Matrix Reordering Effects on a Parallel Frontal Solver for Large Scale Process Simulation

J. U. Mallya, S. E. Zitney*, S. Choudhary
Cray Research, Inc.
655-E Lone Oak Drive
Eagan, MN 55121, USA

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

October 1997
(revised, June 1998)

Originally presented at
PSE '97—ESCAPE-7
Trondheim, Norway, May 25-29, 1997

*Current address: AspenTech UK Ltd., Castle Park, Cambridge CB3 0AX, England
†Author to whom all correspondence should be addressed. Fax: (219) 631-8366; E-mail: markst@nd.edu

Abstract

For the simulation and optimization of large scale chemical processes, the overall computing time is often dominated by the time needed to solve a large sparse system of linear equations. A parallel frontal solver can be used to significantly reduce the wallclock time required to solve these linear equation systems using parallel/vector supercomputers. This is done by exploiting both multiprocessing and vector processing, using a multifrontal-type approach in which frontal elimination is used for the partial factorization of each front. However, the algorithm is based on a bordered block-diagonal matrix form and thus its performance depends on the extent to which the matrix can be reordered to this form. Various approaches to achieving this ordering are discussed here. The performance of these different matrix reordering strategies for achieving the bordered block-diagonal form is then considered. Results, including a visualization of the different matrix orderings on one problem, are presented for several large scale process engineering problems.

1 Introduction

The future success of the chemical process industries depends on the ability to design and operate complex, highly interconnected plants that are profitable and that meet quality, safety, environmental and other standards. Towards this goal, process simulation and optimization tools are increasingly being used industrially in every step of the design process and in subsequent plant operations. However, the solution of realistic, industrial-scale process modeling problems for dynamic simulation and optimization is computationally very intense, and may require the use of high performance computing (HPC) technology to be done in a timely manner, especially if real-time performance is required. For example, Zitney *et al.* (1995) described a dynamic simulation problem at Bayer AG requiring 18 hours of CPU time on a CRAY C90 supercomputer when solved with the standard implementation of SPEEDUP (Aspen Technology, Inc.). To better use HPC technology in such process simulation problems requires the use of techniques that more effectively take advantage of parallel and/or vector processing.

Since most currently used techniques for solving such problems were developed for use on conventional serial machines, it is often necessary to rethink problem solving strategies in order to take full advantage of HPC power. For example, by using a linear equation solving algorithm that exploits vector processing and by addressing other implementation issues, Zitney *et al.* (1995) reduced the time needed to solve the Bayer problem from 18 hours to 21 minutes. In this problem, as in most other industrial-scale problems when an equation-oriented approach is used, the solution of large, sparse systems of linear equations is the single most computationally intensive step, requiring over 80% of the total simulation time in some cases (Zitney, 1992). Thus, any reduction in the linear system solution time will result in a significant reduction in the total simulation time. The matrices

that arise, however, generally do not have any of the desirable structural or numerical properties, such as numerical or structural symmetry, positive definiteness, and diagonal dominance, often associated with sparse matrices, and usually exploited in developing efficient algorithms for high performance computing. We consider here a parallel frontal solver (Mallya *et al.*, 1997) which can significantly reduce the wallclock time required to solve the linear equation systems arising in large scale process simulation problems, and concentrate on the matrix reordering issues that arise when this algorithm is used. In the next section, we outline the key features of the algorithm. In Section 3, we discuss the matrix reordering issues that arise in the application of the parallel frontal method, and outline various approaches to doing the reordering. Results comparing the performance of different reorderings are then presented and discussed in Section 4. As an aid in interpreting these results, different matrix orderings are visualized for one example problem.

2 Numerical Algorithm

The numerical algorithm seeks to exploit both multiprocessing *and* vector processing in the solution of process simulation problems by using a multilevel approach incorporating as many as three levels of task granularity, ranging from fine-grained to large-grained. Each level of granularity is now considered in more detail.

2.1 Fine-Grained Parallelism

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). 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 $Ux = y$.

To exploit fine-grained parallelism we use frontal elimination. The frontal method is an LU factorization technique that was originally developed to solve the banded matrices arising in finite element problems (Irons, 1970; Hood, 1976). The original motivation was, by limiting computational work to a relatively small *frontal matrix*, to be able to solve problems on machines with small core memories. Using codes such as MA42 (successor to the well-known MA32) from the Harwell Subroutine Library, this method is widely applied to finite element problems on vector supercomputers, because, since the frontal matrix can be treated as dense, most of the computations involved can be performed by using very efficient vectorized dense matrix kernels. Stadtherr and Vegeais (1985) extended this idea to the solution of process simulation problems on supercomputers, and later (Vegeais and Stadtherr, 1990) demonstrated its potential on some small problems.

More recently, an implementation (FAMP) of the frontal method, developed at Cray Research, Inc. and the University of Illinois specifically for use in the process simulation context, has been described by Zitney and Stadtherr (1993), and Zitney *et al.* (1995). This solver has been incorporated in supercomputer versions of popular process simulation and optimization codes such as ASPEN PLUS (Aspen Technology, Inc.), SPEEDUP (Aspen Technology, Inc.), and NOVA (Dynamic Optimization Technology Products, Inc.). Zitney (1992) and Zitney *et al.* (1994,1995) give several examples, including the Bayer problem discussed above, showing how the use of the frontal solver (as opposed to conventional solvers) has led to dramatic improvements in the performance of ASPEN PLUS and SPEEDUP. Recent experiments at Rutherford Appleton Laboratory (Duff,

1997) have shown that, on the Cray platform, FAMP is significantly faster than the Harwell frontal solver MA42. This has led to improvements in MA42, but FAMP remains faster on most problems.

2.2 Small-Grained Parallelism

In frontal elimination, the most expensive stage computationally involves outer-product updates of the frontal matrix. When executed on a single vector processor, FAMP performs efficiently because the outer-product update is readily vectorized, which as noted above is essentially a fine-grained, machine-level parallelism. An additional level of parallelism might be exploited by *microtasking* the innermost loops that perform the outer-product update. Microtasking refers to the multiprocessing of tasks with small granularity. Typically, these independent tasks can be identified quickly and exploited using compiler directives without significant code changes. Specific directives in the source code may be used to control microtasking by designating the bounds of a control structure in which each iteration of a DO loop is a process that can be executed in parallel. Our experience (Mallya, 1996), however, has shown that, at least on the Cray C90 platform, the potential for exploiting small-grained parallelism by microtasking the outer-product updates in FAMP is limited. The reason is that the parallel tasks are simply not large enough to overcome the synchronization cost and the overhead associated with invoking multiple processors on the C90. This indicates the need for exploiting a higher, coarse-grained level of parallelism to make multiprocessing worthwhile for the solution of sparse linear systems in process simulation and optimization.

2.3 Coarse-Grained Parallelism

The main deficiency with the frontal code FAMP is that there is little opportunity for mul-

titasking beyond that which can be achieved by microtasking the inner loops or by using higher level BLAS in performing the outer product update (Mallya, 1996). We overcome this problem by using a coarse-grained parallel approach in which frontal elimination is performed simultaneously in multiple independent or loosely connected blocks. This can be interpreted as applying frontal elimination to the diagonal blocks in a bordered block-diagonal matrix form as described below. It can also be interpreted as a multifrontal approach (e.g., Davis and Duff, 1997; Zitney *et al.*, 1996; Mallya and Stadtherr, 1997) with large independent pivot blocks factored by frontal elimination. Duff and Scott (1994) have applied this type of approach in solving finite element problems and referred to it as a "multiple fronts" (as opposed to multifrontal) approach.

Consider a matrix in singly-bordered block-diagonal form:

$$A = \left[\begin{array}{cccc} A_{11} & & & \\ & A_{22} & & \\ & & \ddots & \\ & & & A_{NN} \\ \hline S_1 & S_2 & \ldots & S_N \end{array}\right] \tag{1}$$

where the diagonal blocks A_{ii} are $m_i \times n_i$ and in general are rectangular with $n_i \geq m_i$. Because of the unit-stream nature of the problem, process simulation matrices may occur naturally in this form, as described in detail by Westerberg and Berna (1978). Each diagonal block A_{ii} comprises the model equations for a particular unit, and equations describing the connections between units, together with design specifications, constitute the border (the S_i). Of course, not all process simulation codes may use this type of problem formulation, or order the matrix directly into this form. Thus some matrix reordering scheme may need to be applied, as discussed further below.

The basic idea in the parallel frontal algorithm (PFAMP) is to use frontal elimination to partially

factor each of the A_{ii}, with each such task assigned to a separate processor. Since the A_{ii} are rectangular in general, it usually will not be possible to eliminate all the variables in the block, nor perhaps, for numerical reasons, all the equations in the block. The equations and variables that remain, together with the border equations, form a "reduced" or "interface" matrix that must then be factored. It should be emphasized that while frontal elimination is used here to partially factor the diagonal blocks, since the target machine is a vector processor, any factorization method can be used in this context. For instance, if the target architecture involves parallel computing on a network of scalar processors, then each processor might use Gaussian elimination with Markowitz-style pivoting (as in the Harwell code MA48, for example).

2.3.1 The PFAMP algorithm

The basic PFAMP algorithm is outlined below. For complete details and further discussion, see Mallya *et al.* (1997).

Algorithm PFAMP:

Begin parallel computation on P processors

For $i = 1 : N$, with each task i assigned to the next available processor:

1. Do symbolic analysis on the diagonal block A_{ii} and the corresponding portion of the border (S_i) to obtain memory requirements and last occurrence information (for determining when a column is fully summed) in preparation for frontal elimination.

2. Assemble the nonzero rows of S_i into the frontal matrix.

3. Perform frontal elimination on A_{ii}, beginning with the assembly of the first row of A_{ii} into the frontal matrix. The maximum number of variables that can be eliminated is m_i, but the

actual number of pivots done is $p_i \leq m_i$. The numerical pivoting scheme used is discussed below.

4. Store the computed columns of L and rows of U. Store the rows and columns remaining in the frontal matrix for assembly into the interface matrix.

End parallel computation

5. Assemble the interface matrix from the contributions of Step 4 and factor.

Note that for each block the result of Step 3 is

$$\begin{array}{c} \\ R_i \\ R_i' \end{array} \begin{array}{c} \begin{array}{cc} C_i & C_i' \end{array} \\ \left[\begin{array}{c|c} L_iU_i & U_i' \\ \hline L_i' & F_i \end{array}\right] \end{array} \tag{2}$$

where R_i and C_i are index sets comprising the p_i pivot rows and p_i pivot columns, respectively. R_i is a subset of the row index set of A_{ii}. R_i' contains row indices from S_i (the nonzero rows) as well as from any rows of A_{ii} that could not be eliminated for numerical reasons. As they are computed during Step 3, the computed columns of L and rows of U are saved in arrays local to each processor. Once the partial factorization of A_{ii} is complete, the computed block-column of L and block-row of U are written into global arrays in Step 4 before that processor is made available to start the factorization of another diagonal block. The remaining frontal matrix F_i is a contribution block that is stored in central memory for eventual assembly into the interface matrix in Step 5.

The overall situation at the end of the parallel computation section is:

$$
\begin{array}{c}
 \\
R_1 \\
R_2 \\
\vdots \\
R_N \\
R'
\end{array}
\begin{array}{c}
\begin{array}{ccccc} C_1 & C_2 & \dots & C_N & C' \end{array} \\
\left[\begin{array}{ccccc|c}
L_1U_1 & & & & & U'_1 \\
 & L_2U_2 & & & & U'_2 \\
 & & \ddots & & & \vdots \\
 & & & L_NU_N & & U'_N \\
\hline
L'_1 & L'_2 & \dots & L'_N & & F
\end{array}\right]
\end{array}
\tag{3}
$$

where $R' = \bigcup_{i=1}^{N} R'_i$ and $C' = \bigcup_{i=1}^{N} C'_i$. F is the interface matrix that can be assembled by the summation of elements from the contribution blocks F_i. Note that, since a row index in R' may appear in more than one of the R'_i and a column index in C' may appear in more than one of the C'_i, some elements of F may get contributions from more than one of the F_i.

Once factorization of all diagonal blocks is complete, the interface matrix is factored. This is carried out using the FAMP solver, with microtasking to exploit loop-level parallelism for the outer-product update of the frontal matrix. However, as noted above, this tends to provide little speedup, though there are some exceptions. Thus the factorization of the interface problem can in most cases be regarded as essentially serial. This constitutes a computational bottleneck. Thus, it is critical to keep the size of the interface problem small to achieve good speedups for the overall solution process. It should also be noted that depending on the size and sparsity of the interface matrix, some solver other than FAMP may in fact be more attractive for performing the factorization.

As the doubly-bordered block-diagonal form makes clear, once the interface matrix has been factored and its solution obtained, the remaining triangular solves needed to obtain the overall solution can be done in parallel using the same decomposition used to do the parallel frontal elimination. During this process the solution to the interface problem is made globally available to

each processor.

2.3.2 Numerical Pivoting

It is necessary to perform numerical pivoting to maintain stability during the elimination process. The frontal code FAMP uses partial pivoting to provide numerical stability. However, with the parallel frontal scheme of PFAMP, we need to ensure that the pivot row belongs to the diagonal block A_{ii}. We cannot pick a pivot row from the border S_i because border rows are shared by more than one diagonal block. Thus we use here a partial-threshold pivoting strategy. Partial pivoting is carried out to find the largest element in the pivot column while limiting the search to the rows that belong to the diagonal block A_{ii}. This element is chosen as the pivot element if it satisfies a threshold pivot tolerance criterion with respect to the largest element in the entire pivot column (including the rows that belong to the diagonal block A_{ii} *and* the border S_i). If a pivot search does not find an element that satisfies this partial-threshold criteria, then the elimination of that variable is delayed and the pivot column becomes part of the interface problem. If there are more than $n_i - m_i$ such delayed pivots then $p_i < m_i$ and a row or rows of the diagonal block will also be made part of the interface problem. This has the effect of increasing the size of the interface problem; however, our computational experiments indicate that the increase in size is very small compared to n, the overall problem size.

3 Matrix Reordering

For the solution method described above to be most effective, the size of the interface problem must be kept small. Furthermore, for load balancing reasons, it is desirable that the diagonal blocks be nearly equal in size (and preferably that the number of them be a multiple of the number of

processors to be used). For a large scale simulation or optimization problem, the natural unit-stream structure, as expressed in Eq. (1), may well provide an interface problem of reasonable size. This structure is used in two of the test problems, both occurring in problems solved using NOVA. When the unit-stream structure is used, load balancing is likely to be a problem, as the number of equations in different unit models may vary widely. This might be handled in an *ad hoc* fashion, by combining small units into larger diagonal blocks (with the advantage of reducing the size of the border) or by breaking larger units into smaller diagonal blocks (with the disadvantage of increasing the size of the border). Doing the latter also facilitates an equal distribution of the workload across the processors by reducing the granularity of the tasks. It should be noted in this context that in PFAMP task scheduling is done dynamically, with tasks assigned to processors as the processors become available. This helps reduce load imbalance problems for problems with a large number of diagonal blocks.

To address the issues of load balancing and of the size of the interface problem in a more systematic fashion, and to handle the situation in which the application code does not provide a bordered block-diagonal form directly in the first place, there is a need for matrix reordering algorithms. 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., 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 context of nested dissection applied to finite element problems or in the context of block preconditioners for iterative linear solvers. 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 simulation and optimization problems, the approach cannot be expected to always yield good results, 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. To test one reordering method in this category, we used the TPABLO code of Choi and Szyld (1996) on three of the test problems.

The TPABLO code was developed in the context of block preconditioning, and aims to find an ordering with relatively few nonzeros with magnitude below a given threshold outside the diagonal blocks. For our experiments no threshold was used, so in effect the TPABLO code was executing the earlier PABLO algorithm (O'Neil and Szyld, 1990), which does not have provision for a threshold. This algorithm is based on partitioning an undirected graph G into a number of subgraphs corresponding to diagonal blocks in the reordered matrix. A vertex is added to a subgraph based on criteria that consider the number of adjacent vertices in the subgraph (relative to the number of adjacent vertices not in the subgraph) and the density of the corresponding diagonal block. The goal is to produce a permutation that has relatively dense diagonal blocks with relatively few nonzeros outside the diagonal blocks. When used in connection with the parallel frontal method, the columns containing nonzeros outside the diagonal blocks become part of the interface problem. Rows and other columns that cannot be eliminated for numerical reasons are assigned to the interface problem as a result of the pivoting strategy used in the frontal elimination of the diagonal blocks.

To deal with structurally asymmetric problems, one technique that can be used is the min-net-cut (MNC) approach of Coon and Stadtherr (1995). This technique is designed specifically to address the issues of load balancing and interface problem size. It is based on recursive bisection of a *directed* bipartite graph model of the asymmetric matrix. Since a directed bipartite graph model is

used, the algorithm can consider unsymmetric permutations of rows and columns. The kernel of the algorithm is the bisection of a directed bipartite graph $\vec{G}$ into two subgraphs with a small connecting set. This set is determined by choosing vertex pairs to move or swap across the partition boundary, in order to reduce the size of the connecting set. This is done based on heuristics, with the aim to minimize (approximately) the size of the connecting set. This partitioning proceeds recursively to the resulting subgraphs until a stopping criterion is reached. The matrix form produced is a block-tridiagonal structure in which the off-diagonal blocks have relatively few nonzero columns; this is equivalent to a special case of the bordered block-diagonal form. In applying this in the context of the parallel frontal algorithm, the columns with nonzeros in the off-diagonal blocks are treated as belonging to the interface problem. Rows and other columns that cannot be eliminated for numerical reasons are assigned to the interface problem as a result of the pivoting strategy used in the frontal elimination of the diagonal blocks. This reordering was used on all the test problems.

Another reordering technique that produces a potentially attractive structure is the tear_drop (tear, drag, reorder, partition) algorithm given by Abbott *et al.* (1997). This makes use of the block structure of the underlying process simulation problem (Stadtherr and Wood, 1984), and also employs graph bisection concepts, applied to a directed acyclic graph representation of the matrix. In this case a recursive bordered block-diagonal form results. Rows and columns in the borders are immediately assigned to the interface problem in the parallel frontal method, along with any rows and columns not eliminated for numerical reasons during factorization of the diagonal blocks. This reordering is used on two of the test problems.

In the computational results presented below, we use seven test problems, each a matrix arising in a large scale process engineering problem. For each matrix, two different orderings are considered. The results are used to demonstrate the potential of the parallel frontal solver, and to consider the

effects of reordering. This is not intended to be a systematic comparison of reordering algorithms.

4 Results and Discussion

In this section, we present results for the performance of the PFAMP solver on seven process engineering problems. More information about each problem is given below. We compare the performance of PFAMP on multiple processors with its performance on one processor and with the performance of the frontal solver FAMP on one processor. Of particular interest is the effect of matrix reordering. The numerical experiments were performed on a CRAY C90 parallel/vector supercomputer at Cray Research, Inc., in Eagan, Minnesota. The timing results presented represent the total time to obtain a solution vector from one right-hand-side vector, including analysis, factorization, and triangular solves. The time required for reordering is not included. A threshold tolerance of $t = 0.1$ was used in PFAMP to maintain numerical stability, which was monitored using the 2-norm of the residual $b - Ax$. FAMP uses partial pivoting.

In Table 1, each matrix is identified by name and order (n). In addition, statistics are given for the number of nonzeros (NZ), and for a measure of structural asymmetry (as). The asymmetry, as, is the number off-diagonal nonzeros a_{ij} ($j \neq i$) for which $a_{ji} = 0$ divided by the total number of off-diagonal nonzeros ($as = 0$ is a symmetric pattern, $as = 1$ is completely asymmetric). Also given, for each ordering used, is information about the resulting bordered block-diagonal form, namely the number of diagonal blocks (N), the order of the interface matrix (NI), and the number of equations in the largest and smallest diagonal blocks, m_i^{max} and m_i^{min}, respectively.

The first two problems (*Ethylene_1* and *Ethylene_2*) involve the application of NOVA to an ethylene plant. Each problem involves a flowsheet that consists of 43 units, including five distillation columns. The problems differ in the number of stages in the distillation columns. The linear systems

arising in NOVA are naturally in bordered block-diagonal form, allowing the direct use of PFAMP for the solution of these systems. To see the effect of a different ordering, the MNC reordering was also used.

We note first, that the single processor performance of PFAMP is better than that of FAMP. This is due to the difference in the size of the largest frontal matrix associated with the frontal elimination for each method. For solution with FAMP, the variables which have occurrences in the border equations remain in the frontal matrix until the end. The size of the largest frontal matrix increases for this reason, as does the number of wasted operations on zeros, thereby reducing the overall performance. This problem does not arise for solution with PFAMP because when the factorization of a diagonal block is complete, the remaining variables and equations in the front are immediately written out as part of the interface problem and a new front is begun for the next diagonal block. Thus, for these problems and most other problems tested, PFAMP is a more efficient serial solver than FAMP. This reflects the advantages of the multifrontal-type approach used by PFAMP, namely smaller and less sparse frontal matrices.

In the natural ordering for each problem, there are 43 diagonal blocks, of which five are large, corresponding to the distillation units, with one of these blocks much larger ($m_i = 3337$ on *Ethylene_1*) than the others ($1185 \leq m_i \leq 1804$ on *Ethylene_1*). In the computation, with five processors being used, one processor ends up working on the largest block, while the remaining four processors finish the other large blocks and the several much smaller ones. The load is unbalanced with the factorization of the largest block being the bottleneck. This, together with the solution of the interface problem, results in a speedup (relative to PFAMP on one processor) of two or less on five processors. Use of the MNC reordering provides a somewhat better load balance and a smaller interface problem. This provides for improved processor utilization (e.g., speedup of 2.2 on four

processors vs. speedup of 1.8 on 5 processors on the *Ethylene_2* problem), though this is still not particularly efficient processor utilization. Given the irregular and highly asymmetric nature of these problems this is not surprising, however.

The next three problems have been reordered into a bordered block-diagonal form using both MNC and TPABLO. Two of these problems (*Hydr1c* and *Icomp*) occur in dynamic simulation problems solved using SPEEDUP (Aspen Technology, Inc.). The *Hydr1c* problem involves a 7-component hydrocarbon process with a de-propanizer and a de-butanizer. The *Icomp* problem comes from a plantwide dynamic simulation of a plant that includes several interlinked distillation columns. The *lhr_71* problem is derived from the prototype simulator SEQUEL (Zitney and Stadtherr, 1988), and is based on a light hydrocarbon recovery plant. Neither of the application codes produces directly a matrix in bordered block-diagonal form. For example, the occurrence matrix for the *Hydr1c* problem is shown in Figure 1. Thus a reordering such as provided by MNC or TPABLO is required.

When the TPABLO reordering is used, the size of the interface problem is extremely large, over half the size of the original problem. Since the interface problem is a bottleneck in PFAMP, its performance would be clearly be very inefficient when this ordering is used, and actual numerical runs were thus not attempted. It should be noted that TPABLO has a user adjustable parameter for the maximum block size allowed. For each of the three matrices, the largest block found matched the maximum allowable block size. When this parameter was adjusted, different block partitions were found, but in general the size of the interface problem remained extremely large. The difficulty can be seen in the occurrence matrix shown in Figure 2 for the *Hydr1c* matrix after ordering with TPABLO. Clearly diagonal blocks are being formed, but with a very large number of nonzeros outside these blocks. The poor performance of this type of reordering method is not surprising

since it is based on *symmetric* permutations of very *highly asymmetric* systems. This is apparently not an appropriate application for TPABLO, which performs quite well in other contexts.

On the other hand, the MNC ordering, which does allow for asymmetric permutations, performs relatively well on these problems, as seen in the MNC reordering of *Hydr1c* in Figure 3. Here there are four diagonal blocks of fairly similar size and relatively few nonzeros outside these diagonal blocks.

On two of the three problems, the PFAMP algorithm again outperforms FAMP even on a single processor, for the reasons discussed above. This enhancement of performance can be quite significant, around a factor of two in the case of *lhr_71*. MNC achieves its best reordering on the *Icomp* problem, for which it finds four diagonal blocks of nearly the same size ($17168 \leq m_i \leq 17393$) and the size of the interface problem is relatively small in comparison to n. The speedup observed for PFAMP on this problem was about 2.5 on four processors. While this represents a substantial savings in wallclock time, it still does not represent efficient processor utilization. In this context, it should be remembered that even a relatively small serial component in a computation can greatly reduce the efficiency of processor utilization [see Vegeais and Stadtherr (1992) for further discussion of this point].

The final two problems arise from simulation problems solved using ASCEND (Piela *et al.*, 1991), and ordered using the tear_drop approach (Abbott, 1996) and also using MNC. Problems *4cols.smms* and *10cols.smms* involve nine components with four and ten interlinked distillation columns, respectively. With the tear_drop reordering, the resulting moderate task granularity helps spread the load over the four processors used, but the size of the interface problem tends to be relatively large, 17-19% of n, as opposed to 1-3% when MNC is used. However, for MNC the load balancing characteristics are less desirable, as in each case two of the four blocks are

significantly smaller than the other the two. Thus, though both approaches provide significant reductions in wallclock time, neither achieved particularly good parallel efficiency. MNC does have user adjustable parameters that could possibly be modified to provide a better balance between the number of blocks and the size of the interface problem. It should be noted that reasonably good performance was obtained with the tear_drop reordering despite the relatively large size of the interface problem because, for these systems, the use of small-grained parallelism within FAMP for solving the interface problem provided a significant speedup (about 1.7 on 10cols.smms). Overall on *10cols.smms* the use of PFAMP resulted in the reduction of the wallclock time by an order of magnitude; however only a factor of about two of this was due to multiprocessing.

5 Concluding Remarks

The results presented above demonstrate that PFAMP can be an effective solver for use in process simulation and optimization on parallel/vector supercomputers with a relatively small number of processors. In addition to making better use of multiprocessing than the standard solver FAMP, on most problems the single processor performance of PFAMP was better than that of FAMP. The combination of these two effects led to five- to ten-fold performance improvements on some large problems. Two keys to obtaining better parallel performance are improving the load balancing in factoring the diagonal blocks and better parallelizing the solution of the interface problem.

Clearly the performance of PFAMP with regard to multiprocessing depends strongly on the quality of the reordering into bordered block-diagonal form. In most cases considered above it is likely that the reorderings used were far from optimal, and no systematic attempt was made to find better reorderings. The graph partitioning problems underlying the reordering algorithms are NP-complete. Thus, one can easily spend a substantial amount of computation time attempting to

find improved orderings. Indeed, for the MNC reorderings of the larger problems, several hundred seconds of CPU time was required for the ordering, which is much more than the solution time. Thus, the cost of a good ordering must be weighed against the number of times a given simulation or optimization problem is going to be solved. Typically, if the effort is made to develop a large scale simulation or optimization model, then it is likely to be used a very large number of times, especially if it is used in an operations environment. In this case, the investment made to find a good reordering for PFAMP to exploit might have substantial long term paybacks.

Acknowledgments – This work has been supported by the National Science Foundation under Grants DMI-9322682 and DMI-9696110. We also acknowledge the support of the National Center for Supercomputing Applications at the University of Illinois, Cray Research, Inc. and Aspen Technology, Inc. We thank Dr. Kirk Abbott for providing the ASCEND matrices and the tear_drop reorderings.

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. (Available at http://www.math.temple.edu/~szyld) (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).

Davis, T. A. and I. S. Duff, An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM J. Matrix Anal. Appl.*, **18**, 140–158 (1997).

Duff, I. S., Sparse numerical linear algebra: Direct methods and pre-conditioning. Presented at AspenWorld 97, Boston, MA, October 12-15, 1997.

Duff, I. S. and J. A. Scott, The use of multiple fronts in Gaussian elimination. Technical Report RAL 94-040, Rutherford Appleton Laboratory, Oxon, UK (1994).

Hood, P., Frontal solution program for unsymmetric matrices. *Int. J. Numer. Meth. Engng*, **10**, 379 (1976).

Irons, B. M., A frontal solution program for finite element analysis. *Int. J. Numer. Meth. Engng*, **2**, 5 (1970).

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 factoriation. In Rodrigue, G., editor, *Parallel Processing for Scientific Computing*, pages 27–31. SIAM, Philadelphia, PA (1989).

Mallya, J. U., *Vector and Parallel Algorithms for Chemical Process Simulation on Supercomputers.* PhD thesis, Dept. of Chemical Engineering, University of Illinois, Urbana, IL (1996).

Mallya, J. U. and M. A. Stadtherr, A multifrontal approach for simulating equilibrium-stage processes on supercomputers. *Ind. Eng. Chem. Res.*, **36**, 144–151 (1997).

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 (1997).

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

Piela, P. C., T. G. Epperly, K. M. Westerberg, and A. W. Westerberg, ASCEND: An object-oriented computer environment for modeling and analysis: The modeling language. *Comput. Chem. Engng*, **15**, 53–72 (1991).

Stadtherr, M. A. and J. A. Vegeais, Process flowsheeting on supercomputers. *IChemE Symp. Ser.*, **92**, 67–77 (1985).

Stadtherr, M. A. and E. S. Wood, Sparse matrix methods for equation-based chemical process flowsheeting: I. Reordering phase. *Comput. Chem. Engng*, **8**, 9–18 (1984).

Vegeais, J. A. and M. A. Stadtherr, Vector processing strategies for chemical process flowsheeting. *AIChE J.*, **36**, 1687–1696 (1990).

Vegeais, J. A. and M. A. Stadtherr, Parallel processing strategies for chemical process flowsheeting. *AIChE J.*, **38**, 1399–1407 (1992).

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 Rippin, D. W. T., J. C. Hale, and J. F. Davis, editors, *Proc. 2nd International Conference on Foundations of Computer-Aided Process Operations*, pages 463–468. CACHE Corp., Austin, TX (1994).

Zitney, S. E., J. U. Mallya, T. A. Davis, and M. A. Stadtherr, Multifrontal vs frontal techniques for chemical process simulation on supercomputers. *Comput. Chem. Engng*, **20**, 641–646 (1996).

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: Description of test matrices and summary of results. For each matrix, the order n, the number of nonzeros NZ and the degree of asymmetry as are given (see text for complete definition of as). For each reordering, the number of diagonal blocks N, the order of the interface problem NI, and the orders of the largest and smallest diagonal blocks, m_i^{max} and m_i^{min}, respectively, are given. Solution times for the FAMP and PFAMP solvers are on a CRAY C90.

Name (Ordering)	n	NZ	as	N	m_i^{max}	m_i^{min}	NI	FAMP 1 proc. sec.	PFAMP 1 proc. sec.	PFAMP NP proc. sec. (NP)
Ethylene_1	10673	80904	0.99							
(Natural)				43	3337	1	708	0.697	0.550	0.267 (5)
(MNC)				4	3560	1637	181		0.682	0.360 (4)
Ethylene_2	10353	78004	0.99							
(Natural)				43	3017	1	698	0.667	0.510	0.290 (5)
(MNC)				4	2930	2388	264		0.570	0.256 (4)
Hydr1c	5308	23752	0.99							
(TPABLO)				90	500	2	3288			
(MNC)				4	1449	1282	180	0.258	0.243	0.139 (4)
Icomp	69174	301465	0.99							
(TPABLO)				199	8000	2	37335			
(MNC)				4	17393	17168	1054	3.78	4.33	1.72 (4)
lhr_71	70304	1528092	0.99							
(TPABLO)				733	8000	2	35510			
(MNC)				10	9215	4063	1495	14.8	7.67	3.04 (4)
4cols.smms	11770	43668	0.99							
(Tear_drop)				24	1183	33	2210	1.14	1.13	0.680 (4)
(MNC)				4	4456	883	365		0.874	0.443 (4)
10cols.smms	29496	109588	0.99							
(Tear_drop)				66	1216	2	5143	11.3	3.69	1.81 (4)
(MNC)				4	10334	3810	293		1.53	0.905 (4)

Figure Captions

Figure 1. Occurrence matrix for the *Hydr1c* problem as generated by the SPEEDUP simulator.

Figure 2. Occurrence matrix for the *Hydr1c* problem after ordering by TPABLO.

Figure 3. Occurrence matrix for the *Hydr1c* problem after ordering by MNC.

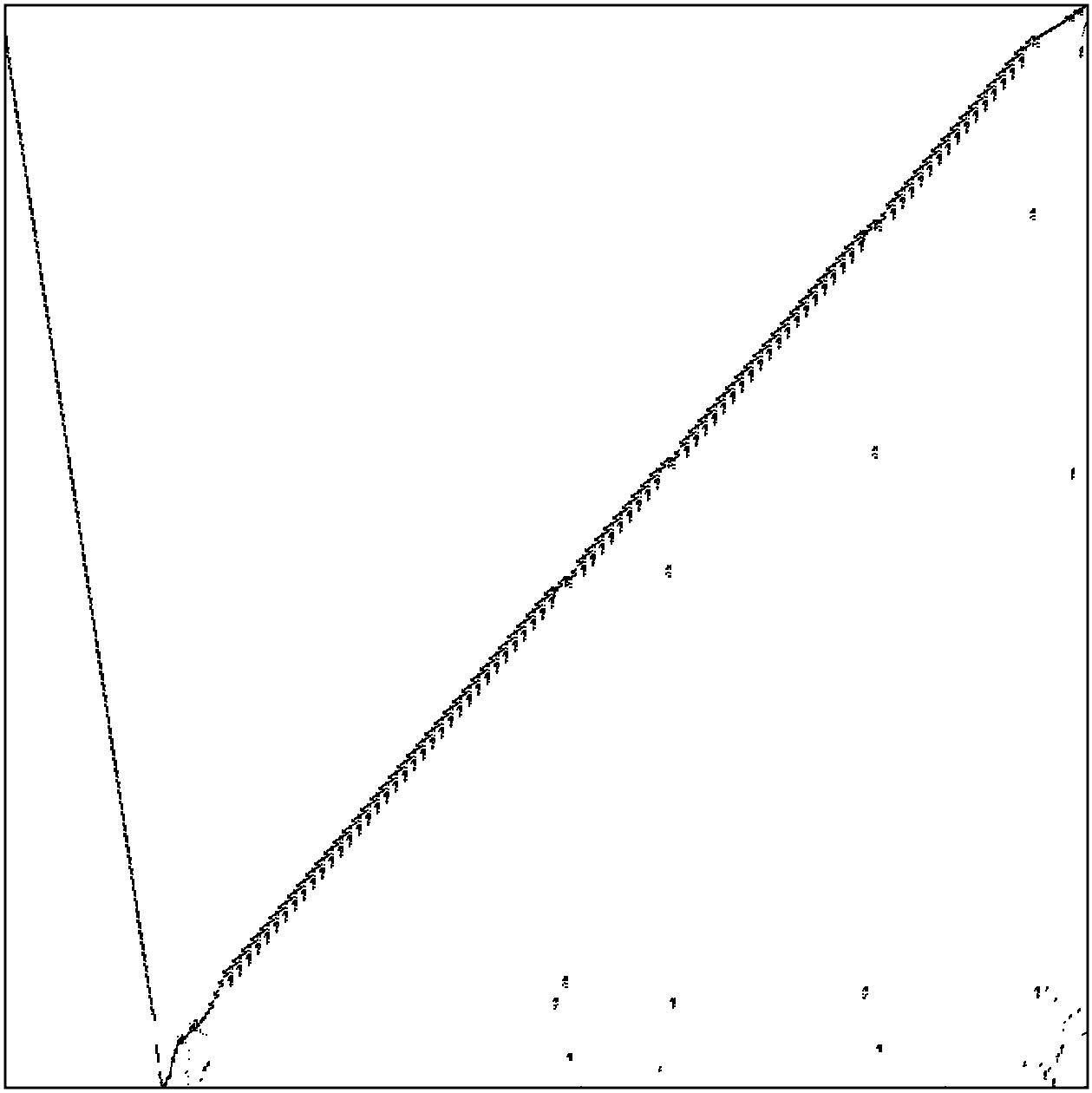

Figure 1: Occurrence matrix for the *Hydr1c* problem as generated by the SPEEDUP simulator.

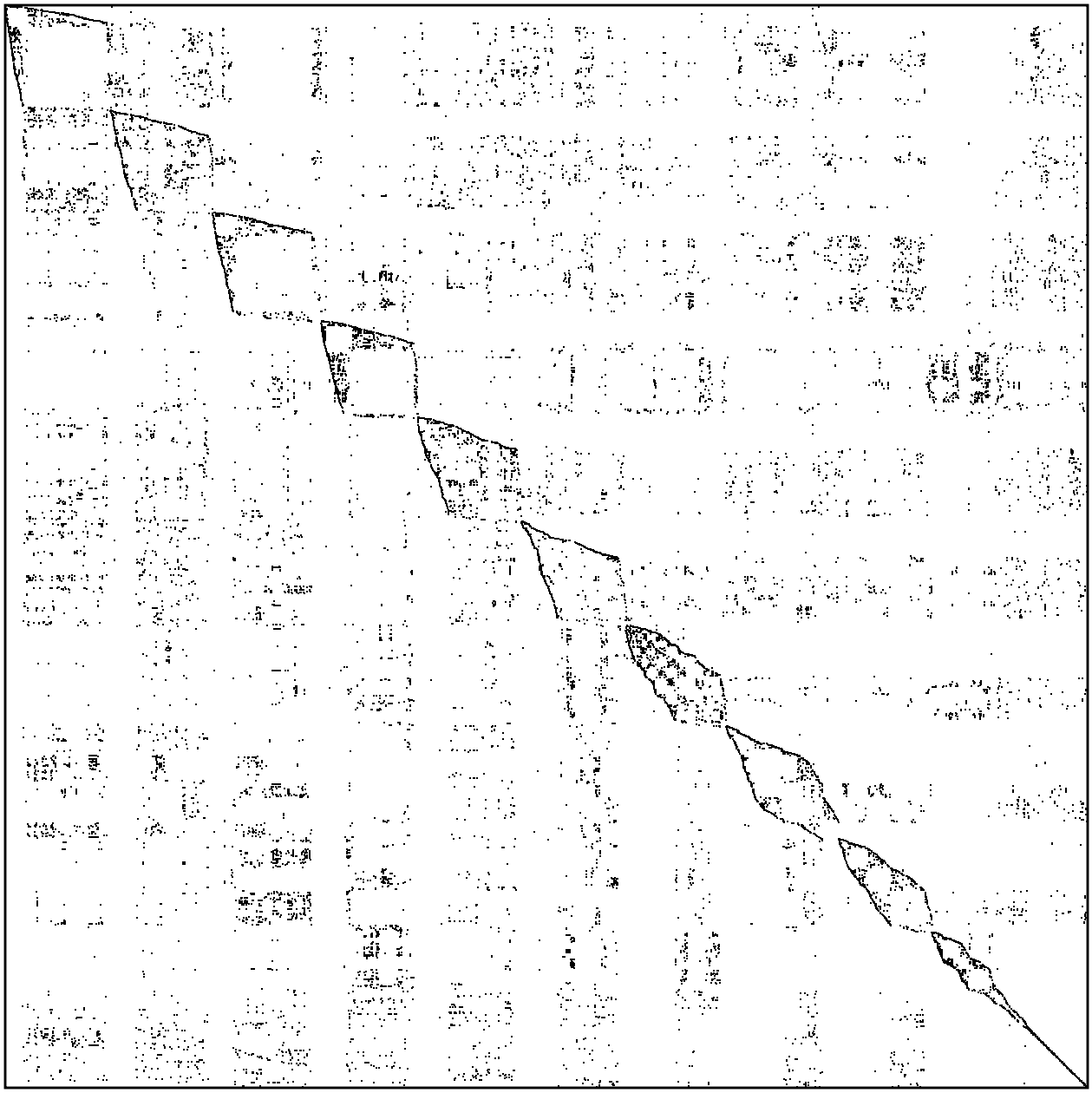

Figure 2: Occurrence matrix for the *Hydr1c* problem after ordering by TPABLO.

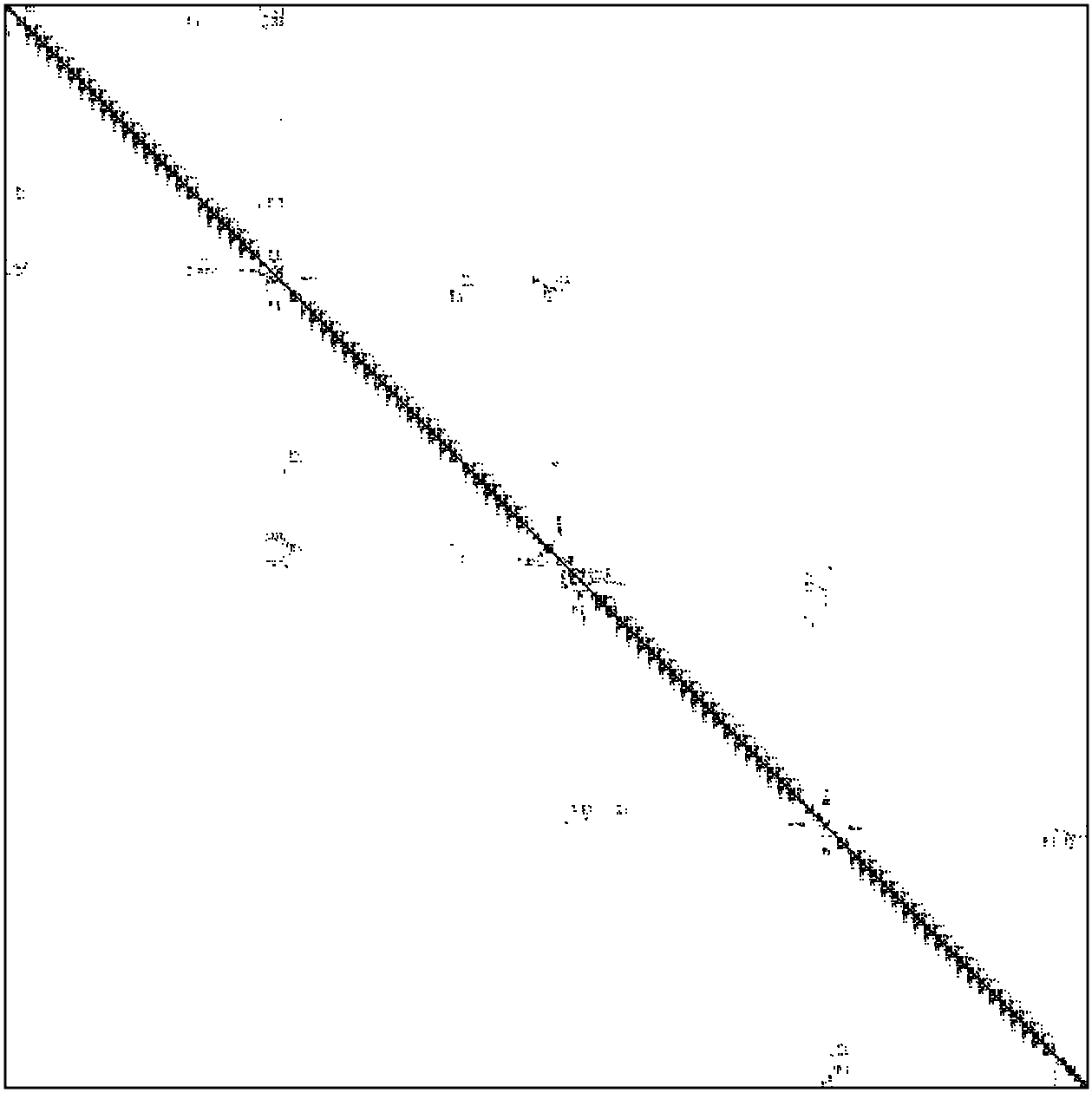

Figure 3: Occurrence matrix for the *Hydr1c* problem after ordering by MNC.