A Parallel Frontal Solver for Process Simulation

J. Mallya* M.A. Stadtherr† S.E. Zitney‡ S. Choudhary§

Abstract

The solution of large-scale chemical processes is often dominated by the time spent to solve large sparse systems of linear equations. We describe here a parallel frontal algorithm which significantly reduces the wallclock time to solve these linear equation systems. The algorithm is based on a bordered block-diagonal form arising in equation-based process simulation, and exploits vector and multiprocessing by using a multilevel approach in which frontal elimination is used for partial factorization of each diagonal block. Results on several large-scale process simulation and optimization problems are presented.

1 Introduction

The solution of realistic, industrial-scale process simulation and optimization problems is computationally very intense, and often requires high performance computing (HPC) technology to be done in a timely manner. To better use HPC technology in process simulation requires the use of techniques that effectively take advantage of vector and/or parallel processing.

In large-scale process simulation and optimization using rigorous equation-based models, the key computational step is the solution of sparse linear equation systems ($Ax = b$ and $A^T x = b$), representing as much as 80% of the total computation time. Zitney and Stadtherr [1] have demonstrated the usefulness of a frontal based linear solver scheme which exploits the vector processing capabilities of the CRAY-C90 vector/parallel supercomputer. By using this frontal solver and addressing other implementation issues, they reduced the time needed to solve a dynamic simulation problem at Bayer AG from 18 hours to 21 minutes [2]. This implementation of the frontal algorithm (called FAMP) is currently used in CRAY versions of commercially used simulation packages such as SPEEDUP (Aspen Technology, Inc.) and ASPEN PLUS (Aspen Technology, Inc.)

FAMP is effective on vector machines since most of the computations involved can be performed using efficiently vectorized dense matrix kernels. However, this solver does not well exploit the multiprocessing architecture of parallel/vector supercomputers. In this paper we describe a new parallel frontal solver (PFAMP) that exploits both the vector *and* parallel processing architectures of modern supercomputers. Results demonstrate that the approach described leads to significant reductions in the wallclock time required to solve the sparse linear systems arising in large scale process simulation and optimization.

*Cray Research, 655E Lone Oak Drive, Eagan, MN 55121
†Department of Chemical Engineering, University of Notre Dame, Notre Dame, IN 46556
‡Cray Research (now with AspenTech UK Ltd., Castle Park, Cambridge CB3 0AX)
§Cray Research, 655E Lone Oak Drive, Eagan, MN 55121

2 Background

The frontal elimination scheme used here can be briefly outlined 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 be more precise, it is the LU factors of the permuted matrix PAQ that have been found, where P is a row permutation matrix determined by the partial pivoting, and Q is a column permutation matrix determined by the order in which the columns become fully summed. Thus the solution to $Ax = b$ is found as the solution to the equivalent system $PAQQ^Tx = LUQ^Tx = Pb$, which is solved by forward substitution to solve $Ly = Pb$ for y, back substitution to solve $Uw = y$ for w, and finally the permutation $x = Qw$.

In the frontal algorithm, the most expensive stage computationally is the outer-product update of the frontal matrices. When executed on a single vector processor, FAMP performs efficiently because the outer-product update is readily vectorized. However, the main deficiency with the frontal code FAMP 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. Mallya [3] has shown that only very limited speedups (often only about 1.2 on four processors) could be achieved by adopting this strategy, primarily because the small-grained parallel tasks generated are simply not large enough to overcome the synchronization cost and the overhead associated with invoking multiple processors on the C90.

3 Coarse-grained Parallelism

We overcome the deficiencies of the frontal code FAMP by using a coarse-grained parallel approach in which frontal elimination is performed simultaneously on multiple independent or loosely connected blocks. Consider a matrix in singly-bordered block-diagonal form:

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

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 occur naturally in this form, as described in detail by Westerberg and Berna [4]. 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. This approach is similar to the multiple fronts approach for solving finite element problems described by Duff and Scott [5].

4 Parallel Frontal Algorithm (PFAMP)

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 pivoting scheme used is described in detail in Section 4.1.

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

$$(2) \qquad \begin{array}{c} \\ R_i \\ R_i' \end{array} \begin{array}{c} \begin{array}{cc} C_i & C_i' \end{array} \\ \begin{bmatrix} L_iU_i & U_i' \\ L_i' & F_i \end{bmatrix} \end{array}$$

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 R_i^A, the row index set of A_{ii}. R_i' contains all row indices from R_i^S, the row index set of nonzero rows of S_i, as well as the indices of 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:

(3)
$$\begin{array}{c|cccc|c} & C_1 & C_2 & \dots & C_N & C' \\ R_1 & L_1U_1 & & & & U'_1 \\ R_2 & & L_2U_2 & & & U'_2 \\ \vdots & & & \ddots & & \vdots \\ R_N & & & & L_NU_N & U'_N \\ \hline R' & L'_1 & L'_2 & \dots & L'_N & F \end{array}$$

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 frontal 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.

4.1 Numerical Pivoting

We use a partial-threshold pivoting strategy to maintain numerical stability while choosing pivots. 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 may be shared by more than one diagonal block. Partial pivoting is carried out to find the largest magnitude element (say, $a_{i^*,p}$) in the pivot column p while limiting the search to the rows that belong to the diagonal block A_{ii} (so $i^* \in R_i^A$). This element is chosen as the pivot element if it satisfies a threshold pivot tolerance criterion with respect to the largest magnitude element (say, $a_{j^*,p}$) in the *entire* pivot column (so $j^* \in R_i^A \cup R_i^S$). That is, if $a_{i^*,p} \geq u \times a_{j^*,p}$ then $a_{i^*,p}$ is chosen as pivot, where u is a threshold pivot tolerance in the range $0 < u \leq 1$. If a pivot search does not find an element that satisfies this criterion, then the elimination of that variable is delayed and the pivot column becomes part of the interface problem. This increases the size of the mostly serial interface problem; however our computational experiments indicates that the increase in size is very small compared to n, the overall problem size.

4.2 Load Balancing and Reordering

As discussed above, 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 an ideal ordering, with each diagonal block presenting an equal workload and no interface matrix (i.e., a block diagonal matrix), the speedup of the algorithm would in principle scale linearly. However, this ideal ordering rarely exists.

In a natural unit-stream structure, the interface problem size may be small, but the number of equations in different units may vary widely thereby giving rise to blocks of various sizes. This may be handled by combining some of the smaller blocks into larger blocks. In our experience, even that may not be possible as one block may be larger than all the rest combined together. It is also possible to break the larger block into smaller diagonal blocks with the disadvantage of increasing the interface problem size.

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 structurally *symmetric* matrices, there are various approaches that can be used to try to get an appropriate matrix reordering (e.g., [6], [7], [8]). These are generally based on solving graph partitioning, bisection or min-cut problems, often in the context of nested dissection applied to finite element problems. Such methods can be applied to a structurally *asymmetric* matrix A by applying them to the structure of the symmetric matrix $A + A^T$, and this may provide satisfactory results if the degree of asymmetry is low. However, when the degree of asymmetry is 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 deal with structurally asymmetric problems, one technique that can be used is the min-net-cut (MNC) approach of Coon and Stadtherr [9]. This technique is designed specifically to address the issues of load balancing and interface problem size. It is based on recursive bisection of a bipartite graph model of the asymmetric matrix. Since a bipartite graph model is used, the algorithm can consider unsymmetric permutations of rows and columns while still providing a structurally stable reordering. 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. 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.

5 Results and Discussion

In this section, we present results for the performance of the PFAMP solver on two sets of 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

TABLE 1
Description of Test Problems.

No.	Name	n	NZ	as	N	$m_{i,max}$	$m_{i,min}$	NI
1	ethylene_1	10673	80904	0.99	43	3337	1	708
2	ethylene_2	10353	78004	0.99	43	3017	1	698
3	ethylene_3	10033	75045	0.99	43	2697	1	708
4	hydr1c	5308	23752	0.99	4	1449	1282	180
5	Icomp	69174	301465	0.99	4	17393	17168	1057
6	lhr_34k	35152	764014	0.99	6	9211	4063	782
7	lhr_71k	70304	1528092	0.99	10	9215	4063	1495

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 $u = 0.1$ was used in PFAMP to maintain numerical stability, which was monitored using the 2-norm of the residual $b - Ax$.

The first set of problems (Problems 1–3) come from 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. This application code produces directly a matrix in bordered block-diagonal form based on the natural unit-stream structure of the problem. Thus, no reordering is done prior to use of PFAMP for the solution of these systems. The second set of problems includes dynamic simulation problems (Problems 4–5) solved using SPEEDUP (Aspen Technology, Inc.) and steady-state simulation problems solved using SEQUEL [10]. Neither of these application codes directly produces a matrix in bordered block-diagonal form, so the MNC reordering is used prior to use of PFAMP. Reordering time is not included in the results, as in these applications the reordering can be reused in the factorization of many systems of similar structure.

In Table 1, each matrix is identified by problem number, 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 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.

Timing comparisons are given for the first set of problems in Table 2, and for the second in Table 3. We note first that the single processor performance of PFAMP is somewhat better then that of FAMP. This is because FAMP keeps the variables associated with the interface in the front 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.

For Problems 1–3, there are five large diagonal blocks in the matrices, with one of these

TABLE 2
Timings (wallclock) for Problems 1–3 in milliseconds.

No.	FAMP	PFAMP (1 CPU)	PFAMP (5 CPUs)
1	697	550	297
2	667	510	290
3	628	505	280

TABLE 3
Timings (wallclock) for Problems 4–7 in milliseconds.

No.	FAMP	FAMP (MNC)	PFAMP (MNC; 1 CPU)	PFAMP (MNC; 4 CPUs)
4	291	258	243	139
5	3990	3777	4328	1716
6	7402	7178	3813	1783
7	14960	14797	7670	3036

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 a bottleneck. This, together with the solution of the interface problem, another bottleneck, 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.

Problems 4–7 were reordered using MNC. Thus, since the performance of the frontal solver FAMP usually depends on the row ordering in the matrix, it was run using both the original ordering and the MNC ordering. The difference in the performance of FAMP when using the different orderings was not significant. MNC achieves the best ordering for Problem 5, 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, even a small interface problem can substantially decrease the speedup attained.

6 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 four- to six-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 reordering used is far from optimal, and no attempt was made to find better orderings or compare the reordering approaches used. 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. The cost of a good ordering must be weighed against the number of times the 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 ordering for PFAMP to exploit might have substantial long term paybacks.

References

[1] S. E. Zitney and M. A. Stadtherr *Frontal Algorithms for Equation-Based Chemical Process Flowsheeting on Vector and Parallel Computers*, Computers and Chemical Engineering, 17 (1993), pp. 319–338.

[2] S. E. Zitney, L. Brüll, L. Lang and R. Zeller *Plantwide Dynamic Simulation on Supercomputers for Modeling a Bayer Distillation Process*, AIChE Symposium Series, 91(304), 1995, pp. 313-316.

[3] J. U. Mallya *Vector and Parallel Algorithms for Chemical Process Simulation on Supercomputers*, PhD thesis, Dept. of Chemical Engr., Univ. of Illinois, Urbana, Illinois, 1996.

[4] A. W. Westerberg and T. J. Berna *Decomposition of Very Large-Scale Newton-Raphson based Flowsheeting Problems*, Computers and Chemical Engineering, 2, 61, 1978.

[5] I. S. Duff and J. A. Scott *The use of Multiple Fronts in Gaussian Elimination*, Technical Report, RAL, 94-040, Rutherford Appleton Laboratory, Oxon, UK, 1994.

[6] B. W. Kernighan and S. Lin *An Efficient Heuristic Procedure for Partitioning Graphs*, Bell System Tech. J., 49, 1970, pp. 291-307.

[7] C. E. Leiserson and J. G. Lewis *Orderings for Parallel Sparse Symmetric Factorization*, In Rodrigue G., editor, Parallel Processing for Scientific Computing, SIAM, Philadelphia, 1989, pp. 27–31.

[8] G. Karypis and V. Kumar *Multilevel k-way Partitioning Scheme for Irregular Graphs*, Technical Report 95-064, Dept. of Computer Science, Univ. of Minnesota, 1995.

[9] A. B. Coon and M. .A. Stadtherr *Generalized Block-Tridiagonal Matrix Orderings for Parallel Computation in Process Flowsheeting*, Computers and Chemical Engineering, 19, 1995, pp. 787-805.

[10] S. E. Zitney and M. A. Stadtherr *Computational Experiments in Equation-Based Chemical Process Flowsheeting*, Computers and Chemical Engineering, 12 (1988), pp. 1171–1186.