Computational Strategies for Chemical Process Engineering Using Parallel/Vector Supercomputers

By MARK A. STADTHERR[1] AND JAYARAMA U. MALLYA[2]

[1] Department of Chemical Engineering, University of Notre Dame, Notre Dame, IN 46556 USA

[2] Cray Research, Inc., 655E Lone Oak Dr., Eagan, MN 55121 USA

In this article we consider strategies for exploiting supercomputer technology in solving the sparse matrix problems arising in process simulation and optimization. A new multifrontal solver for use in simulating equilibrium-stage processes and a new parallel frontal solver for large-grained parallel solution of process simulation and optimization problems are described. Results for several problems, including large-scale industrial problems, are given.

---

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. To perform realistic process simulation for very large scale industrial processes, however, requires adequate computational resources. Today, parallel/vector supercomputers provide the computational power to realistically model, simulate, design, and optimize complex chemical manufacturing processes, steady- and unsteady-state. To better use this leading edge technology in process simulation requires the use of techniques that efficiently exploit vector and parallel 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 the supercomputing power.

A key step in solving complex process simulation problems is the solution of a large, sparse system of linear equations. In fact, this step may account for as much as 90% of the computation time on industrial-scale problems. Thus, any reduction in the linear system solution time will result in a significant reduction of the total simulation time. The matrices that arise, however, 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 parallel/vector computing. We describe here vector and parallel algorithms for the solution of linear equation systems arising in process simulation and optimization on supercomputers.

2. Frontal Method

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 $Ux = y$.

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. The well-known code MA32 from the Harwell Subroutine library (Duff, 1984) is a frontal code for solving finite element problems on vector computers. This method is widely used for 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. An implementation of the frontal method developed specifically for use in the process simulation context has been described by Zitney (1992), Zitney and Stadtherr (1993), and Zitney *et al.* (1995), and is now incorporated in supercomputer versions of popular process simulation and optimization codes.

The frontal elimination scheme can be outlined briefly as follows:

(1) Assemble a row into the frontal matrix.

(2) Determine if any columns are fully summed in the frontal matrix. A column is fully summed if it has all of its nonzero elements in the frontal matrix.

(3) If there are fully summed columns, then perform partial pivoting in those columns, eliminating the pivot rows and columns and doing an outer-product update on the remaining part of the frontal matrix.

This procedure begins with the assembly of row 1 into the initially empty frontal matrix, and proceeds sequentially row by row until all are eliminated, thus completing the LU factorization. To see this in mathematical terms, consider the submatrix $A^{(k)}$ remaining to be factored after the $(k-1)$-th pivot:

$$A^{(k)} = \begin{bmatrix} F^{(k)} & 0 \\ A_{ps}^{(k)} & A_{ns}^{(k)} \end{bmatrix}. \tag{2.1}$$

Here $F^{(k)}$ is the frontal matrix, $A_{ps}^{(k)}$ contains columns that are partially summed (some but not all nonzeros in the frontal matrix), and $A_{ns}^{(k)}$ contains columns that are not summed (no nonzeros in the frontal matrix). Assembly of rows into the frontal matrix then proceeds until $g_k \geq 1$ columns become fully summed:

$$A^{(k)} = \begin{bmatrix} \bar{F}_{11}^{(k)} & \bar{F}_{12}^{(k)} & 0 \\ \bar{F}_{21}^{(k)} & \bar{F}_{22}^{(k)} & 0 \\ 0 & \bar{A}_{ps}^{(k)} & \bar{A}_{ns}^{(k)} \end{bmatrix}. \tag{2.2}$$

$\bar{F}^{(k)}$ is now the frontal matrix and $\bar{F}_{11}^{(k)}$ and $\bar{F}_{21}^{(k)}$ comprise the columns that have become fully summed, which are now eliminated using rows chosen during partial pivoting and which are shown as belonging to $\bar{F}_{11}^{(k)}$ here. This amounts to the factorization $\bar{F}_{11}^{(k)} = L_{11}^{(k)} U_{11}^{(k)}$ of the order-g_k block $\bar{F}_{11}^{(k)}$, resulting in:

$$A^{(k)} = \begin{bmatrix} L_{11}^{(k)} U_{11}^{(k)} & U_{12}^{(k)} & 0 \\ L_{21}^{(k)} & F^{(k+g_k)} & 0 \\ 0 & A_{ps}^{(k+g_k)} & A_{ns}^{(k+g_k)} \end{bmatrix} \tag{2.3}$$

where the new frontal matrix $F^{(k+g_k)}$ is the Schur complement $F^{(k+g_k)} = \bar{F}_{22}^{(k)} - L_{21}^{(k)} U_{12}^{(k)}$, which is computed using an efficient full-matrix outer-product update kernel, $A_{ps}^{(k+g_k)} = \bar{A}_{ps}^{(k)}$ and $A_{ns}^{(k+g_k)} = \bar{A}_{ns}^{(k)}$. Note that operations are done within the frontal matrix only. At this point $L_{11}^{(k)}$ and $L_{21}^{(k)}$ contain columns k through $k + g_k - 1$ of L and $U_{11}^{(k)}$ and $U_{12}^{(k)}$ contain rows k through $k + g_k - 1$ of U. The computed columns of L and rows of U are saved and the procedure continues with the assembly of the next row into the new frontal matrix $F^{(k+g_k)}$.

2.1. *Application of frontal method in process simulation*

The outer-product updates in the innermost loops of the frontal code are done using readily vectorizable BLAS2 or BLAS3 dense matrix kernels (BLAS indicates Basic Linear Algebra Subroutines: BLAS2 covers matrix-vector operations and BLAS3 matrix-matrix). However, for process simulation problems the frontal matrices are often relatively large and sparse. Thus, while a high computational rate can be achieved when operating on frontal matrices, a large number of unnecessary operations on zeros may be performed, potentially lowering overall performance. In most cases, however, the time spent on wasted operations is more than made up for by the faster computational rate achievable.

FAMP has now been incorporated in CRAY versions of popular commercial simulation codes, such as ASPEN PLUS, SPEEDUP, RATEFRAC, and BATCHFRAC (Aspen Technology, Inc.). Zitney (1992) and Zitney *et al.* (1994) give several examples 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. Zitney *et al.* (1995) describe 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. With a CRAY optimized version of SPEEDUP, which contains the frontal code FAMP and an improved residual evaluator CRAYRES, this simulation now takes only 21 CPU minutes, with most of the improvement due to the frontal solver. As a result, Bayer engineers can run this simulation many times per day instead of only once per day. This improves engineering productivity and let users consider more alternatives in a shorter time while not sacrificing model size and/or complexity.

3. Multifrontal Method

The multifrontal method is a generalization of the frontal method, and was originally developed for symmetric matrices. Like the frontal method, it also exploits low-level parallelism and vectorization through the use of dense matrix kernels on frontal matrices. However, the frontal matrices are generally smaller and denser than in the frontal method. The classical multifrontal approach (Duff and Reid, 1984) has met with only limited success when the pattern of nonzeros is highly unsymmetric. However, recently a new unsymmetric-pattern multifrontal algorithm has been described by Davis and Duff (1993,1996) and implemented in the code UMFPACK (Davis and Duff, 1995). In this method, a frontal matrix, consisting of pivot row(s) and column(s), their entries from the original matrix A, and contributions to them from previous frontal matrices, is assembled at each stage of the factorization process. The frontal matrix E_k for steps k through $k + g_k - 1$ of the LU factorization, where g_k is the number of pivots performed in E_k can be represented as

$$\begin{array}{cc} & \begin{array}{cc} C_k & C'_k \end{array} \\ \begin{array}{c} R_k \\ R'_k \end{array} & \left[\begin{array}{cc} F_k & B_k \\ T_k & D_k \end{array} \right]. \end{array} \tag{3.4}$$

E_k is labeled with the ordered sets R_k and C_k, representing the pivot rows and columns, respectively, and with the sets R'_k and C'_k, representing the non-pivotal rows and columns. The blocks F_k, B_k, and T_k are all fully assembled with contributions from both the original matrix and from previous frontal matrices; however contributions to D_k may be only partially assembled or not assembled at all. The pivot block F_k is now factored ($F_k = L_k U_k$) thus determining a block-column L'_k of L and a block-row U'_k of U. An outer-product update $D'_k = D_k - L'_k U'_k$ is then performed to complete the elimination operations on this frontal matrix, thus resulting in

$$\begin{array}{cc} & \begin{array}{cc} C_k & C'_k \end{array} \\ \begin{array}{c} R_k \\ R'_k \end{array} & \left[\begin{array}{cc} L_k U_k & U'_k \\ L'_k & D'_k \end{array} \right] \end{array} . \tag{3.5}$$

L_k and L'_k can be written into an array for L factors. Similarly, U_k and U'_k can be written into an array for U factors. The contribution block D'_k is saved since some of its elements may need to be assembled into future frontal matrices. This interwoven assembly-elimination process confines the arithmetic operations to the frontal matrices, and so permits the use of efficient dense matrix vector operations during the factorization of F_k and the update of D_k.

Zitney *et al.* (1996) have compared the performance of the general-purpose unsymmetric-pattern multifrontal approach (UMFPACK) with that of the frontal code (FAMP), as well as with that of the conventional code MA28. Results on a set of six chemical process simulation problems and five other engineering problems show that the frontal and multifrontal methods are significantly faster than MA28, reflecting in part the use of vectorized dense matrix kernels. Comparing the frontal and multifrontal methods, the frontal method is most effective on problems with good initial matrix orderings, while the multifrontal method is most attractive for problems without a good initial ordering. For process simulation problems, good initial orderings are not uncommon if the equations describing each unit (or equilibrium stage) in a process are kept together, and if adjacent units and streams are numbered consecutively, thus resulting in a nearly block-banded matrix corresponding to the unit-stream nature of the problem. In the general-purpose multifrontal approach, a pivot element is chosen using a Markowitz-style strategy to preserve sparsity. Additional pivots may then be chosen to form a pivot block if they do not cause growth of the assembled frontal matrix beyond a preset limit. In the context of process simulation, the disadvantage of this approach is that it does not take advantage of the good initial structure of the matrix, and may in fact destroy it. The multifrontal algorithm presented below is designed to avoid this difficulty.

3.1. *The MFA1P algorithm*

MFA1P (**M**ulti**F**rontal **A**lgorithm, **1** Pivot) is designed to take advantage of good initial structure in process simulation matrices, especially those that primarily involve equilibrium-stage operations. The algorithm uses a modified threshold pivot search strategy that attempts to maintain the structure during the factorization process. The basic MFA1P algorithm is outlined below:

Algorithm MFA1P:

For $k = 1 : n$

(1) Start the k-th frontal matrix by assembling all contributions to the k-th column (including entries from the original matrix and contributions from previous frontal matrices). This is the pivot column. Store as a column of L.

(2) Choose as a pivot the element in the pivot column closest to (preferably on) the diagonal that satisfies a threshold pivot tolerance. This determines the pivot row.

Name	n	NZ	as	FAMP	UMFPACK	MA28	MFA1P
v3	1078	16937	0.91	0.114	0.243	2.159	**6.01x10^{-2}**
v10	1148	15729	0.94	0.109	0.227	1.862	**6.10x10^{-2}**
v13	834	9713	0.95	6.35x10^{-2}	0.140	0.983	**4.20x10^{-2}**
mpex2	848	11413	0.96	6.60x10^{-2}	0.176	0.299	**4.27x10^{-2}**
mpex3	2473	46503	0.94	0.359	0.567	10.598	**0.173**
mpex4	2478	44075	0.95	0.317	0.559	9.19	**0.172**
mpmult1	2023	31894	0.95	0.234	0.472	6.131	**0.13**
rdist1	4134	94408	0.94	0.730	1.854	30.21	**0.32**
rdist2	3198	56934	0.95	0.392	0.696	16.11	**0.22**
rdist3	2398	61896	0.85	0.478	1.172	32.87	**0.20**
sumb	523	4998	0.95	3.22x10^{-2}	8.30x10^{-2}	0.314	**2.18x10^{-2}**
traycalc	1145	20296	0.88	0.14	0.244	2.649	**6.81x10^{-2}**
uosb	523	4998	0.95	3.22x10^{-2}	8.29x10^{-2}	0.315	**2.19x10^{-2}**
userupp	1269	22508	0.89	0.154	0.289	3.310	**7.39x10^{-2}**

TABLE 1. Run times (s) for ASPEN PLUS test problems on A + F + S execution path. See text for definition of column headings.

(3) Assemble all contributions to the pivot row and normalize it. Store as a row of U.

(4) Perform an outer product update of D_k using the pivot column and normalized pivot row to compute the contribution block D'_k for this frontal matrix. Store this contribution block for later use.

The key feature of the algorithm is the simple pivot selection scheme used in Step 2. The pivot row j is chosen to minimize $|j-k|$ subject to the threshold tolerance criterion $|a_{jk}| \geq t \times max_s|a_{sk}|$, where t is a preset fraction in the range $0 < t \leq 1.0$. This is in contrast to the frontal method, in which partial pivoting is used and the largest element in the column is chosen as the pivot ($t = 1$). It is also in contrast to the general-purpose unsymmetric-pattern frontal method, in which a global Markowitz-style pivot search with threshold is used. MFA1P tries to maintain the initial matrix structure by choosing as the pivot the element closest to and preferably on the diagonal, while maintaining numerical stability by using the threshold tolerance. In our experiments a threshold tolerance of $t = 0.1$ was adequate to maintain numerical stability.

3.2. *Results and discussion*

Table 1 compares the performance of MFA1P with that of the frontal solver FAMP, the general-purpose unsymmetric-pattern multifrontal solver UMFPACK, and, to provide a familiar benchmark, the conventional solver MA28. Version 2.0 of UMFPACK was used; this version (Davis and Duff, 1995) incorporates features of the frontal method into the multifrontal solver in order to improve overall efficiency. The default parameter settings were used for each code. Numerical experiments were carried out on a CRAY C90 parallel/vector supercomputer at Cray Research, Inc. in Eagan, Minnesota (USA). The problem set includes 14 steady-state simulation problems solved using ASPEN PLUS (Aspen Technology, Inc.). These problems use the RADFRAC module of ASPEN PLUS. This module does rigorous calculations for all types of fractionation, including absorption, reboiled absorption, stripped, reboiled stripping, and extractive, azeotropic, and three phase distillation, in addition to ordinary distillation.

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).

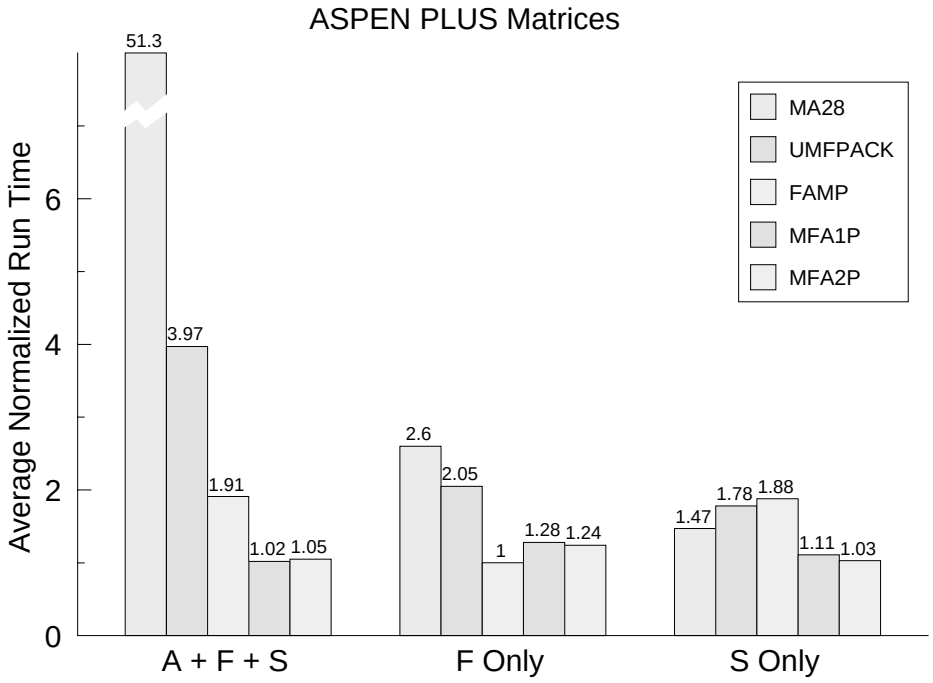

FIGURE 1. Average normalized run times for ASPEN PLUS matrices on the analyze + factorize + solve (A + F + S), factor (F) only, and solve (S) only execution paths.

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). Run times (in CPU seconds) represent the total time to perform analysis to determine a pivot sequence, to compute the L and U factors of A, and to perform the forward and backward substitution to solve $Ax = b$. This execution path (**A**nalyze + **F**actor + **S**olve) is typically used at each iteration in a steady-state simulation. The fastest run time for each problem is shown in bold. Data for the factor only and solve only execution paths are given by Mallya and Stadtherr (1996), along with results for a variety of other problems. Mallya and Stadtherr (1996) also describe a version (MFA2P) of this multifrontal approach in which two pivots are performed in each frontal matrix.

Figure 1 shows a summary of the relative performance of the methods for each execution path. This summary is based on *average normalized run time* (ANRT). For each problem and method the run time is normalized by dividing by the run time of the best method on that problem. This is then averaged over the entire problem set to determine the ANRT for each method. Thus, an ANRT of one for a method would indicate that the method was the best method on all problems in the problem set.

For the critical A + F + S execution path, the MFA1P or MFA2P solver is the best on all problems and on the average is nearly twice as fast as the frontal solver FAMP. Neither UMFPACK nor MA28 are able to take good advantage of the structure of these problems and, in fact, spend considerable effort finding a different pivot sequence. Some savings can be achieved in these codes by turning off the default permutation to block upper triangular form, which in general is not useful on these problems. It should also be noted that MA48, the successor to MA28 in the Harwell Subroutine Library, should

perform better than MA28 on these problems, but still not better than the other codes on the A+F+S execution path. For instance, on the *rdist1* problem, Davis and Duff (1995) found that UMFPACK was about six times faster than MA48.

4. Parallel Frontal Method

The main deficiency with the frontal code FAMP and multifrontal codes MFA1P and MFA2P is that there is little opportunity for parallelism beyond that which can be achieved by microtasking the inner loops or by using higher level BLAS in performing the outer product update, which unfortunately usually provides relatively little speedup (Mallya, 1996; Camarda and Stadtherr, 1994). 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 coarse-grained multifrontal approach (e.g., Davis and Duff, 1996; Zitney *et al.*, 1996) 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:

$$A = \begin{bmatrix} A_{11} & & & & \\ & A_{22} & & & \\ & & \ddots & & \\ & & & A_{NN} & \\ S_1 & S_2 & \dots & S_N & \end{bmatrix} \tag{4.6}$$

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 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.

4.1. *The PFAMP algorithm*

The basic PFAMP algorithm is outlined below, followed by a more detailed explanation of the key steps. For complete details, see Mallya *et al.* (1996).

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 (see Section 2). The maximum number of variables that can be eliminated is m_i, but the actual number of pivots done is $p_i \leq m_i$. We use a partial-threshold pivoting strategy 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 may be shared by more than one diagonal block.

(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

$$\begin{array}{cc} & \begin{array}{cc} C_i & C_i' \end{array} \\ \begin{array}{c} R_i \\ R_i' \end{array} & \left[\begin{array}{cc} L_iU_i & U_i' \\ L_i' & F_i \end{array}\right] \end{array} \tag{4.7}$$

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:

$$\begin{array}{cc} & \begin{array}{ccccc} C_1 & C_2 & \ldots & C_N & C' \end{array} \\ \begin{array}{c} R_1 \\ R_2 \\ \vdots \\ R_N \\ R' \end{array} & \left[\begin{array}{ccccc} L_1U_1 & & & & U_1' \\ & L_2U_2 & & & U_2' \\ & & \ddots & & \vdots \\ & & & L_NU_N & U_N' \\ L_1' & L_2' & \ldots & L_N' & F \end{array}\right] \end{array} \tag{4.8}$$

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 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. As this doubly-bordered block-diagonal form makes clear, once values of the variables in the interface problem have been solved for, the remaining triangular solves needed to complete the 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.

Once factorization of all diagonal blocks is complete, the interface matrix is factored. This is carried out by 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, so the factorization of the interface problem can in most cases be regarded as essentially serial. This constitutes a computational bottleneck. Therefore, 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

Name	n	NZ	as	N	$m_{i,max}$	$m_{i,min}$	NI	P
Ethylene_1	10673	80904	0.99	43	3337	1	708	5
Ethylene_2	10353	78004	0.99	43	3017	1	698	5
Ethylene_3	10033	75045	0.99	43	2697	1	708	5
Hydr1c	5308	23752	0.99	4	1449	1282	180	4
Icomp	69174	301465	0.99	4	17393	17168	1057	4
lhr_17k	17576	381975	0.99	6	4301	1586	581	4
lhr_34k	35152	764014	0.99	6	9211	4063	782	4
lhr_71k	70304	1528092	0.99	10	9215	4063	1495	4
Bigequil.smms	3961	21169	0.97	18	887	12	733	4
Wood_7k.smms	3508	16246	0.96	37	897	6	492	4
4cols.smms	11770	43668	0.99	24	1183	33	2210	4
10cols.smms	29496	109588	0.99	66	1216	2	5143	4

TABLE 2. Description of PFAMP test problems. See text for definition of column headings.

size and sparsity of the interface matrix, some solver other than FAMP may in fact be more attractive for performing the factorization.

4.2. *Results and discussion*

In this section, we present results for the performance of the PFAMP solver on several process optimization and simulation problems. 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. 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. 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 2 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), as defined above. Also given is information about the bordered block-diagonal form used, 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. P is the number of processors used for evaluating the parallel performance of PFAMP.

The first three problems involve the optimization of an ethylene plant using NOVA, a chemical process optimization package from Dynamic Optimization Technology Products, Inc. NOVA uses an equation-based approach that requires the solution of a series of large sparse linear systems, which accounts for a large portion of the total computation time. The linear systems arising during optimization with NOVA are in bordered block-diagonal form, allowing the direct use of PFAMP for the solution of these systems. 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 next five problems have been reordered into a bordered block-diagonal form using the Minimum-Net-Cut (MNC) approach (Coon and Stadtherr, 1995). Two of the 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

Name	Source	n	FAMP	PFAMP (1 Proc)	PFAMP (P Proc)
Ethylene_1	NOVA	10673	0.697	0.550	0.297
Ethylene_2	NOVA	10353	0.667	0.510	0.290
Ethylene_3	NOVA	10033	0.628	0.505	0.280
Hydr1c	SPEEDUP	5308	0.258	0.243	0.139
Icomp	SPEEDUP	69174	3.78	4.33	1.72
lhr_17k	SEQUEL	17576	3.62	1.77	0.808
lhr_34k	SEQUEL	35152	7.18	3.81	1.78
lhr_71k	SEQUEL	70304	14.8	7.67	3.04
Bigequil.smms	ASCEND	3961	0.235	0.232	0.149
Wood_7k.smms	ASCEND	3508	0.208	0.205	0.129
4cols.smms	ASCEND	11770	1.14	1.13	0.680
10cols.smms	ASCEND	29496	11.3	3.69	1.81

TABLE 3. FAMP and PFAMP wallclock run times (s). In the last column, P refers to the values in Table 2.

dynamic simulation of a plant that includes several interlinked distillation columns. The other three problems are derived from the prototype simulator SEQUEL (Zitney and Stadtherr, 1988), and are based on light hydrocarbon recovery plants, described by Zitney *et al.* (1996). Neither of the application codes produces directly a matrix in bordered block-diagonal form, so a reordering such as provided by MNC is required.

The final four problems arise from simulation problems solved using ASCEND (Piela *et al.*, 1991), and re-ordered to bordered block-diagonal form using the tear_drop approach (Abbott, 1996). Problem *Bigequil.smms* represents a 9-component, 30-stage distillation column. Problem *Wood_7k* is a complex hydrocarbon separation process. Problems *4cols.smms* and *10cols.smms* involve nine components with four and ten interlinked distillation columns, respectively.

Table 3 shows the performance of PFAMP. We note first, that the single processor performance of PFAMP is usually 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 due to 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, usually 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 each of the three ethylene plant matrices, there are five large diagonal blocks, corresponding to the distillation units, with one of these blocks much larger ($m_i = 3337$) than the others ($1185 \leq m_i \leq 1804$). In the computation, 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 less than two on five processors. It is likely that more efficient processor utilization could be obtained by using a better partition into bordered block-diagonal form.

For MNC-reordered SPEEDUP and SEQUEL matrices, the speedup is around two. MNC achieves the best reordering on the *Icomp* problem, for which it finds four diagonal blocks of roughly 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 particularly 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].

On ASCEND problems, the moderate task granularity helps spread the load over the four processors used, but the size of the interface problem tends to be relatively large, 14-19% of n, as opposed to less than about 7% on the previous problems. The best parallel efficiency was achieved on the largest problem (*10cols.smms*), with a speedup of about two on four processors. This was achieved despite the relatively large size of the interface problem because, for this system, the use of small-grained parallelism within FAMP for solving the interface problem provided a significant speedup (about 1.7). As on the previous problems, this represents a substantial reduction in wallclock time, but is not especially good processor utilization. Overall on *10cols.smms* the use of PFAMP resulted in the reduction of the wallclock time by a factor of six; however only a factor of two of this was due to multiprocessing.

5. Concluding Remarks

We have shown here how a simple multifrontal approach (MFA1P) can be used to efficiently solve, in a supercomputing environment, the sparse linear equation systems that arise in the simulation of equilibrium-stage processes. The solution of such systems is typically the dominant item in the overall simulation time. By taking advantage of the problem's structure, the new approach provides significant improvements over both the standard frontal solver FAMP and the general-purpose multifrontal solver UMFPACK.

The results presented above demonstrate that the parallel frontal solver PFAMP can be effective 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 four- to six-fold performance improvements on some large problems.

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 Kirk Abbott for providing the ASCEND matrices and the tear_drop reorderings.

REFERENCES

Abbott, K. A. 1996 Very Large Scale Modeling. PhD thesis, Dept. of Chemical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Camarda, K. V. & Stadtherr, M. A. 1993 Exploiting small-grained parallelism in chemical process simulation on massively parallel machines. Presented at AIChE Annual Meeting, Paper 142d, St. Louis, MO, November.

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

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

Davis, T. A. & Duff, I. S. 1993 An unsymmetric-pattern multifrontal method for sparse LU factorization. Technical Report TR-93-018, CIS Department, University of Florida, Gainesville, FL (see http://www.cise.ufl.edu/research/tech-reports).

Davis, T. A. & Duff, I. S. 1995. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. Technical Report TR-95-020, CIS Department, University of Florida, Gainesville, FL (see http://www.cise.ufl.edu/research/tech-reports).

Davis, T. A. & Duff, I. S. 1996 An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM J. Matrix Anal. Appl.*, in press (available as Technical Report TR-94-038; see http://www.cise.ufl.edu/ research/tech-reports).

Duff, I. S. 1984 Design features of a frontal code for solving sparse unsymmetric linear systems out-of-core. *SIAM J. Sci. Statist. Comput.* **5**, 270.

Duff, I. S. & Reid, J. K. 1984 The multifrontal solution of unsymmetric sets of linear equations. *SIAM J. Sci. Stat. Comput.* **5**, 633–641.

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

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

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

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

Mallya, J. U. & Stadtherr, M. A. 1996 A multifrontal approach for simulating equilibrium-stage processes on supercomputers. Submitted for publication.

Mallya, J. U., Zitney, S. E., Choudhary, S. & Stadtherr, M. A. 1996 A parallel frontal solver for large scale process simulation and optimization. Submitted for publication.

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

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

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

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

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

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

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

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

Zitney, S. E., Camarda, K. V. & Stadtherr, M. A. 1994 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. Rippin, J. C. Hale & J. F. Davis), pp. 463-468. CACHE Corp., Austin, TX.

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

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