A Multifrontal Approach for Simulating Equilibrium-Stage Processes on Supercomputers

Jayarama U. Mallya
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

Revised, October 1996

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

Abstract

For the simulation of complex equilibrium-stage operations, the overall computing time is often dominated by the solution of large, sparse systems of linear equations. If the modeling equations for such separation systems are grouped by equilibrium stage, the linear systems take on an almost banded form with relatively few off-band elements. We present here a simple multifrontal approach for solving such linear systems on supercomputers. Like the frontal approach, these solvers exploit supercomputing technology by treating parts of the sparse matrix as full, thereby allowing arithmetic operations to be performed with highly vectorized and optimized BLAS dense matrix kernels. In addition, these solvers exploit the almost banded structure of the distillation matrices by using a modified threshold pivot search strategy that attempts to maintain the desirable structure of the matrix during the solution process. Results indicate that this multifrontal approach provides substantial savings in solution time compared to other techniques often used.

1 Introduction

Steady-state or dynamic simulation tools are widely used in the design, optimization, and operation of equilibrium-stage processes. Increasingly these tools are being used industrially to model very large-scale processes (Zitney *et al.*, 1995). This trend has been made possible in part by impressive gains in computer performance and advances in numerical methods. Today, supercomputing technology, offering parallel and/or vector processing architectures, is increasingly seen at price levels that make it available to process systems engineers. Thus, the use of supercomputers 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 typically do not effectively take advantage of the vector and parallel processing architecture of supercomputers. Thus, exploiting this technology in process simulation requires a rethinking of the solution strategies used. In this paper, we consider the sparse linear equation solving strategies used in this context.

In simulating complex chemical processes, the overall simulation time is often dominated by the repeated solution of large, sparse systems of linear equations. In general, the linear system or systems that arise, may not have any of the desirable structural and numerical properties, such as numerical or structural symmetry, positive definiteness and diagonal dominance, that are often exploited in solving large, sparse linear systems. Nevertheless, these systems do have some structural properties that can be exploited. One such structural property is that, due to the unit-stream nature of the problem, simulation matrices tend to be block-banded, usually with several off-band blocks representing recycle and/or feedforward streams. This assumes that one has grouped the equations and variables by unit and stream, respectively, and that some reasonable attempt has been made to order adjacent units in the process consecutively in the matrix. For simulation problems that primarily involve distillation or other equilibrium-stage operations, the block-band is in fact typically a block-tridiagonal, assuming the modeling equations are grouped by equilibrium stage. Off-band elements may occur for various reasons, including interlinking streams in multicolumn problems, pumparounds in single column problems, and the way in which design specifications are handled. Special-purpose solvers for block-tridiagonal and almost-block-tridiagonal systems have often been employed in this context in order to take advantage of this structure. For the solution of equilibrium-stage operation on supercomputers, Zitney and Stadtherr (1993b), Zitney (1990), and Zitney *et al.*(1995) have demonstrated the effectiveness of the frontal method for solving the sparse linear systems. In principle, a multifrontal approach may have advantages over the frontal method, as discussed in more detail below. However, when applied to equilibrium-stage separation problems, available multifrontal solvers do not take good advantage of the structure of the problem, and may in fact spend considerable effort in finding a suitable pivot sequence that may not be desirable. It is important to exploit all information which comes from knowing the structure of the underlying problems. Thus, in this article, we present a simple multifrontal approach designed specifically to take advantage of the structure of such matrices. We compare the performance of this approach with an implementation of the frontal algorithm (FAMP), with a general-purpose multifrontal solver (MA38), and with MA28, a well-known conventional solver.

2 Background

The frontal method is a 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. Today it 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. Implementations of the frontal method developed specifically for use in the process simulation context have been described by Zitney (1992) and Zitney and Stadtherr (1993a,b). One of these codes, FAMP, developed at University of Illinois and later extended at Cray Research, Inc., has now been incorporated in CRAY versions of popular commercial simulation codes, such as ASPEN PLUS, SPEEDUP, RATEFRAC, and BATCHFRAC (Aspen Technology, Inc.).

Because process simulation matrices are not truly banded, the frontal matrix can become relatively large and sparse when solving such problems. Thus, recently (Zitney *et al.*, 1996) we have considered the multifrontal approach as an alternative to the frontal method. Like the frontal method, it also exploits 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 (Version 1.1). Our experiments (Zitney *et al.*, 1996) have shown that the multifrontal algorithm used by UMFPACK V1.1 does not perform as well as the frontal algorithm (FAMP) on many equilibrium-stage separation problems, primarily because it is not able to take advantage of the problem structure. Thus we have developed simple new multifrontal algorithms (Mallya and Stadtherr, 1995) designed to take advantage of the good initial problem structure typically found in simulating processes that primarily involve equilibrium-stage operations. In the next section, we first describe the basic ideas involved in the multifrontal approach, and then detail the new algorithms that have been developed.

3 Multifrontal Approach

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 $Ux = y$.

In an unsymmetric-pattern multifrontal method (Davis and Duff, 1993,1996), 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

$$\begin{array}{c|c|c|}
 & C_k' & C_k'' \\
R_k' & F_k & B_k \\ \hline
R_k'' & T_k & D_k
\end{array}$$

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

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

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 matrix, and so permits the use of efficient dense matrix vector operations during the factorization of F_k and the update of D_k.

In the general-purpose approach of Davis and Duff (1993,1996), 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 equilibrium-stage processes, 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 algorithms presented below are designed to avoid this difficulty.

4 The MFA1P Algorithm

In this and the subsequent section, we discuss the new multifrontal solvers, MFAXP (**M**ulti**F**rontal **A**lgorithm, **X** **P**ivots), designed to take advantage of good initial structure in equilibrium-stage process simulation matrices. These solvers use a modified threshold pivot search strategy that attempts to maintain the structure during the factorization process. The pivot block F_k is of size 1×1 for MFA1P and 2×2 for MFA2P. The basic MFA1P algorithm is outlined below, followed by a more detailed discussion of its steps and a simple illustrative example.

Algorithm MFA1P:

For ($k = 1$; $k \leq n$; $k = k + 1$)

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.

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.

Most of the computational work in this algorithm is the outer product update in Step 4. For purposes of this computation the contribution block D'_k is treated as dense (though this may not necessarily be true). Treating the contribution block as dense is attractive from the standpoint of vectorization because it means that we can use full-matrix code without indirect addressing to perform the update. Conventional sparse linear equation solvers use an inner product formulation that requires indirect addressing; such codes typically do not vectorize efficiently on vector supercomputers. MFA1P uses the Cray Assembly Language (CAL) coded BLAS2 routine *RNK1S* for performing the outer product update. The *RNK1S* routine is highly optimized and vectorized for Cray Research supercomputers and outperforms the more general BLAS2 routine *SGER* for performing RANK-1 updates.

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 pivot the element closest to and preferable 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.

In Steps 1 and 3, portions of the L and U factors are determined and must be stored, and in Step 4 a contribution block is computed and must be stored. The method of storage to be used will depend on the amount of central (core) memory available and the disk capabilities of the computer, as well as on the size of the problem. Though the frontal and multifrontal approaches were originally developed as strategies for performing out-of-core factorizations, today it is preferable and often possible, due to the large central memories (rapid access) available on supercomputers, to use core memory for the necessary storage, as opposed to auxiliary storage (slower access). A linear solver that functions entirely in central memory requires considerably less solution time than a code that uses auxiliary storage. However, when working with very large process simulation problems, it may still be necessary to use auxiliary storage even on supercomputer systems with very large central memories. The test problems used here are not particularly large by current standards, though they are comparable in size to test problems used by others. Thus, in our experiments the use of auxiliary storage was unnecessary.

To illustrate the basic ideas of the algorithm we use the matrix shown in Figure 1. For $k = 1$, in Step 1 we assemble column 1 into the first frontal matrix. In Step 2 we first consider the diagonal element (1,1) as a possible pivot. Say that in this case it does *not* meet the threshold tolerance criterion; thus we would next consider element (2,1) as a possible pivot. Say that this element does meet the threshold tolerance. Thus, in Step 3, row 2 is assembled into the first frontal matrix and the nonzero elements in the first frontal matrix are as shown in Figure 2. In step 4, a dense matrix kernel is used to perform an outer product update to compute the contribution block for the first frontal matrix. At the end of the first pass ($k = 1$) through the algorithm, the situation is as indicated in Figure 3. This shows that we have computed a row of U and a column of L, as well as a contribution block that is saved for use with future frontal matrices. Now, for $k = 2$, in Step 1, we assemble column 2 into a new frontal matrix. This requires assembling contributions from the original matrix in positions (1,2) and (4,2), as well as contributions in positions (1,2) and

(3,2) from the contribution block of the first frontal matrix. In Step 2, we first consider element (1,2) as a possible pivot since it is closest to the diagonal. Say that this element does meet the threshold tolerance and thus is chosen as the pivot element. Thus, in Step 3, row 1 is assembled into the frontal matrix, including the contribution in position (1,4) from the first frontal matrix. The initial nonzero elements in the second frontal matrix are as shown in Figure 4, and Figure 5 shows the result of the second pass through the algorithm. New frontal matrices now continue to be assembled and outer product updates performed in them until the LU factorization is complete.

5 The MFA2P Algorithm

In this algorithm, the k-th and $(k+1)$-th column are assembled into the same frontal matrix and a 2×2 pivot block is used. The innermost loop consists of a rank-2 outer product update which is done using BLAS3 kernels as opposed to BLAS2 in MFA1P. The hope is that by using BLAS3 rather than BLAS2 in the innermost loop that the overall efficiency is improved. The basic algorithm is outlined below.

Algorithm MFA2P:

For $(k = 1;\ k \leq n;\ k = k + 2)$

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

2. In the k-th column, choose as pivot the element closest to (preferably on) the diagonal that satisfies the threshold tolerance criterion. This determines the first pivot row.

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

4. Perform a rank-one outer product update of the $(k+1)$-th column using the k-th column and the normalized first pivot row. Store as a column of L.

5. In the $(k+1)$-th column, choose as pivot the element closest to (preferably on) the diagonal that satisfies the threshold tolerance criterion. This determines the second pivot row.

6. Assemble all contributions to the second pivot row and normalize it. Store as a row of U.

7. Perform a rank-2 outer product update of D_k using the pivot columns and normalized pivot rows to compute the contribution block D'_k for this frontal matrix. Store this contribution block for later use.

Note that the second pivot is not chosen until its column has been updated using the first pivot.

6 Results and Discussion

In this section, we present the results for the performance of the MFA1P and MFA2P solvers on four sets of process simulation problems. More information about each problem set is given below. Most of the test matrices used are available from the authors. We compare the performance of MFA1P and MFA2P with that of:

1. The frontal solver FAMP.

2. Version 2.0 of UMFPACK. This version (Davis and Duff, 1995) incorporates features of the frontal method into the general-purpose, unsymmetric-pattern multifrontal solver in order to improve overall efficiency. This code is now incorporated into the Harwell Subroutine Library as MA38.

3. The conventional solver MA28. This code was used to provide a familiar and widely available benchmark. The proprietary code MA48, which has now superceded MA28 in the Harwell Subroutine Library, should perform better than MA28 on these problems, at least for a complete analyze, plus factorize, plus solve execution path. However, the experiments of Davis and Duff (1995) on a CRAY C90 suggest that in the context of vector processing, MA38 will be the most competitive Harwell code on these problems. The results of some of these experiments are included in the discussion below.

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).

In the tables of results presented below, 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). For each solver, run times (in cpu seconds) are obtained for three execution paths, namely:

1. Analyze + Factor + Solve (A+F+S): This represents 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 is typically used at each iteration in a steady-state simulation.

2. Factor only (F): This gives the time for computing the L and U factors of A given a previously determined and saved pivot sequence for a matrix of the same structure. This execution path might be used in some iterations of a simulation, though care is needed to prevent loss of

numerical stability due to changes in the relative magnitudes of the pivot elements.

3. Solve only (S): This represents the time to solve $Ax = b$ by performing forward and backward substitution using previously computed L and U factors of A. This execution path is typically suitable for several sequential time steps in a dynamic simulation, with an A+F+S or F only execution path used as needed.

Timing results are listed for MFA1P, MFA2P, FAMP, MA38, and MA28. In the tables of results, the fastest run time for each problem is shown in bold. A threshold tolerance of $t = 0.1$ was used in all codes except FAMP (which uses partial pivoting) in order to maintain numerical stability, which was monitored using the 2-norm of the residual $b - Ax$.

For each problem set a figure is shown summarizing 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.

No *a priori* reordering was done on any of the test matrices. The performance of the MFAXP algorithms depends on both the row and column orderings. These algorithms are designed to take advantage of the already good row and column orderings present in typical equilibrium-stage process simulation problems. The performance of the frontal algorithm (FAMP) depends on the row ordering. The natural structure of an equilibrium-stage process simulation problem generally provides a good row ordering in this regard. MA28 and MA38 both use Markowitz-type pivot strategies to maintain sparsity, in effect determining a row and column ordering.

6.1 Problem Set 1

These 14 problems involve the steady-state simulation of equilibrium-stage processes 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. Matrix statistics for these problems are shown in Table 1. Though highly asymmetric structurally, these matrices are nearly banded, but with some off-band elements. This is due to the grouping of equations by equilibrium stage. Tables 1, 2, and 3 summarize the run time results for the A+F+S, F only and S only execution paths, respectively. Figure 6 shows the relative performance of the methods for each execution path based on their ANRT, as defined above. The A+F+S execution path is used in each iteration in this application code, so the results for this path are the most significant.

The new MFAXP solvers are the best on all problems for the A+F+S execution path, and on the average are nearly twice as fast as the frontal solver FAMP. Neither MA38 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, as discussed above, that MA48, the successor to MA28 in the Harwell Subroutine Library, should perform better than MA28 on these problems, at least on the A+F+S execution path. However, on the *rdist1* problem, Davis and Duff (1995) found that, on a CRAY C90, MA38 was about six times faster than MA48 on the A+F+S execution path, and nearly four times as fast on the F only path. The use of BLAS3 in the innermost loop of MFA2P, as opposed to BLAS2 in the innermost loop of MFA1P, does not appear to offer any advantage, and on many problems it is slower than MFA1P. We attribute this to the fact that MFA1P uses a highly optimized assembly language BLAS2 routine, while no comparably optimized BLAS3 routine was available.

6.2 Problem Set 2

These six problems involve the steady-state simulation of equilibrium-stage processes using SEPARATE, a simulation package developed at University of Illinois for steady-state simulation of single and interlinked distillation columns (O'Neill *et al.*, 1994). Matrix statistics for these problems are shown in Table 4. This package uses a Naphtali-Sandholm (1971) formulation in which the equations are grouped by equilibrium stage. The linear equation system that must be solved has a block-tridiagonal or almost-block-tridiagonal form, with off-tridiagonal blocks corresponding to interlinking streams between columns or pumparounds within a single column. Tables 4, 5, and 6 summarize the run time results for the A+F+S, F only and S only execution paths, respectively. Figure 7 shows the relative performance of the methods for each execution path based on their ANRT. The A+F+S execution path is used in each iteration in this application code, so the results for this path are the most significant.

The new MFA1P solver is the best on all these problems for the A+F+S execution path, about twice as fast as the frontal solver FAMP on the larger problems. Again, neither MA38 nor MA28 are able to take good advantage of the structure of these problems. It should be noted that the performance of the frontal solver might be significantly improved here, and in Problem Set 1, if some attempt had been made to improve the row ordering, for instance by using the heuristic approach of Camarda and Stadtherr (1995). However, applying such reordering techniques can be relatively expensive computationally.

6.3 Problem Set 3

These six problems involve the dynamic simulation of equilibrium-stage processes using DYNDIST. Matrix statistics for these problems are shown in Table 7. DYNDIST is a prototype code from Aspen Technology, Inc. for dynamic simulation of multicomponent distillation columns. This dynamic simulator uses a global time step and solves an entire DAE system simultaneously. Tables 7, 8, and 9 summarize the run time results for the A+F+S, F only and S only execution paths, respectively. Figure 8 shows the relative performance of the methods for each execution path based on their ANRT.

For all these problems, on all execution paths, the frontal solver FAMP is the best, with the new multifrontal solver MFA1P close behind. Given a very good row ordering, as in this case, the frontal solver can be extremely efficient. Because it employs only a single frontal matrix, it does not have the overhead expense required to keep track of the many frontal matrices in the multifrontal approach.

6.4 Problem Set 4

Finally, we consider a problem set not derived from equilibrium-stage process simulation, but from more general process simulation problems. This will demonstrate the performance of the MFAXP codes in the absence of the good initial row and column ordering present in equilibrium-stage simulation problems. These seven problems involve steady-state simulation problems solved using SEQUEL-II, an equation-based simulation program developed as a prototype at the University of Illinois (Stadtherr and Hilton, 1982; Zitney and Stadtherr, 1988). Matrix statistics for these problems are shown in Table 10. Tables 10, 11, and 12 summarize the run time results for the A+F+S, F only and S only execution paths, respectively. Figure 9 shows the relative performance of the methods for each execution path based on their ANRT. It is also interesting to note that, for the *lhr_4k* problem, Davis and Duff (1995) have found that, on a CRAY C90, MA38 is over twice as fast as MA48 on both the A+F+S and F only execution paths.

As expected, on these problems, especially the larger ones, the MFAXP codes do not perform well, and on the largest problem they are not even competitive due to extremely large computational requirements. The MFAXP solvers require both a good row and column ordering, and while this is a feature of equilibrium-stage process simulation problems, it is not a feature of process simulation problems in general.

7 Concluding Remarks

We have demonstrated here how a simple multifrontal approach 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 the general-purpose multifrontal approach as implemented in MA38. On most problems, including those derived from ASPEN PLUS simulations, the new multifrontal approach also provides significant improvements over the frontal method.

Acknowledgements – This work has been supported by the National Science Foundation under Grant 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 Mark Parent for his help in the development of some of the problems in Problem Set 1, and Steve Zitney (Aspen Technology, Inc.) for providing the DYNDIST problems.

References

Camarda, K. V.; Stadtherr, M. A. Frontal solvers for process simulation: Local row ordering strategies. Presented at AIChE 1995 Annual Meeting, Miami, Florida, 1995.

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

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

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

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

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

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

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

Mallya, J.; Stadtherr, M. A. A new multifrontal solver for process simulation on parallel/vector supercomputers. Presented at AIChE 1995 Annual Meeting, Paper 168h, Miami, Florida, 1995.

Napthali, L. M.; Sandholm, D. P. Multicomponent separation calculations by linearization. *AIChE J.* **1971**, *17*, 148.

O'Neill, A. J.; Kaiser, D. J.; Stadtherr, M. A. Strategies for equilibrium-stage separation calculations on parallel computers. *AIChE J.* **1994**, *40*, 65–72.

Stadtherr, M. A.; Hilton, C. M. Development of a new equation-based process flowsheeting system: numerical studies. In *Selected Topics on Computer-Aided Process Design and Analysis*; R.S.H. Mah and G.V. Reklaitis, Eds.; *AIChE Symposium Series* **1982**, *78*(214), 12–28.

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

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

Zitney, S. E. A frontal code for ASPEN PLUS on advanced architecture computers. Presented at AIChE 1990 Annual Meeting, Chicago, 1990.

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

Zitney, S. E.; Brüll, L.; Lang, L.; Zeller, R., Plantwide dynamic simulation on supercomputers: modeling a Bayer distillation process. *AIChE Symposium Series* **1995**, *91*(304), 313–316.

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

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

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

Zitney, S. E.; Stadtherr, M. A. Supercomputing strategies for the design and analysis of complex separation systems. *Ind. Eng. Chem. Res.* **1993b**, *32*, 604–612.

Figure Captions

Figure 1. Example occurrence matrix. See text for discussion.

Figure 2. First frontal matrix in example problem.

Figure 3. First frontal matrix after elimination of variable 1. A u indicates an element of the U factor and an ℓ an element of the L factor.

Figure 4. Second frontal matrix in example problem.

Figure 5. Second frontal matrix after elimination of variable 2. A u indicates an element of the U factor and an ℓ an element of the L factor.

Figure 6. Average normalized run times for ASPEN PLUS matrices (Problem Set 1).

Figure 7. Average normalized run times for SEPARATE matrices (Problem Set 2)

Figure 8. Average normalized run times for DYNDIST matrices (Problem Set 3).

Figure 9. Average normalized run times for SEQUEL-II matrices (Problem Set 4). EX indicates an excessive computational requirement, and NS when an execution path is not solved as a result.

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

Figure 1: Example occurrence matrix. See text for discussion.

$$\begin{array}{c|c|cc|}
 & 1 & 2 & 4 \\ \hline
2 & \times & \times & \times \\ \hline
1 & \times & & \\
3 & \times & & \\ \hline
\end{array}$$

Figure 2: First frontal matrix in example problem.

$$\begin{array}{c|c|cc|}
\multicolumn{1}{c}{} & \multicolumn{1}{c}{1} & 2 & \multicolumn{1}{c}{4} \\ \cline{2-4}
2 & u & u & u \\ \cline{2-4}
1 & \ell & \times & \times \\
3 & \ell & \times & \times \\ \cline{2-4}
\end{array}$$

Figure 3: First frontal matrix after elimination of variable 1. A u indicates an element of the U factor and an ℓ an element of the L factor.

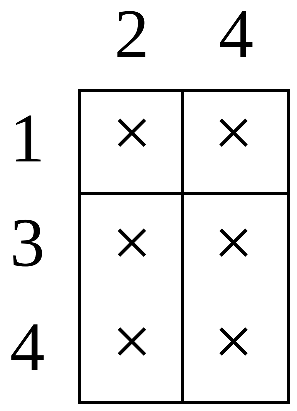

Figure 4: Second frontal matrix in example problem.

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

Figure 5: Second frontal matrix after elimination of variable 2. A u indicates an element of the U factor and an ℓ an element of the L factor.

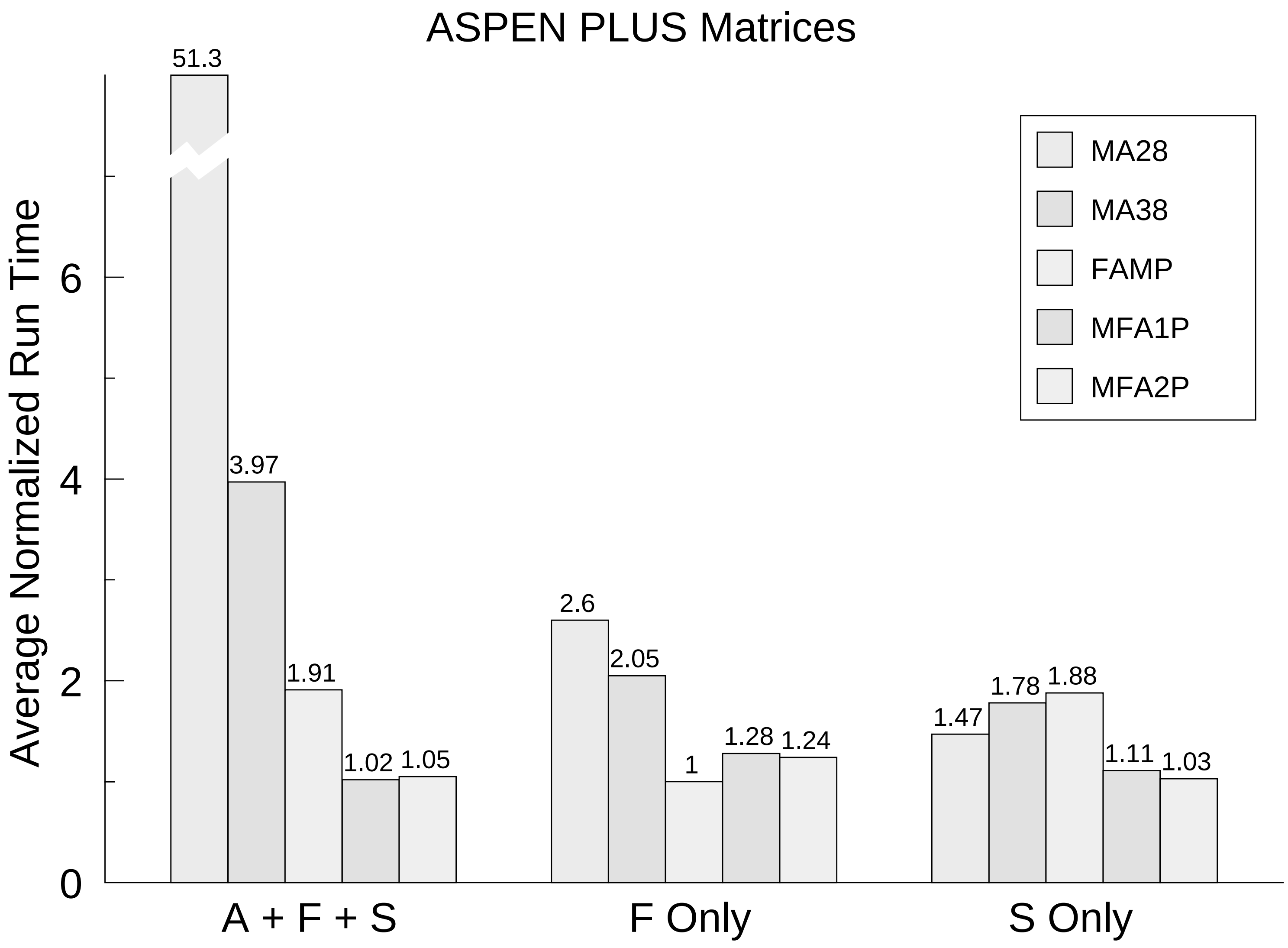

Figure 6: Average normalized run times for ASPEN PLUS matrices (Problem Set 1).

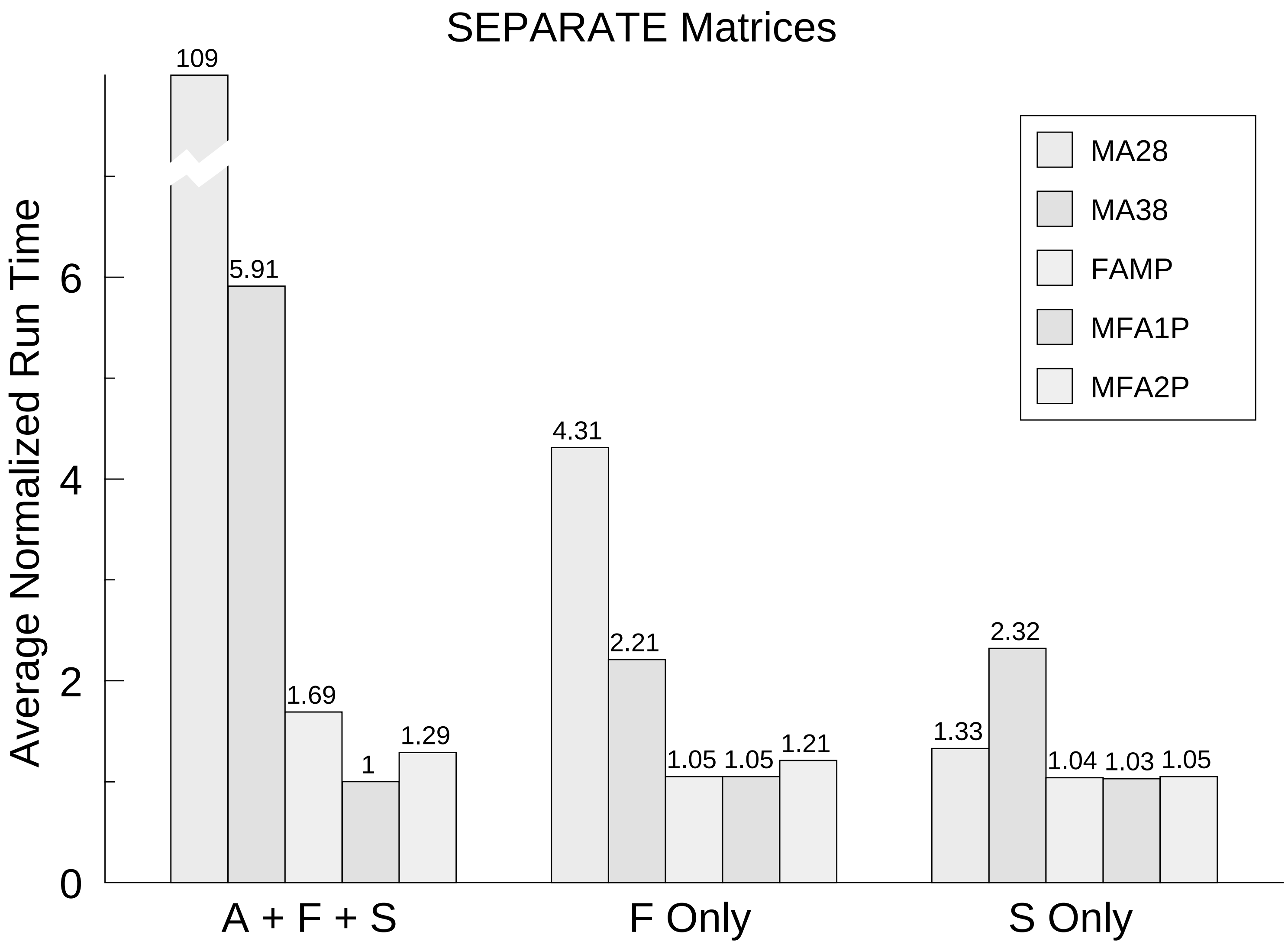

Figure 7: Average normalized run times for SEPARATE matrices (Problem Set 2).

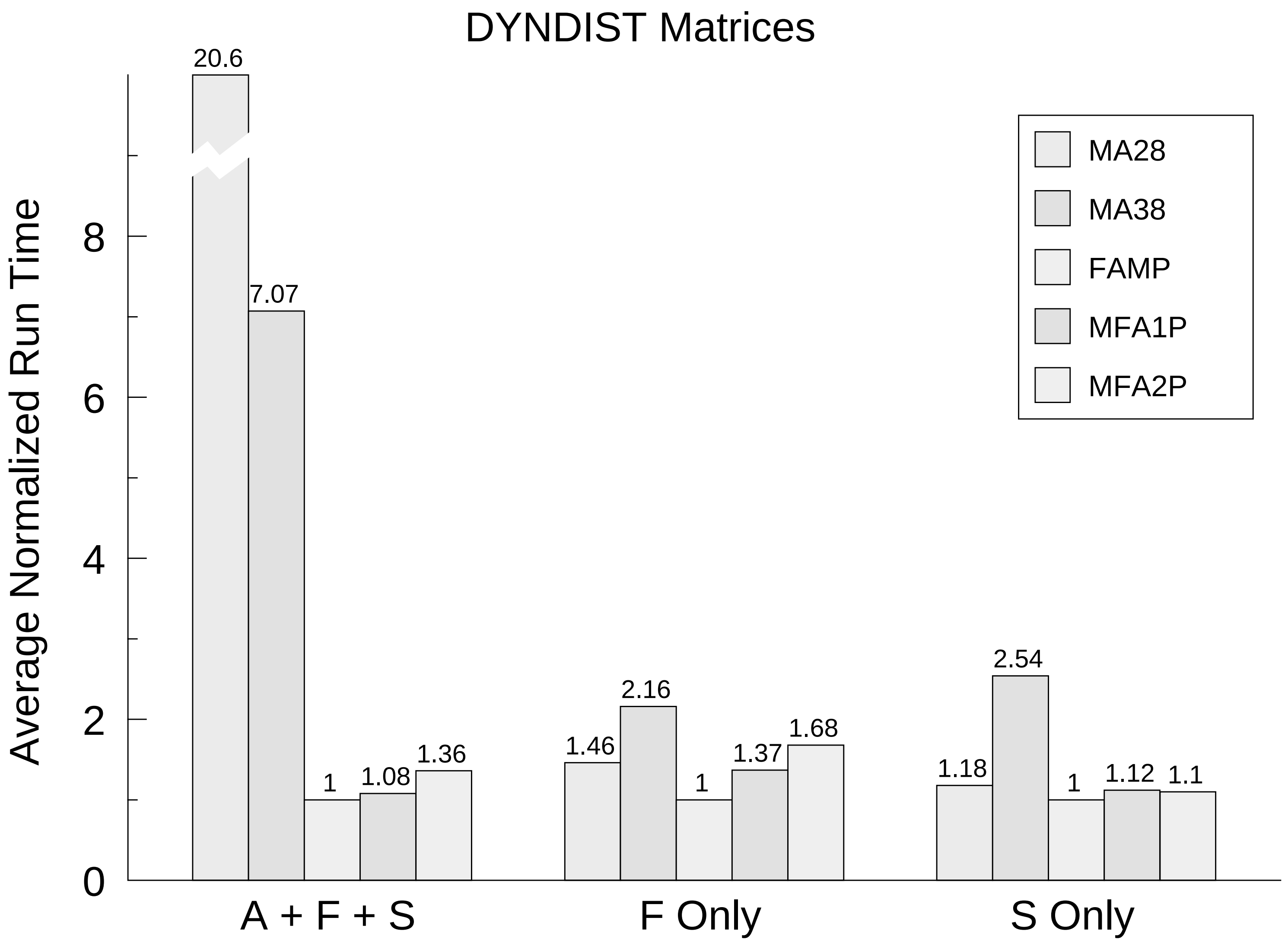

Figure 8: Average normalized run times for DYNDIST matrices (Problem Set 3).

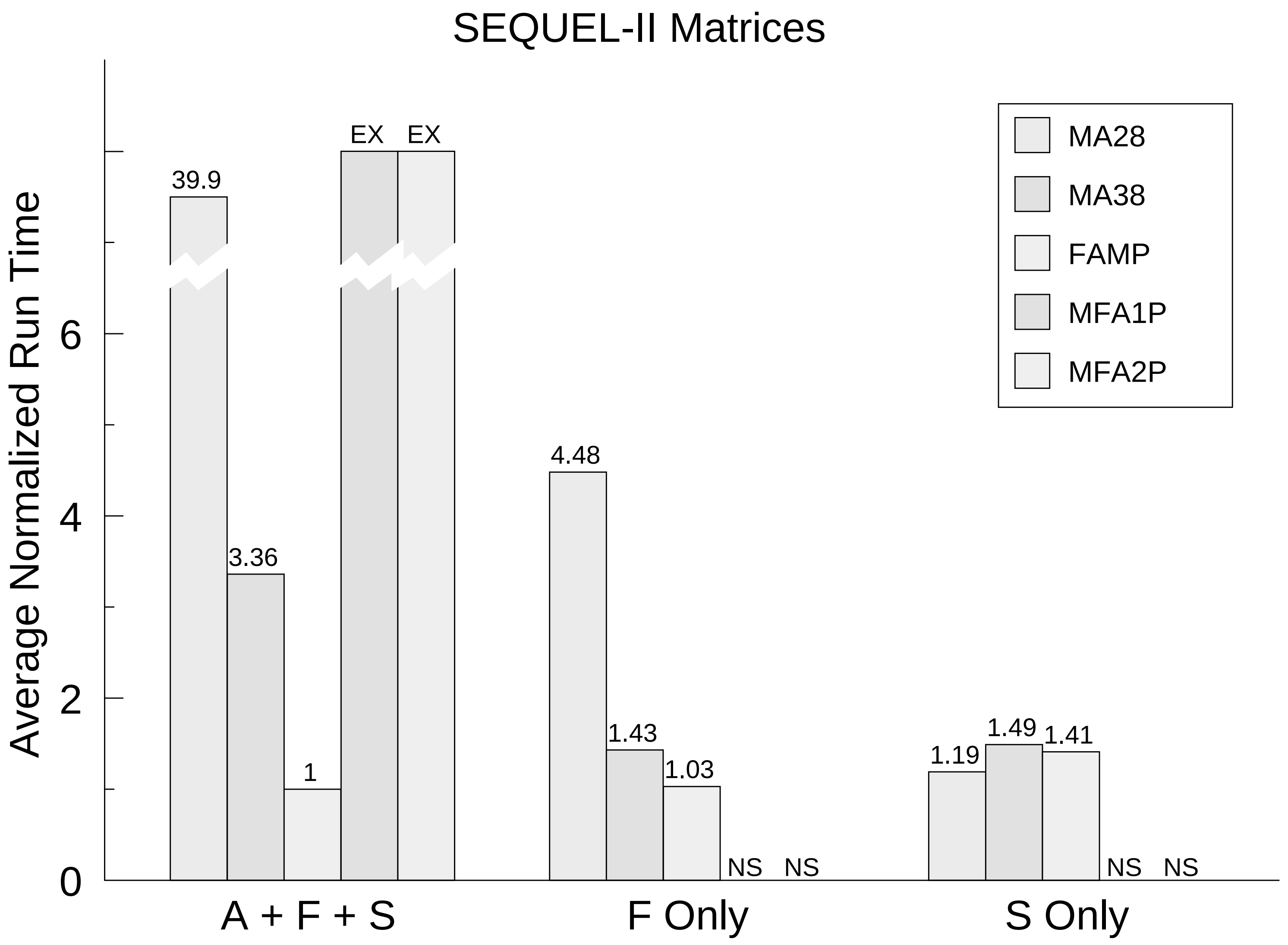

Figure 9: Average normalized run times for SEQUEL-II matrices (Problem Set 4). EX indicates an excessive computational requirement, and NS when an execution path is not solved as a result.

Table 1: A+F+S Run times for ASPEN PLUS test problems.

Name	**N**	**NZ**	**as**	**FAMP**	**MA38**	**MA28**	**MFA1P**	**MFA2P**
v3	1078	16937	0.91	0.114	0.243	2.159	$6.01\text{x}10^{-2}$	**$5.72\text{x}10^{-2}$**
v10	1148	15729	0.94	0.109	0.227	1.862	$6.10\text{x}10^{-2}$	**$5.79\text{x}10^{-2}$**
v13	834	9713	0.95	$6.35\text{x}10^{-2}$	0.140	0.983	$4.20\text{x}10^{-2}$	**$4.01\text{x}10^{-2}$**
mpex2	848	11413	0.96	$6.60\text{x}10^{-2}$	0.176	0.299	$4.27\text{x}10^{-2}$	**$4.21\text{x}10^{-2}$**
mpex3	2473	46503	0.94	0.359	0.567	10.598	0.173	**0.166**
mpex4	2478	44075	0.95	0.317	0.559	9.19	0.172	**0.164**
mpmult1	2023	31894	0.95	0.234	0.472	6.131	0.13	**0.117**
rdist1	4134	94408	0.94	0.730	1.854	30.21	**0.32**	0.378
rdist2	3198	56934	0.95	0.392	0.696	16.11	**0.22**	0.26
rdist3	2398	61896	0.85	0.478	1.172	32.87	**0.20**	0.228
sumb	523	4998	0.95	$3.22\text{x}10^{-2}$	$8.30\text{x}10^{-2}$	0.314	**$2.18\text{x}10^{-2}$**	$2.26\text{x}10^{-2}$
traycalc	1145	20296	0.88	0.14	0.244	2.649	$6.81\text{x}10^{-2}$	**$7.1\text{x}10^{-2}$**
uosb	523	4998	0.95	$3.22\text{x}10^{-2}$	$8.29\text{x}10^{-2}$	0.315	**$2.19\text{x}10^{-2}$**	$2.20\text{x}10^{-2}$
userupp	1269	22508	0.89	0.154	0.289	3.310	**$7.39\text{x}10^{-2}$**	$7.9\text{x}10^{-2}$

Table 2: Factor only timings for ASPEN PLUS test problems.

Name	**N**	**FAMP**	**MA38**	**MA28**	**MFA1P**	**MFA2P**
v3	1078	$\mathbf{4.85x10^{-2}}$	$9.52x10^{-2}$	0.123	$5.9x10^{-2}$	$5.4x10^{-2}$
v10	1148	$\mathbf{4.40x10^{-2}}$	$9.03x10^{-2}$	0.110	$5.98x10^{-2}$	$5.50x10^{-2}$
v13	834	$\mathbf{3.04x10^{-2}}$	$5.74x10^{-2}$	$6.42x10^{-2}$	$4.10x10^{-2}$	$3.89x10^{-2}$
mpex2	848	$\mathbf{3.23x10^{-2}}$	$6.55x10^{-2}$	$5.43x10^{-2}$	$4.18x10^{-2}$	$4.01x10^{-2}$
mpex3	2473	**0.124**	0.247	0.382	0.169	0.141
mpex4	2478	**0.114**	0.244	0.343	0.170	0.140
mpmult1	2023	$\mathbf{8.5x10^{-2}}$	0.182	0.24	0.128	0.10
rdist1	4134	**0.248**	0.595	0.903	0.28	0.327
rdist2	3198	**0.15**	0.311	0.548	0.189	0.224
rdist3	2398	**0.156**	0.394	1.17	0.175	0.190
sumb	523	$\mathbf{1.71x10^{-2}}$	$3.18x10^{-2}$	$2.9x10^{-2}$	$2.13x10^{-2}$	$2.04x10^{-2}$
traycalc	1145	$\mathbf{5.29x10^{-2}}$	0.105	0.144	$6.66x10^{-2}$	$6.1x10^{-2}$
uosb	523	$\mathbf{1.71x10^{-2}}$	$3.19x10^{-2}$	$2.91x10^{-2}$	$2.13x10^{-2}$	$2.0x10^{-2}$
userupp	1269	$\mathbf{5.83x10^{-2}}$	0.116	0.171	$7.27x10^{-2}$	$6.9x10^{-2}$

Table 3: Solve only timings for ASPEN PLUS problems.

Name	**N**	**FAMP**	**MA38**	**MA28**	**MFA1P**	**MFA2P**
v3	1078	$5.05\text{x}10^{-3}$	$4.38\text{x}10^{-3}$	$3.45\text{x}10^{-3}$	$2.76\text{x}10^{-3}$	$\mathbf{2.28x10^{-3}}$
v10	1148	$5.35\text{x}10^{-3}$	$4.93\text{x}10^{-3}$	$3.58\text{x}10^{-3}$	$2.86\text{x}10^{-3}$	$\mathbf{2.39x10^{-3}}$
v13	834	$3.08\text{x}10^{-3}$	$3.83\text{x}10^{-3}$	$2.54\text{x}10^{-3}$	$2.02\text{x}10^{-3}$	$\mathbf{1.71x10^{-3}}$
mpex2	848	$3.61\text{x}10^{-3}$	$3.60\text{x}10^{-3}$	$2.53\text{x}10^{-3}$	$2.03\text{x}10^{-3}$	$\mathbf{1.74x10^{-3}}$
mpex3	2473	$1.34\text{x}10^{-2}$	$8.68\text{x}10^{-3}$	$8.3\text{x}10^{-3}$	$7.01\text{x}10^{-3}$	$\mathbf{6.29x10^{-3}}$
mpex4	2478	$1.21\text{x}10^{-2}$	$8.99\text{x}10^{-3}$	$8.27\text{x}10^{-3}$	$6.68\text{x}10^{-3}$	$\mathbf{6.25x10^{-3}}$
mpmult1	2023	$1.04\text{x}10^{-2}$	$8.31\text{x}10^{-3}$	$6.57\text{x}10^{-3}$	$5.21\text{x } 10^{-3}$	$\mathbf{5.1x10^{-3}}$
rdist1	4134	$1.46\text{x}10^{-2}$	$1.982\text{x}10^{-2}$	$1.46\text{x}10^{-2}$	$1.27\text{x}10^{-2}$	$\mathbf{1.16x10^{-2}}$
rdist2	3198	$1.60\text{x}10^{-2}$	$1.24\text{x}10^{-2}$	$1.10\text{x}10^{-2}$	$9.05\text{x}10^{-3}$	$\mathbf{8.05x10^{-3}}$
rdist3	2398	$9.3\text{x}10^{-3}$	$1.17\text{x}10^{-2}$	$9.94\text{x}10^{-3}$	$7.86\text{x}10^{-3}$	$\mathbf{6.1x10^{-3}}$
sumb	523	$2.20\text{x}10^{-3}$	$2.61\text{x}10^{-3}$	$1.52\text{x}10^{-3}$	$\mathbf{1.20x10^{-3}}$	$1.23\text{x}10^{-3}$
traycalc	1145	$5.62\text{x}10^{-3}$	$4.01\text{x}10^{-3}$	$3.79\text{x}10^{-3}$	$\mathbf{3.15x10^{-3}}$	$3.52\text{x}10^{-3}$
uosb	523	$2.20\text{x}10^{-3}$	$2.60\text{x}10^{-3}$	$1.53\text{x}10^{-3}$	$\mathbf{1.20x10^{-3}}$	$1.33\text{x}10^{-3}$
userupp	1269	$6.20\text{x}10^{-3}$	$4.39\text{x}10^{-3}$	$4.16\text{x}10^{-3}$	$\mathbf{3.35x10^{-3}}$	$3.75\text{x}10^{-3}$

Table 4: A+F+S timings for SEPARATE problems.

Name	**N**	**NZ**	**as**	**FAMP**	**MA38**	**MA28**	**MFA1P**	**MFA2P**
sep56_4	504	4006	0.52	2.42x10^{-2}	8.29x10^{-2}	0.40	**2.11x$\mathbf{10^{-2}}$**	2.73x10^{-2}
sep56_10	1176	16669	0.52	9.23x10^{-2}	0.34	4.35	**5.44x$\mathbf{10^{-2}}$**	6.94x10^{-2}
sep56_15	1736	33665	0.52	0.203	0.63	14.63	**9.18x$\mathbf{10^{-2}}$**	0.117
sep100_4	900	7203	0.53	4.18x10^{-2}	0.17	1.22	**3.53x$\mathbf{10^{-2}}$**	4.62x10^{-2}
sep100_10	2100	29902	0.53	0.163	0.61	13.04	**9.40x$\mathbf{10^{-2}}$**	0.123
sep100_15	3100	60333	0.53	0.35	1.14	36.11	**0.161**	0.21

Table 5: Factor only timings for SEPARATE problems.

Name.	**N**	**FAMP**	**MA38**	**MA28**	**MFA1P**	**MFA2P**
sep56_4	504	**$1.46\text{x}10^{-2}$**	$2.67\text{x}10^{-2}$	$2.63\text{x}10^{-2}$	$1.80\text{x}10^{-2}$	$2.12\text{x}10^{-2}$
sep56_10	1176	$4.65\text{x}10^{-2}$	0.102	0.16	**$4.60\text{x}10^{-2}$**	$5.06\text{x}10^{-2}$
sep56_15	1736	$8.54\text{x}10^{-2}$	0.19	0.534	**$7.70\text{x}10^{-2}$**	$8.07\text{x}10^{-2}$
sep100_4	900	**$2.63\text{x}10^{-2}$**	$5.02\text{x}10^{-2}$	$5.04\text{x}10^{-2}$	$2.84\text{x}10^{-2}$	$3.54\text{x}10^{-2}$
sep100_10	2100	$8.23\text{x}10^{-2}$	0.183	0.336	**$8.00\text{x}10^{-2}$**	$8.75\text{x}10^{-2}$
sep100_15	3100	0.153	0.346	0.99	**0.13**	0.16

Table 6: Solve only timings for SEPARATE problems.

Name	**N**	**FAMP**	**MA38**	**MA28**	**MFA1P**	**MFA2P**
sep56_4	504	**$1.25\text{x}10^{-3}$**	$3.42\text{x}10^{-3}$	$1.49\text{x}10^{-3}$	$1.41\text{x}10^{-3}$	$1.44\text{x}10^{-3}$
sep56_10	1176	$3.08\text{x}10^{-3}$	$6.86\text{x}10^{-3}$	$3.9\text{x}10^{-3}$	**$3.02\text{x}10^{-3}$**	$3.11\text{x}10^{-3}$
sep56_15	1736	$4.85\text{x}10^{-3}$	$8.15\text{x}10^{-3}$	$6.46\text{x}10^{-3}$	**$4.60\text{x}10^{-3}$**	$4.74\text{x}10^{-3}$
sep100_4	900	**$2.17\text{x}10^{-3}$**	$6.29\text{x}10^{-3}$	$2.69\text{x}10^{-3}$	$2.28\text{x}10^{-3}$	$2.33\text{x}10^{-3}$
sep100_10	2100	$5.48\text{x}10^{-3}$	$1.27\text{x}10^{-2}$	$7.16\text{x}10^{-3}$	**$5.20\text{x}10^{-3}$**	$5.28\text{x}10^{-3}$
sep100_15	3100	$8.70\text{x}10^{-3}$	$1.50\text{x}10^{-2}$	$1.17\text{x}10^{-2}$	**$8.0\text{x}10^{-3}$**	$8.26\text{x}10^{-3}$

Table 7: A+F+S timings for DYNDIST problems.

Name	**N**	**NZ**	**as**	**FAMP**	**MA38**	**MA28**	**MFA1P**	**MFA2P**
dyndist20_10	1465	8527	0.976	**5.86x10^{-2}**	0.454	1.26	6.61x10^{-2}	8.39x10^{-2}
dyndist30_10	2125	12457	0.977	**8.48x10^{-2}**	0.752	2.37	9.55x10^{-2}	0.113
dyndist40_10	2785	16387	0.978	**0.112**	1.11	4.16	0.114	0.151
dyndist20_5	925	4987	0.977	**3.50x10^{-2}**	0.180	0.412	3.80x10^{-2}	4.89x10^{-2}
dyndist40_5	1745	9547	0.979	**6.45x10^{-2}**	0.346	1.209	6.90x10^{-2}	8.96x10^{-2}
dyndist50_5	2155	11827	0.980	**7.85x10^{-2}**	0.428	1.776	8.50x10^{-2}	0.103

Table 8: Factor only timings for DYNDIST problems.

Name.	**N**	**FAMP**	**MA38**	**MA28**	**MFA1P**	**MFA2P**
dyndist20_10	1465	**3.86x10^{-2}**	8.71x10^{-2}	6.18x10^{-2}	5.85x10^{-2}	6.74x10^{-2}
dyndist30_10	2125	**5.62x10^{-2}**	0.128	9.14x10^{-2}	8.10x10^{-2}	9.73x10^{-2}
dyndist40_10	2785	**7.44x10^{-2}**	0.168	0.132	0.09	0.125
dyndist20_5	925	**2.34x10^{-2}**	4.79x10^{-2}	2.93x10^{-2}	3.23x10^{-2}	4.01x10^{-2}
dyndist40_5	1745	**4.40x10^{-2}**	9.09x10^{-2}	5.63x10^{-2}	6.01x10^{-2}	7.09x10^{-2}
dyndist50_5	2155	**5.43x10^{-2}**	0.112	6.99x10^{-2}	7.24x10^{-2}	8.79x10^{-2}

Table 9: Solve only timings for DYNDIST problems.

Name	**N**	**FAMP**	**MA38**	**MA28**	**MFA1P**	**MFA2P**
dyndist20_10	1465	$\mathbf{3.49x10^{-3}}$	$8.55x10^{-3}$	$4.10x10^{-3}$	$4.10x10^{-3}$	$3.96x10^{-3}$
dyndist30_10	2125	$\mathbf{5.03x10^{-3}}$	$1.21x10^{-2}$	$5.98x10^{-3}$	$5.75x10^{-3}$	$5.74x10^{-3}$
dyndist40_10	2785	$\mathbf{6.66x10^{-3}}$	$1.57x10^{-2}$	$7.89x10^{-3}$	$7.64x10^{-3}$	$7.55x10^{-3}$
dyndist20_5	925	$\mathbf{2.16x10^{-3}}$	$5.90x10^{-3}$	$2.52x10^{-3}$	$2.41x10^{-3}$	$2.39x10^{-3}$
dyndist40_5	1745	$\mathbf{4.04x10^{-3}}$	$1.09x10^{-2}$	$4.77x10^{-3}$	$4.32x10^{-3}$	$4.28x10^{-3}$
dyndist50_5	2155	$\mathbf{5.0x10^{-3}}$	$1.34x10^{-2}$	$5.88x10^{-3}$	$5.34x10^{-3}$	$5.06x10^{-3}$

Table 10: A+F+S run times for SEQUEL-II problems. EX indicates an excessive computational requirement.

Name	**N**	**NZ**	**as**	**FAMP**	**MA38**	**MA28**	**MFA1P**	**MFA2P**
ngc	1235	16868	0.97	**0.135**	0.616	2.457	0.184	0.21
lhr	1477	18592	0.99	**0.164**	0.463	2.769	0.21	0.30
cyclo1	517	2420	0.98	$\mathbf{2.17x10^{-2}}$	$6.55x10^{-2}$	0.169	$3.31x10^{-2}$	$5.24x10^{-2}$
beef	1197	12070	0.99	$\mathbf{8.36x10^{-2}}$	0.298	2.24	0.144	0.24
lhr_2k	2954	37206	0.99	**0.316**	0.927	4.267	0.471	0.63
lhr_4k	4101	82682	0.98	**0.818**	3.35	52.81	5.45	5.84
lhr_17k	17576	381975	1.0	**3.96**	10.176	522.01	EX	EX

Table 11: Factor only timings for SEQUEL-II problems. NS indicates not solved due to excessive computational requirement in the A+F+S execution path.

Name	**N**	**FAMP**	**MA38**	**MA28**	**MFA1P**	**MFA2P**
ngc	1235	0.132	**0.120**	0.138	0.178	0.19
lhr	1477	0.143	**0.126**	0.146	0.20	0.27
cyclo1	517	**1.83x10^{-2}**	2.11x10^{-2}	1.94x10^{-2}	3.21x10^{-2}	5.12x10^{-2}
beef	1197	**5.14x10^{-2}**	9.29x10^{-2}	0.103	0.131	0.21
lhr_2k	2954	**0.169**	0.251	0.41	0.391	0.58
lhr_4k	4101	**0.347**	0.62	2.31	5.36	5.68
lhr_17k	17576	**1.63**	2.89	27.67	NS	NS

Table 12: Solve only timings for SEQUEL-II problems. NS indicates not solved due to excessive computational requirement in the A+F+S execution path.

Name	**N**	**FAMP**	**MA38**	**MA28**	**MFA1P**	**MFA2P**
ngc	1235	5.57x10^{-3}	5.83x10^{-3}	3.52x10^{-3}	**3.19x10^{-3}**	4.4x10^{-3}
lhr	1477	7.0x10^{-3}	7.37x10^{-3}	4.18x10^{-3}	**3.60x10^{-3}**	5.67x10^{-3}
cyclo1	517	1.91x10^{-3}	2.17x10^{-3}	1.29x10^{-3}	1.34x10^{-3}	**1.23x10^{-3}**
beef	1197	4.61x10^{-3}	4.62x10^{-3}	3.4x10^{-3}	**2.79x10^{-3}**	4.38x10^{-3}
lhr_2k	2954	**1.39x10**$^{-2}$	1.472x10^{-2}	1.6x10^{-2}	5.2x 10^{-2}	5.55x10^{-2}
lhr_4k	4101	**2.21x10^{-2}**	2.26x10^{-2}	2.63x10^{-2}	7.1x 10^{-2}	8.1x10^{-2}
lhr_17k	17576	**9.13x10^{-2}**	0.10	0.135	NS	NS