Multifrontal Techniques for Chemical Process Simulation on Supercomputers

S. E. Zitney[1], J. Mallya [2], T. A. Davis[3], and M. A. Stadtherr[2]†

1) Cray Research, Inc., 655 E. Lone Oak Drive, Eagan, MN 55121, USA
2) Department of Chemical Engineering, University of Illinois, 600 S. Mathews Avenue, Urbana, IL 61801, USA
3) Computer and Information Sciences Department, University of Florida, Gainesville, FL 32611, USA

ABSTRACT

A critical computational step in large-scale process simulation using rigorous equation-based models is the solution of a sparse linear equation system. Traditional sparse solvers based on indirect addressing are not effective on supercomputers because they do not vectorize well. By relying on vectorized dense matrix kernels, the multifrontal and frontal methods provide much better performance, as demonstrated using several examples. On problems with good initial matrix orderings the frontal method is most effective, while without a good initial ordering the multifrontal method is attractive.

INTRODUCTION

Steady-state or dynamic simulation tools are widely used in the design, optimization, and operation of chemical processes. Increasingly these tools are being used industrially in very large-scale, plant-wide studies based on rigorous physical and chemical models. These trends have been made possible by impressive gains in computer performance and advances in numerical methods. Leading the way, modern supercomputers offer vector and parallel processing architectures for solving problems that, until now, could not be solved with other computational tools. Today, this leading-edge technology is increasingly seen at price levels that make it more widely available to process systems engineers. Thus, the use of supercomputer technology in process simulation is more practicable than ever before, and provides opportunities to solve larger-scale and more realistic plant models than ever before. However, since most current methods for solving process simulation problems were developed for use on conventional serial machines, they usually do not effectively take advantage of the vector and parallel processing architecture of supercomputers. Thus, to exploit supercomputing (as opposed to just using a supercomputer) requires the rethinking of the solution strategies used in process simulation. In this paper, we consider the sparse linear equation solving strategies used in this context.

†Author to whom correspondence should be addressed

BACKGROUND

In large-scale process simulation using rigorous equation-based models, the key computational step, representing as much as 90% of the computation time on industrial problems, is often the solution of a large sparse system of linear equations. Process simulation matrices, however, do not have any of the desirable structural or numerical properties, such as symmetry, positive definiteness, diagonal dominance, and bandedness, often associated with sparse matrices, and usually exploited in developing efficient algorithms for vector and parallel computation. Recent work (Zitney, 1992; Zitney and Stadtherr, 1993a,b; Zitney *et al.*, 1993) has demonstrated the potential of the *frontal method* as a sparse linear equation solver for process simulation problems. In fact, an implementation of the frontal method (FAMP), developed at the University of Illinois and Cray Research, Inc., has now been incorporated in Cray Research versions of commercially used tools such as ASPEN PLUS, BATCHFRAC, RATEFRAC, and SPEEDUP (Aspen Technology, Inc.).

The frontal method is effective because, unlike traditional general-purpose sparse matrix solvers such as MA28 (Harwell) and LU1SOL (University of Illinois) that rely on indirect addressing, it makes use of easily vectorizable full matrix operations performed on a series of frontal matrices. 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 the frontal matrices, a large number of unnecessary operations on zeros are performed, thus lowering overall performance.

In this paper we consider the *multifrontal method* as an alternative to the frontal method. The multifrontal method is a generalization of the frontal method, and was originally developed for symmetric systems. Like the frontal method, it also exploits low-level parallelism and vectorization through the use of dense matrix kernels on frontal matrices (e.g., Amestoy and Duff, 1989). However, the frontal matrices are generally smaller and denser than in the frontal method. Furthermore the multifrontal method offers more opportunities for exploiting a higher level of parallelism than the frontal method. Though this classical multifrontal approach can be applied to unsymmetric systems (Duff and Reid, 1984), this has met with only limited success, as noted by Liu (1992) in his review of work on the symmetric problem. Recently a new unsymmetric-pattern multifrontal algorithm has been described by Davis and Duff (1993). In the new algorithm, unlike the classical multifrontal approach, frontal matrices are permitted to be rectangular and the unsymmetric structure is accounted for explicitly through the use of a directed acyclic graph. The performance of this new multifrontal method in process simulation problems on a CRAY Y-MP supercomputer is compared here with the that of the frontal method (FAMP), as well as that of the conventional code MA28.

FRONTAL METHOD

The use of the frontal method is demonstrated in a simple example given by Zitney and Stadtherr (1993a). The basic idea is to restrict elimination operations to a frontal matrix, on which dense matrix operations are performed. Beginning with row 1, equations are assembled (added) into the frontal matrix until some variable or variables become fully summed (i.e., all their nonzero elements appear in the frontal matrix). Partial pivoting is then applied, the fully summed variable or variables are eliminated, and the pivot row(s) and column(s) removed from the frontal matrix. The assembly process then begins again and the method proceeds to alternate between assembly and elimination phases until the matrix factorization is complete. To implement the frontal method, we use here the code FAMP, which originated at the University of Illinois (Zitney and Stadtherr, 1993a) and was later extended at Cray Research to include the use of BLAS2 and BLAS3 dense matrix kernels, as well as an out-of-core option for solving very large-scale problems, and separate analyze-factorize, factorize, and solve options.

MULTIFRONTAL METHOD

The unsymmetric-pattern multifrontal method factors a sparse, unsymmetric matrix with a sequence of dense frontal matrices, each of which corresponds to one or more steps of the overall LU factorization. To demonstrate the basic idea in the unsymmetric-pattern multifrontal method we use the example shown in Figure 1.

$$\begin{array}{c|ccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & \times & & & \times & \times & & \\
2 & \times & & & \times & \times & \times & \\
3 & \times & \times & \times & & & & \times \\
4 & \times & \times & \times & & \times & & \times \\
5 & & \times & \times & & \times & \times & \\
6 & & & & & & \times & \times \\
7 & \times & \times & & & \times & & \times \\
\hline
\end{array}$$

Figure 1. Matrix used in example.

An initial pivot element is chosen, say element (1,1). The first frontal matrix is then started with this pivot row and column and all contributions to them. This leads to the matrix shown in Figure 2(a).

$$\begin{array}{c|c|cc|}
 & 1 & 4 & 5 \\
\hline
1 & \times & \times & \times \\
\hline
2 & \times & & \\
3 & \times & & \\
4 & \times & & \\
7 & \times & & \\
\hline
\end{array}
\qquad
\begin{array}{c|c|cc|}
 & 1 & 4 & 5 \\
\hline
1 & u & u & u \\
\hline
2 & \ell & \times & \times \\
3 & \ell & \times & \times \\
4 & \ell & \times & \times \\
7 & \ell & \times & \times \\
\hline
\end{array}$$

(a) (b)

Figure 2. First frontal matrix in multifrontal method if first pivot is (1,1). (a) Before factorization; (b) After factorization.

The pivot operation is then performed using a dense matrix kernel. As shown in Figure 2(b), this computes a row of U and a column of L, as well as a contribution block (in the nonpivot rows and columns) that is saved for later use. Another pivot is now selected and a new frontal matrix begun. Say element (3,2) is selected, and note that this implies an unsymmetric permutation of the matrix. The frontal matrix is started with row 3 and column 2; all contributions to this row and column must now be assembled, both those from the original matrix elements and those from the contribution block of the previous frontal matrix. This leads to the matrix shown in Figure 3(a). Note that since all contributions to row 4 and column 3 can also be assembled into this frontal matrix, an additional pivot (4,3) can be performed, resulting in Figure 3(b). The search for additional pivots that can be performed within a frontal matrix, or that can be performed with only a small growth of the frontal matrix is significant since it allows the use of BLAS3 as opposed to BLAS2 kernels.

$$
\begin{array}{c|cc|ccc|}
 & 2 & 3 & 4 & 5 & 7 \\
\hline
3 & \times & \times & \times & \times & \times \\
4 & \times & \times & \times & \times & \times \\
\hline
5 & \times & \times & & & \\
7 & \times & & & & \\
\hline
\end{array}
\qquad
\begin{array}{c|cc|ccc|}
 & 2 & 3 & 4 & 5 & 7 \\
\hline
3 & u & u & u & u & u \\
4 & \ell & u & u & u & u \\
\hline
5 & \ell & \ell & \times & \times & \times \\
7 & \ell & \ell & \times & \times & \times \\
\hline
\end{array}
$$

(a) (b)

Figure 3. Next frontal matrix if (3,2) is the next pivot. Note that pivoting on (4,3) can also be done in this frontal matrix. (a) Before factorization of pivot block; (b) After factorization.

Frontal matrices continue to be assembled and pivot operations performed in them until the L and U factors are completed. In the next section we focus on the implementation of the multifrontal method and how it differs from the frontal method.

MULTIFRONTAL VS. FRONTAL METHOD

We discuss here only the sequential implementation of the unsymmetric-pattern multifrontal method of Davis and Duff (1993), as embodied in the UMFPACK library (Davis, 1993). A copy of UMFPACK may be obtained via anonymous ftp to ftp.cis.ufl.edu:pub/umfpack (it is free for non-commercial use only).

In the frontal method, the pivot order of the columns is dependent on the row ordering, and the pivot order of the rows can vary only within certain constraints. A row can become pivotal any time between the time it is entered into the frontal matrix and the end of the factorization. Rows are entered into the frontal matrix in a predefined order. The pivot column ordering depends solely on the initial preordering of the rows. Unlike the frontal method, UMFPACK finds both a row and column pivot ordering as the matrix is factorized. No preordering, or partial preordering, is used.

At the start of the factorization, no frontal matrix exists. UMFPACK starts a new frontal matrix with a global Markowitz-style pivot search. Suppose row i and column j are selected as the k-th pivot row and column. A new $c_j \times r_i$ frontal matrix is formed in an working array of size $(G \cdot c_j) \times (G \cdot r_i)$, where the column degree c_j and row degree r_i are the number of nonzeros in column j and row i of the partially-factorized submatrix, and $G \geq 1$ is a parameter controlling frontal matrix growth. The pivot row and column, some of the entries of the original matrix, and some of the updates from previous frontal matrices are assembled (added) into the current frontal matrix. Not all original entries or previous frontal updates can be assembled into the current frontal matrix (unless the current frontal matrix is as large as the $(n - k) \times (n - k)$ submatrix yet to be factorized). The *approximate degree update* phase determines upper and lower bounds on the number of nonzeros in the c_j rows and r_i columns affected by this frontal matrix. These bounds are used for subsequent pivot searches (it is too costly to maintain the true degrees of each row and column).

The frontal matrix is augmented with additional pivots, using a local pivot search of rows and columns within the frontal matrix. Augmentation continues until a subsequent pivot would cause the size of the frontal matrix (including all of its pivot rows and columns) to become larger than the working array. Figure 4 shows 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 factorized within E_k.

$$
\begin{array}{c}
 \quad C_k' \qquad C_k'' \\
\begin{array}{c} R_k' \\ R_k'' \end{array}
\left[\begin{array}{cc} F_k & B_k \\ T_k & D_k \end{array}\right]
\end{array}
$$

Figure 4.

The frontal matrix is labeled with the ordered sets R_k' and C_k' (the g_k pivot rows and columns, respectively), and with the sets R_k'' and C_k'' (the non-pivotal rows and columns that contain entries updated by the g_k pivots). Contributions to the matrices F_k, B_k, and T_k must be fully assembled; however, D_k may hold only a partial summation of the original matrix elements and contributions from previous frontal matrices. The pivot block F_k is now factored ($F_k = L_k'U_k'$), thus computing the block-column L_k'' of L and block-row U_k'' of U, and replacing D_k with the Schur complement $D_k' = D_k - L_k''U_k''$. Fig. 5 shows the factorized frontal matrix, where the notation $L_k' \backslash U_k'$ indicates two matrices packed in the same array.

$$\begin{array}{c|cc|}
 & C_k' & C_k'' \\
R_k' & L_k' \backslash U_k' & U_k'' \\
R_k'' & L_k'' & D_k'
\end{array}$$

Figure 5.

The goal of multifrontal methods is to replace indirect addressing in innermost loops with dense matrix kernels such as the Level-3 BLAS (Dongarra *et al.*, 1990). A frontal matrix E_k can be factorized with the Level-3 BLAS if g_k is greater than one.

The method generates a directed acyclic graph (a *dag*) during factorization that describes the assembly process and precedence between the frontal matrices. This dag is referred to as the assembly dag, since it functions in a similar role as the assembly tree in the classical, symmetric-pattern multifrontal method (MA37) (Duff and Reid, 1984). In contrast, the assembly dag for the frontal method would simply be a linear chain, although the frontal method does not make use of such a dag. These dags or trees also express the parallelism inherent in the method, if each frontal matrix factorization is considered a separate task. Thus, the frontal method cannot exploit parallelism between the tasks, whereas the multifrontal methods can. UMFPACK is better suited to unsymmetric matrices than MA37, and simulations show comparable levels of exploitable parallelism in the two methods (Hadfield and Davis, 1992).

The frontal method only stores the D_k term in its single working array, writing the pivot rows and columns into a separate data structure for the LU factors. The multifrontal method stores all of E_k until the factorization of E_k is complete. Thus, if the assembly dag in UMFPACK were forced to be a linear chain (as in the frontal method), UMFPACK would still require more than one frontal matrix (unless the single frontal matrix is $n \times n$, which defeats the purpose).

RESULTS AND DISCUSSION

Table 1 presents results for the comparison of UMFPACK, FAMP, and MA28. All times are given in cpu seconds on one processor of a CRAY Y-MP system. The analyze/factor (AF) time is that required to determine a pivot sequence and factor the matrix; the factor (F) time is that required to factor the matrix given a pivot sequence; and the solution (S) time is that required to obtain a solution given the LU factors and a right-hand-side vector. The memory figures reported (in megawords) represent the minimum required; more memory was actually used in these runs. Attempting to use the minimum memory would significantly lengthen factorization times. Two memory figures for FAMP are given: *oc* refers to use of the out-of-core option and *ic* to the memory that would have been required to keep the problem in core, as is done in the other two methods. The number of nonzeros in the LU factors is provided as a measure of the amount of numerical computation performed by each method. In MA28 and UMFPACK, the relative threshold pivot tolerance used was 0.1; FAMP uses partial pivoting. The growth factor G in UMFPACK was set at 3. No *a priori* reordering of the matrices was performed outside the packages used. Cases marked NS were not solved due to excessive computational requirement.

Results for six process simulation problems are shown. *Rdist1* is a reactive distillation problem described by Zitney (1992), *hydr1c* and *extr1b* are dynamic simulation problems described by Zitney *et al.* (1993), and the *lhr* problems are extensions of the light hydrocarbon recovery process described by Zitney and Stadtherr (1993a). On these problems, both the frontal (FAMP) and multifrontal (UMFPACK) methods are significantly faster than MA28, reflecting the use of vectorized dense matrix kernels. Also, on these problems, the frontal method outperforms the multifrontal method. This is due to the presence of a good initial row ordering in these matrices. The performance of the frontal method depends strongly on the initial row ordering; thus in general some *a priori* reordering step is required, though there appears to be no consistently good technique for doing this on unsymmetric problems. The time to perform this reordering would have to be added to the frontal method's AF time. No such reordering is needed in the multifrontal method. It was fortuitous in these cases that the matrices had a good (though probably not optimal) initial ordering. Such a good ordering is 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. Whether or not this will occur depends on the simulation software that generates the matrix and on the unit and stream numbers assigned by the user in the input to the simulation package. Some problems, such as the ones used here, that primarily involve separation columns, are especially likely to have a good initial ordering. However, good initial orderings cannot be guaranteed; thus there is a need either for *a priori* reordering schemes, or techniques, like the multifrontal method, that do not require them.

To see the effect of not having such a good initial ordering, five additional problems were considered (see Davis and Duff (1993) for the sources of these problems). These are two circuit simulation problems (*add32* and *mem+*), an electrical power system problem (*gematl1*), a computational fluid dynamics problem (*lns_3937*), and a reservoir simulation problem (*sherman5*). On these problems the potential of the multifrontal method can be seen. On one problem (*mem+*), the frontal method is not even competitive, due to the extremely large frontal matrix required to factor the matrix. The performance of the frontal method could be improved using an *a priori*

Table 1. Comparison of sparse matrix solvers. All times are in cpu seconds. Memory is in megawords. See text for discussion and further definition of terms.

Matrix											
Name	rdist1	hydr1c	extr1b	lhr_4k	lhr_17k	lhr_70k	lns_3937	sherman5	add32	mem+	gemat11
Order	4134	5308	2836	4101	17576	70304	3937	3312	4960	17758	4929
Nonzeros	94408	25276	12094	82682	381975	1528092	25407	20793	19848	99147	33108
AF Time											
MA28	104.79	7.06	4.74	52.81	522.01	NS	20.87	6.40	0.33	2.05	0.75
UMFPACK	1.78	1.43	0.67	4.73	16.54	73.06	3.24	1.40	0.58	3.10	0.66
FAMP	0.43	0.23	0.09	0.46	2.14	8.00	5.63	1.90	0.67	NS	0.48
F Time											
MA28	5.21	0.25	0.12	2.31	27.67	NS	2.30	0.76	0.14	0.81	0.23
UMFPACK	0.56	0.31	0.15	0.71	3.14	13.67	1.07	0.50	0.21	0.96	0.24
FAMP	0.42	0.20	0.08	0.44	1.99	7.37	5.57	1.87	0.59	NS	0.44
S Time											
MA28	0.035	0.025	0.013	0.026	0.135	NS	0.027	0.016	0.024	0.087	0.027
UMFPACK	0.022	0.026	0.014	0.023	0.101	0.415	0.030	0.016	0.023	0.094	0.020
FAMP	0.021	0.019	0.010	0.023	0.096	0.369	0.052	0.021	0.016	NS	0.019
NZ in LU											
MA28	841900	73954	37541	459626	3445027	NS	423305	167256	23914	126150	51727
UMFPACK	573287	121901	65062	495503	2225140	9667709	761194	426616	46953	202888	97195
FAMP	624140	133399	51735	517004	1937363	7181471	2268987	799907	31388	NS	257346
Memory											
MA28	2.166	0.250	0.134	1.232	7.866	NS	1.130	0.464	0.156	0.681	0.219
UMFPACK	1.185	0.395	0.233	1.217	4.238	16.431	1.558	0.864	0.287	1.316	0.394
FAMP-ic	1.412	0.391	0.152	1.250	4.756	16.922	4.644	1.458	4.779	NS	0.891
-oc	0.166	0.126	0.053	0.218	0.885	2.563	2.383	0.663	4.760	NS	0.637

reordering scheme, but unless this can be done cost effectively, the multifrontal approach may still be more attractive.

Finally it should be noted that all runs reported here were made on a single vector processor. Thus, while vectorization was exploited, there are still higher levels of parallelism that have not been exploited. As discussed above, the multifrontal method is better suited than the frontal method to exploiting parallelism at the task level. Looking to the future, it is this feature of the multifrontal method that may make it especially useful in solving process simulation and other problems on supercomputers.

Acknowledgements--This work was supported in part by the National Science Foundation under Grants DDM-9024946 at the University of Illinois and ACS-9111263 at the University of Florida, and, through the allocation of supercomputer resources, by Florida State University, the National Center for Supercomputing Applications at the University of Illinois, and Cray Research, Inc. Steve Hamm at Motorola provided the *mem+* and *add32* matrices.

REFERENCES

Amestoy, P. and I. S. Duff, Vectorization of a Multiprocessor Multifrontal Code, *Int. J. Supercomput. Appl.*, **3**, 41 (1989).

Davis, T. A., User's Guide for the Unsymmetric-Pattern Multifrontal Package (UMFPACK), Technical Report TR-93-020, CIS Department, University of Florida, Gainesville, FL (1993).

Davis, T. A. and I. S. Duff, An Unsymmetric-Pattern Multifrontal Method for Sparse LU Factorization, Technical Report TR-93-018, CIS Department, University of Florida, Gainesville, FL (1993).

Dongarra, J. J., J. Du Croz, and S. Hammarling, A Set of Level 3 Basic Linear Algebra Subprograms, *ACM Trans. Math. Softw.*, **16**, 1-17 (1990).

Duff, I. S. and J. K. Reid, The Multifrontal Solution of Unsymmetric Sets of Linear Equations, *SIAM J. Sci. Stat. Comput.*, **5**, 633-641 (1984).

Hadfield, S. and T. A. Davis, Analysis of Potential Parallel Implementations of the Unsymmetric-Pattern Multifrontal Method for Sparse LU Factorization, Technical Report TR-92-017, CIS Department, University of Florida, Gainesville, FL (1993).

Liu, J. W. H., The Multifrontal Method for Sparse Matrix Solution: Theory and Practice, *SIAM Review*, **34**, 82-109 (1992).

Zitney, S. E., Sparse Matrix Methods for Chemical Process Separation Calculations on Supercomputers, in *Proc. Supercomputing '92*, pp. 414-423, IEEE Computer Society Press, Los Alamitos, CA (1992).

Zitney, S. E. and M. A. Stadtherr, Frontal Algorithms for Equation-Based Chemical Process Flowsheeting on Vector and Parallel Computers, *Comput. Chem. Eng.*, **17**, 319-338 (1993a).

Zitney, S. E. and M. A. Stadtherr, Supercomputing Strategies for the Design and Analysis of Complex Separation Systems, *Ind. Eng. Chem. Res.*, **32**, 604-612 (1993b).

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 (FOCAPO-II)*, CACHE Corp., Ann Arbor, MI, in press (1993).