MNC, and that this difference increases as the order of the matrix increases. While some of the reordering time differences are due to the creation of a different number of partitions, it is clear that the GPA-SUM algorithm is many times faster on a per-partition basis. Results on the series of matrices *lhr**, which have similar structures but vary in order, show the increase in reordering time for the two algorithms as the order of the matrix increases. For an increase in size of about a factor of four, the GPA-SUM reordering time increases (roughly linearly) by only about six, while the MNC reordering time increases by a factor of 16.5. The GPA-SUM reordering is also creating more than twice as many partitions for these matrices.

The relative size of the diagonal blocks is important when load-balancing considerations are taken into account. A maximum block ratio (MBR) much greater than one indicates that one diagonal block is significantly larger than the mean block size and thus could create a bottleneck in a parallel solution scheme. The results show that for most cases, the blocks created by the GPA-SUM ordering are more nearly equal in size than those created by MNC. An MNC ordering which creates more diagonal blocks is also more likely to have a block size imbalance, since each partitioning could fail for some subpartitions, leaving much larger blocks in the matrix. This is overcome in the GPA-SUM algorithm by removing the matching constraints on row vertex exchanges. This allows more of the row vertex moves to be exchanges between partitions, rather than moves of a single vertex. Since it is the latter type of move that changes the size of the partitions, block size balance can be better maintained by GPA-SUM.

5 Concluding Remarks

The experiments conducted in this work have shown the effectiveness of the GPA-SUM method for creating bordered block-diagonal forms for sparse matrices resulting from process engineering problems. This graph-partitioning algorithm creates orderings comparable to those determined with previously used methods, but does so in a much shorter time. Furthermore, the GPA-SUM algorithm is tunable to create matrix structures with more diagonal blocks, or to reduce the size of the interface matrix while still determining a useful partitioning. This capability allows the algorithm to be useful when orderings are needed for parallel solvers on a variety of parallel architectures. The nearly two order of magnitude savings in reordering time seen on the large problems will be of particularly significance in dealing with even larger industrial-scale problems, especially if real-time capabilities are desired.

Acknowledgments – This work has been supported by the National Science Foundation under Grants DMI-9322682 and DMI-9696110. We would like to thank Cray Research for providing computer time, and the Albert-Einstein Institute in Potsdam, Germany for the use of its computer resources. We also acknowledge the support of the National Center for Supercomputing Applications at the University of Illinois and Aspen Technology, Inc.

References

Abbott, K. A., B. A. Allan, and A. W. Westerberg, Global preordering for Newton equations using model hierarchy. *AIChE J.*, **43**, 3193–3204 (1997).

Choi, H. and D. B. Szyld, Threshold ordering for preconditioning nonsymmetric problems with highly varying coefficients. Technical Report 96-51, Dept. of Mathematics, Temple University, Philadelphia, PA (1996).

Cofer, H. N. and M. A. Stadtherr, Reliability of iterative linear solvers in chemical process simulation. *Comput. Chem. Engng*, **20**, 1123–1132 (1996).

Coon, A. B. and M. A. Stadtherr, Generalized block-tridiagonal matrix orderings for parallel computation in process flowsheeting. *Comput. Chem. Engng*, **19**, 787-805 (1995).

Duff, I.S., On algorithms for obtaining a maximum transversal. *ACM Trans. Math. Softw.*, **7**, 315-330 (1981).

Fiduccia, C. M. and R. M. Mattheyses, A linear-time heuristic for improving network partitions. In *Proc. 19th ACM-IEEE Design Automation Conf.*, Las Vegas, ACM (1982).

Gallivan, K. A., B. A. Marsolf, and H. A. G. Wijshoff, Solving large nonsymmetric sparse linear systems using MCSPARSE. *Parallel Comput.*, **22**, 1291–1333 (1996).

Karpis, G. and V. Kumar, Multilevel k-way partitioning scheme for irregular graphs. Technical Report 95-064, Dept. of Computer Science, Univ. of Minnesota, Minneapolis, MN (1995).

Kernighan, B. W. and S. Lin, An efficient heuristic procedure for partitioning graphs. *Bell System Tech. J.* **49**, 291-307 (1970).

Leiserson, C. E. and J. G. Lewis, Orderings for parallel sparse symmetric factorization. In Rodrigue, G., editor, *Parallel Processing for Scientific Computing*, pages 27–31. SIAM, Philadelphia, PA (1989).

Mallya, J. U., S. E. Zitney, S. Choudhary, and M. A. Stadtherr, A parallel frontal solver for large scale process simulation and optimization. *AIChE J.*, **43**, 1032–1040 (1997a).

Mallya, J. U., S. E. Zitney, S. Choudhary, and M. A. Stadtherr, A parallel block frontal solver for large scale process simulation: Reordering effects. *Comput. Chem. Engng*, **21**, S439–S444 (1997b).

O'Neil, J. and D. B. Szyld, A block ordering method for sparse matrices. *SIAM J. Sci. Stat. Comput.*, **11**, 811–823 (1990).

Schweikert, D. G. and B. W. Kernighan, A proper model for the partitioning of electrical circuits. In *Proc. 9th ACM-IEEE Design Automation Workshop*, ACM, Dallas, (1972).

Westerberg, A. W. and T. J. Berna, Decomposition of very large scale Newton-Raphson based flowsheeting problems. *Comput. Chem. Engng.* **2**, 61 (1978).

Zitney, S. E., Sparse matrix methods for chemical process separation calculations on supercomputers. In *Proc. Supercomputing '92*, pages 414–423. IEEE Press, Los Alamitos, CA (1992).

Zitney, S. E., L. Brüll, L. Lang, and R. Zeller, Plantwide dynamic simulation on supercomputers: Modeling a Bayer distillation process. *AIChE Symp. Ser.*, **91**(304), 313–316 (1995).

Zitney, S. E., K. V. Camarda and M. A. Stadtherr, Impact of supercomputing in simulation and optimization of process operations. In Proc. Second International Conference on Foundations of Computer-Aided Process Operations (eds. D. W. T. Rippen, J. C. Hale, and J. F. Davis), CACHE, Austin, TX, 463-468 (1994).

Zitney, S. E. and M. A. Stadtherr, Computational experiments in equation-based chemical process flowsheeting. *Comput. Chem. Engng*, **12**, 1171–1186 (1988).

Zitney, S. E. and M. A. Stadtherr, Frontal algorithms for equation-based chemical process flowsheeting on vector and parallel computers. *Comput. Chem. Engng*, **17**, 319–338 (1993).

Table 1: Statistics for example matrices. See text for definition of symmetry ratio and further discussion.

Name	Order	Nonzeros	Symmetry Ratio	Source	Description
ngc	1235	16868	.0336	SEQUEL	natural gas plant
lhr	1477	18592	.00732	SEQUEL	light hydrocarbon recovery
mult1	612	4459	.0103	SEQUEL	multiunit flowsheet
mult2	714	5001	.0196	SEQUEL	multiunit flowsheet
mult3	526	5363	.0329	SEQUEL	multiunit flowsheet
cyclo1	517	2420	.0158	SEQUEL	cyclohexane plant
beef	1197	12070	.00680	SEQUEL	multiunit flowsheet
lhr4	4101	82682	.0152	SEQUEL	light hydrocarbon recovery
lhr7	7338	156508	.0174	SEQUEL	light hydrocarbon recovery
lhr10	10672	232633	.00879	SEQUEL	light hydrocarbon recovery
lhr11	10964	233741	.00820	SEQUEL	light hydrocarbon recovery
lhr14	14270	307858	.00662	SEQUEL	light hydrocarbon recovery
lhr17	17576	381975	.00151	SEQUEL	light hydrocarbon recovery
segm3_m1	1045	4963	.00524	SPEEDUP	multistage reactor
segm3_m2	1045	5131	.00546	SPEEDUP	multistage reactor
extr1b_m1	2836	11404	.00386	SPEEDUP	extractive distillation
extr1b_m2	2836	11579	.00380	SPEEDUP	extractive distillation
extr1b_m3	2836	9227	.00780	SPEEDUP	extractive distillation
hydr1c_m1	5308	23752	.00413	SPEEDUP	two-column separation
hydr1c_m2	5308	23956	.00409	SPEEDUP	two-column separation
hydr1c_m3	5308	19892	.00332	SPEEDUP	two-column separation

Table 1: (continued)

Name	Order	Nonzeros	Symmetry Ratio	Source	Description
traycalc	1145	20296	.117	ASPEN PLUS	reactive distillation
userupp	1269	22508	.106	ASPEN PLUS	reactive distillation
v3	1078	16937	.0918	ASPEN PLUS	distillation process
v10	1148	15729	.0602	ASPEN PLUS	distillation process
v13	834	9713	.0541	ASPEN PLUS	distillation process
mpex2	848	11413	.0402	ASPEN PLUS	distillation process
mpex3	2473	46503	.0617	ASPEN PLUS	distillation process
mpex4	2478	44075	.0537	ASPEN PLUS	distillation process
sumb	523	4998	.0513	ASPEN PLUS	distillation process
uosb	523	4998	.0513	ASPEN PLUS	distillation process
gre1107	1107	5664	0	H/B	computer system
west0989	989	3537	.0181	H/B	distillation process
west1505	1505	5445	.00110	H/B	distillation process
west2021	2021	7353	.00327	H/B	distillation process
mpmult1	2023	31894	.0486	ASPEN PLUS	distillation process
bp_1000	822	4661	.00944	H/B	LP basis
mahindas	1258	7862	.0166	H/B	economic model
lns3937	3937	25407	.850	H/B	fluid flow model
sherman5	3312	20793	.739	H/B	reservoir simulation

Table 2: Statistics for example problems after reordering. Columns are the number of diagonal blocks (NB), the percentage of cut nets (%NC), the reordering time (RT) in seconds on a CRAY J-90, and the ratio (MBR) of the order of the largest block to the mean block order. Parameters are *cutratio* (CR) and *minratio* (MR). See the text for further discussion.

	GPA-SUM								MNC			
	CR=0.07 MR=0.45				CR=0.15 MR=0.25				CR=0.20 MR=0.45			
Name	NB	%NC	RT	MBR	NB	%NC	RT	MBR	NB	%NC	RT	MBR
ngc	2	9.15	0.0928	1.023	4	18.2	0.167	1.049	4	18.5	4.9	1.147
lhr	2	10.8	0.111	1.036	4	17.0	0.200	1.067	4	19.0	5.7	1.113
mult1	2	15.0	0.0368	1.085	2	15.0	0.0367	1.085	2	12.6	1.2	1.020
mult2	2	20.9	0.0326	1.053	4	23.9	0.0655	1.092	4	21.4	1.5	1.182
mult3	2	28.7	0.0402	1.106	2	28.1	0.0417	1.118	1	-	-	-
cyclo1	2	8.12	0.0161	1.075	4	15.5	0.0318	1.161	3	11.4	0.80	1.648
beef	2	10.0	0.0640	1.026	4	15.9	0.122	1.056	3	10.6	2.9	1.649
lhr4	4	7.85	0.734	1.034	8	15.9	1.32	1.102	3	6.91	35.0	1.496
lhr7	8	11.7	1.66	1.070	16	18.1	3.22	1.110	4	6.90	100.2	1.093
lhr10	8	9.59	2.33	1.060	16	15.8	4.71	1.079	10	9.65	207.7	2.516
lhr11	8	9.87	2.66	1.069	16	16.3	4.78	1.118	10	9.64	242.9	2.488
lhr14	8	9.01	3.49	1.060	16	13.4	6.28	1.106	6	5.40	393.8	1.588
lhr17	8	7.44	4.20	1.063	16	11.8	7.74	1.082	6	3.48	581.3	1.468
segm3_m1	4	12.0	0.0521	1.087	8	17.0	0.0949	1.10	3	11.6	2.20	1.418
segm3_m2	4	12.7	0.0510	1.072	8	18.3	0.0973	1.110	3	11.2	2.30	1.418
extr1b_m1	8	8.36	0.152	1.038	16	13.6	0.276	1.100	4	5.47	13.1	1.066
extr1b_m2	8	8.53	0.138	1.041	16	13.4	0.278	1.100	4	5.61	12.0	1.078
extr1b_m3	8	6.77	0.136	1.029	16	11.0	0.248	1.117	4	4.80	12.1	1.066
hydr1c_m1	8	9.19	0.318	1.099	16	12.9	0.554	1.185	4	3.43	41.0	1.088
hydr1c_m2	8	9.23	0.309	1.099	16	12.8	0.557	1.182	4	2.60	42.4	1.057
hydr1c_m3	8	7.42	0.281	1.153	16	10.5	0.506	1.248	6	4.97	43.9	1.422

Table 2: (continued)

	GPA-SUM								MNC			
	CR=0.07 MR=0.45				CR=0.15 MR=0.25				CR=0.20 MR=0.45			
Name	NB	%NC	RT	MBR	NB	%NC	RT	MBR	NB	%NC	RT	MBR
traycalc	4	13.4	0.175	1.153	8	30.7	0.315	1.244	7	15.7	5.0	1.706
userupp	4	9.77	0.181	1.151	8	21.6	0.342	1.179	7	15.2	5.8	1.539
v3	4	12.4	0.146	1.180	8	26.9	0.266	1.210	8	20.0	3.5	2.019
v10	4	8.8	0.133	1.192	8	20.3	0.247	1.289	8	20.1	3.8	2.404
v13	4	12.9	0.0890	1.213	8	25.5	0.160	1.429	4	12.7	1.8	1.204
mpex2	4	11.0	0.112	1.212	8	21.8	0.188	1.453	3	7.43	1.9	1.649
mpex3	8	12.5	0.522	1.174	16	28.7	0.924	1.255	13	16.1	18.0	1.561
mpex4	8	13.5	0.489	1.204	16	28.0	0.883	1.214	16	18.7	17.7	1.801
sumb	4	13.0	0.0447	1.208	8	30.2	0.0894	1.576	3	8.6	0.90	1.646
uosb	4	13.0	0.0446	1.208	8	30.2	0.0892	1.576	3	8.6	0.90	1.646
gre1107	2	34.8	0.0350	1.006	2	34.8	0.0350	1.006	2	11.8	1.5	1.059
	CR=0.07 MR=0.45				CR=0.25 MR=0.25				CR=0.20 MR=0.45			
west0989	2	15.5	0.0224	1.011	15	23.4	0.0931	1.926	3	10.4	2.7	1.596
west1505	2	15.1	0.0344	1.010	16	22.7	0.142	1.095	3	9.17	4.6	1.627
west2021	2	15.0	0.0457	1.153	16	21.9	0.186	1.244	3	9.25	7.7	1.706
mpmult1	8	15.3	0.482	1.269	16	20.7	0.854	1.297	12	16.5	11.6	1.602
bp_1000	2	20.2	0.0366	1.063	4	35.6	0.0806	1.348	2	14.1	3.0	1.190
mahindas	2	25.0	0.142	1.083	4	35.8	0.237	1.298	2	12.8	4.1	1.046
lns3937	2	21.7	0.345	1.100	4	55.2	0.560	1.102	6	18.0	13.3	1.722
sherman5	2	18.8	0.175	1.101	4	41.8	0.328	1.167	4	22.7	4.1	2.054

Figure Captions

Figure 1. Original occurrence matrix for example used in text.

Figure 2. Partially reordered occurrence matrix for example used in text.

Figure 3. Partially reordered occurrence matrix for example used in text.

Figure 4. Partially reordered occurrence matrix for example used in text. Nonzeros marked with a B indicate a column to be assigned to the border.

Figure 5. Partially reordered occurrence matrix for example used in text. Nonzeros marked with a B indicate a column to be assigned to the border.

Figure 6. Partially reordered occurrence matrix for example used in text. Nonzeros marked with a B indicate a column to be assigned to the border.

Figure 7. Partially reordered occurrence matrix for example used in text. Nonzeros marked with a B indicate a column to be assigned to the border.

Figure 8. Final reordered occurrence matrix for example used in text, showing a column-bordered block-diagonal form.

	1	2	3	4	5	6	7	8	9	10	11	12	Gain
1	X									X			1
2							X			X			1
3		X				X							1
4								X				X	2
5						X					X		1
6				X					X				2
7					X	X		X					0
8			X			X					X		0
9					X		X			X			1
10	X	X		X		X			X				4
11			X								X		-1
12					X	X						X	0

Figure 1: Original occurrence matrix for example used in text.

	1	2	3	4	5	6	7	8	9	10	11	12	Gain
1	X									X			-2
2							X			X			0
3		X				X							-1
5						X					X		1
6				X					X				-2
10	X	X		X		X			X				L
4								X				X	L
7					X	X		X					-2
8			X			X					X		-1
9					X		X			X			2
11			X								X		-1
12					X	X						X	-2

Figure 2: Partially reordered occurrence matrix for example used in text.

	1	2	3	4	5	6	7	8	9	10	11	12	Gain
1	X									X			-2
2							X			X			-2
3		X				X							-1
6				X					X				-2
10	X	X		X		X			X				L
9					X		X			X			L
5						X					X		L
4								X				X	L
7					X	X		X					-1
8			X			X					X		-2
11			X								X		-2
12					X	X						X	-1

Figure 3: Partially reordered occurrence matrix for example used in text.

$$
\begin{array}{r|cccccccccccc|c}
 & 1 & 2 & 4 & 7 & 9 & 10 & 5 & 6 & 3 & 8 & 11 & 12 & \text{Gain} \\
1 & X & & & & & X & & & & & & & 1 \\
2 & & & & X & & X & & & & & & & 1 \\
3 & & X & & & & & & B & & & & & 1 \\
\cline{2-13}
6 & & & X & & X & & & & & & & & -2 \\
10 & X & X & X & & X & & & B & & & & & 0 \\
9 & & & & X & & X & B & & & & & & 2 \\
\cline{2-13}
5 & & & & & & & & B & & & X & & \\
4 & & & & & & & & & & X & & X & \\
7 & & & & & & & B & B & & X & & & \\
8 & & & & & & & & B & X & & X & & \\
11 & & & & & & & & & X & & X & & \\
12 & & & & & & & B & B & & & & X &
\end{array}
$$

Figure 4: Partially reordered occurrence matrix for example used in text. Nonzeros marked with a B indicate a column to be assigned to the border.

	1	2	4	7	9	10	5	6	3	8	11	12	Gain
2				X		X							-1
3		X						B					1
9				X		X	B						L
1	X					X							L
6			X		X								-2
10	X	X	X		X			B					-2
5								B			X		
4										X		X	
7							B	B		X			
8								B	X		X		
11									X		X		
12							B	B				X	

Figure 5: Partially reordered occurrence matrix for example used in text. Nonzeros marked with a B indicate a column to be assigned to the border.

$$
\begin{array}{c|cccccccccccc|c}
 & 1 & 2 & 4 & 7 & 9 & 10 & 5 & 6 & 3 & 8 & 11 & 12 & \text{Gain} \\
2 & & & & X & & B & & & & & & & -1 \\
9 & & & & X & & B & B & & & & & & L \\
\cline{2-13}
3 & & X & & & & & & B & & & & & L \\
1 & X & & & & & B & & & & & & & L \\
6 & & & X & & X & & & & & & & & -2 \\
10 & X & X & X & & X & & & B & & & & & -4 \\
\cline{2-13}
5 & & & & & & & & B & & & X & & 1 \\
4 & & & & & & & & & & X & & X & 0 \\
7 & & & & & & & B & B & & X & & & -1 \\
\cline{2-13}
8 & & & & & & & & B & X & & X & & -1 \\
11 & & & & & & & & & X & & X & & -1 \\
12 & & & & & & & B & B & & & & X & 1
\end{array}
$$

Figure 6: Partially reordered occurrence matrix for example used in text. Nonzeros marked with a B indicate a column to be assigned to the border.

$$\begin{array}{c|cccccccccccc|c}
 & 1 & 2 & 4 & 7 & 9 & 10 & 5 & 6 & 3 & 8 & 11 & 12 & \mathsf{Gain} \\
2 & & & & X & & B & & & & & & & \\
9 & & & & X & & B & B & & & & & & \\
\hline
3 & & X & & & & & & B & & & & & \\
1 & X & & & & & B & & & & & & & \\
6 & & & X & & X & & & & & & & & \\
10 & X & X & X & & X & & & B & & & & & \\
\hline
4 & & & & & & & & & & X & & X & -2 \\
7 & & & & & & & B & B & & X & & & -1 \\
12 & & & & & & & B & B & & & & X & L \\
\hline
5 & & & & & & & & B & & & X & & L \\
8 & & & & & & & & B & X & & X & & -2 \\
11 & & & & & & & & & X & & X & & -2
\end{array}$$

Figure 7: Partially reordered occurrence matrix for example used in text. Nonzeros marked with a B indicate a column to be assigned to the border.

$$
\begin{array}{c|ccccccccc|ccc}
 & 7 & 1 & 2 & 4 & 9 & 8 & 12 & 3 & 11 & 10 & 5 & 6 \\
2 & X & & & & & & & & & B & & \\
9 & X & & & & & & & & & B & B & \\
3 & & & X & & & & & & & & & B \\
1 & & X & & & & & & & & B & & \\
6 & & & & X & X & & & & & & & \\
10 & & X & X & X & X & & & & & & & B \\
4 & & & & & & X & X & & & & & \\
7 & & & & & & X & & & & & B & B \\
12 & & & & & & & X & & & & B & B \\
5 & & & & & & & & & X & & & B \\
8 & & & & & & & & X & X & & & B \\
11 & & & & & & & & X & X & & &
\end{array}
$$

Figure 8: Final reordered occurrence matrix for example used in text, showing a column-bordered block-diagonal form.