# Large-Scale Process Simulation and Optimization in a High Performance Computing Environment

Mark A. Stadtherr
Department of Chemical Engineering
University of Notre Dame
Notre Dame, IN 46556 USA
E-mail: markst@nd.edu

Presented at AspenWorld 97, Boston, MA, October 12-16, 1997

**Abstract**

High performance computing (HPC) technology, including parallel and/or vector processing, provides opportunities to solve process optimization and simulation problems faster and more reliably than ever before, thus enabling the solution of increasingly large scale problems, even in a real time environment. This presentation will focus on recent advances in HPC technology and methods for exploiting it in process optimization and simulation. Of particular interest are methods for the large, sparse linear equation systems that often arise in large-scale process engineering problems, and that often represent a computational bottleneck. Also of interest is an approach for guaranteeing the reliable solution of process engineering problems.

## 1 Introduction

The future success of the chemical process industries depends on the ability to design and operate complex, highly interconnected plants that are profitable and that meet quality, safety, environmental and other standards. Towards this goal, process simulation and optimization tools are increasingly being used industrially in every step of the design process and in subsequent plant operations. To perform realistic process simulation for very large scale industrial processes, however, requires adequate computational resources. Today, high performance computing (HPC) technology, including parallel and/or vector computing, provides the computational power to realistically model, simulate, design, and optimize complex chemical manufacturing processes, steady- and unsteady-state. To better use this leading edge technology in process simulation requires the use of techniques that efficiently exploit vector and parallel processing. Since most currently used techniques for solving such problems were developed for use on conventional serial machines, it is often necessary to rethink problem solving strategies in order to take full advantage of HPC power.

High performance computing typically involves some form of parallel processing, which in recent years has been rapidly entering the mainstream of computer technology. Though single processor performance will continue to improve, the most immediate way to improve a system's performance is through the use of multiple processors, as opposed to waiting for the next generation of single processors. Furthermore, physical limits on single processor speeds will eventually be reached. Since in principle there is no upper limit on the speed of a parallel processing machine, this represents the inevitable future of computing, not only for the fastest state-of-the-art supercomputers, but also for desk-based machines and servers. In fact it is the booming network and database server market that has been driving parallel processing into the mainstream today.

Parallel processing takes on many forms. The processors used may be CISC (Complex Instruction Set Computing) processors, such as the Intel Pentium II, RISC (Reduced Instruction Set Computing) processors, such as the MIPS R10000 or Sun UltraSPARC-II, or vector processors, such as used in the CRAY T90. While historically vector processors have been very expensive and produced in fairly low volume, there are some indications today that vector processing technology will be ultimately be incorporated with super-scalar RISC technology in processors produced as a higher volume commodity. Whatever processors are used in the system, they can be connected in a variety of ways, and can access memory in a number of different ways. Most systems can be classified as either shared-memory or distributed-memory, or some hybrid thereof. Both high-end parallel/vector supercomputers, such as the CRAY T90 and lower-end symmetric multiprocessors (SMPs), such as a four processor Compaq ProLiant 6000, feature uniform access to a shared memory. This arrangement may not scale well to larger numbers of processors, so other designs, such as the Cray/SGI Origin2000, feature a nonuniform access to memory that though physically distributed may be considered logically shared. Distributed-memory systems include both massively parallel machines, such as the CRAY T3E, and network-based systems. While for the former, a shared-memory programming model may still be useful in some cases, for the latter one must usually rely on message passing, using popular protocols such as PVM (Parallel Virtual Machine) or MPI (Message Passing Interface), to move data from processor to processor and to memory. With the rapid advancement of networking technology, network-based parallel systems are becoming common. Essentially any collection of machines on a network can be used as a parallel computing system. The machines in the network-based system may range from simple workstations to SMPs to parallel/vector supercomputers. Such a heterogeneous network of computational resources is sometimes referred to as a *metacomputer*. The concept of metacomputing in the context of chemical process engineering was discussed originally by Stadtherr *et al.* (1993). An excellent recent example of the implementation of metacomputing for process simulation and optimization is the Simulator Manager at Bayer AG (Brüll, 1997). This allows different units in a large problem to be considered in parallel on the most appropriate machine with the most appropriate software, with the overall problem converged by the Simulation Manager using a simultaneous-modular approach (e.g., Chen and Stadtherr, 1985; Chimowitz and Bielinis, 1987).

In Sections 2-4 below, the focus is on using vector and parallel computing for increasing the speed of process simulation and optimization computations. A key step in solving complex process engineering problems is the solution of a large, sparse system of linear equations. In fact, this step may account for as much as 90% of the computation time on industrial-scale problems. Thus, any reduction in the linear system solution time will result in a significant reduction of the total simulation time. The matrices that arise, however, do not have any of the desirable structural or numerical properties, such as numerical or structural symmetry, positive definiteness, and diagonal dominance, often associated with sparse matrices, and usually exploited in developing efficient algorithms for high performance computing. We describe here vector and parallel algorithms for the solution of linear equation systems arising in process simulation and optimization. It is important to realize that not only does HPC provide the power to increase the speed with which problems can be solved, but also the reliability with which they can be solved. Thus, in Section 5 we discuss an approach, based on interval mathematics, that is capable of guaranteeing the reliable solution of process engineering problems.

## 2 Frontal Method

Consider the solution of a linear equation system $Ax = b$, where $A$ is a large sparse $n \times n$ matrix and $x$ and $b$ are column vectors of length $n$. While iterative methods can be used to solve such systems, the reliability of such methods is questionable in the context of process simulation (Cofer and Stadtherr, 1996). Thus we concentrate here on direct methods. Generally such methods can be interpreted as an LU factorization scheme in which $A$ is factored $A = LU$, where $L$ is a lower triangular matrix and $U$ is an upper triangular matrix. Thus, $Ax = (LU)x = L(Ux) = b$, and the system can be solved by a simple forward substitution to solve $Ly = b$ for $y$, followed by a back substitution to find the solution vector $x$ from $Ux = y$.

The frontal method is an LU factorization technique that was originally developed to solve the banded matrices arising in finite element problems (Irons, 1970; Hood, 1976). The original motivation was, by limiting computational work to a relatively small *frontal matrix*, to be able to solve problems on machines with small core memories. Using codes such as MA42 (successor to the well-known MA32) from the Harwell Subroutine library, this method is widely applied to finite element problems on vector supercomputers, because, since the frontal matrix can be treated as dense, most of the computations involved can be performed by using very efficient vectorized dense matrix kernels. Stadtherr and Vegeais (1985) extended this idea to the solution of process simulation problems on supercomputers, and later (Vegeais and Stadtherr, 1990) demonstrated its potential. An implementation of the frontal method developed specifically for use in the process simulation context has been described by Zitney (1992), Zitney and Stadtherr (1993), and Zitney *et al.* (1995), and is now incorporated in supercomputer versions of popular process simulation and optimization codes.

The frontal elimination scheme can be outlined briefly as follows:

1. Assemble a row into the frontal matrix.

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

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

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

$$A^{(k)} = \begin{bmatrix} F^{(k)} & 0 \\ A_{ps}^{(k)} & A_{ns}^{(k)} \end{bmatrix}. \tag{1}$$

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

$$A^{(k)} = \begin{bmatrix} \bar{F}_{11}^{(k)} & \bar{F}_{12}^{(k)} & 0 \\ \bar{F}_{21}^{(k)} & \bar{F}_{22}^{(k)} & 0 \\ 0 & \bar{A}_{ps}^{(k)} & \bar{A}_{ns}^{(k)} \end{bmatrix}. \tag{2}$$

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

$$A^{(k)} = \begin{bmatrix} L_{11}^{(k)} U_{11}^{(k)} & U_{12}^{(k)} & 0 \\ L_{21}^{(k)} & F^{(k+g_k)} & 0 \\ 0 & A_{ps}^{(k+g_k)} & A_{ns}^{(k+g_k)} \end{bmatrix} \tag{3}$$

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

### 2.1 Application of frontal method in process engineering

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

FAMP has now been incorporated in CRAY versions of popular commercial simulation codes, such as ASPEN PLUS, SPEEDUP, RATEFRAC, and BATCHFRAC (Aspen Technology, Inc.). Zitney (1992) and Zitney *et al.* (1994) give several examples showing how the use of the frontal solver (as opposed to conventional solvers) has led to dramatic improvements in the performance of ASPEN PLUS and SPEEDUP. Zitney *et al.* (1995) describe a dynamic simulation problem at Bayer AG requiring 18 hours of CPU time on a CRAY C90 supercomputer when solved with the standard implementation of SPEEDUP. With a CRAY optimized version of SPEEDUP, which contains the frontal code FAMP and an improved residual evaluator CRAYRES, this simulation now takes only 21 CPU minutes, with most of the improvement due to the frontal solver. As a result, Bayer engineers can run this simulation many times per day instead of only once per day. This improves engineering productivity and let users consider more alternatives in a shorter time while not sacrificing model size and/or complexity.

## 3 Multifrontal Method

The multifrontal method is a generalization of the frontal method, and was originally developed for symmetric matrices. Like the frontal method, it also exploits low-level parallelism and vectorization through the use of dense matrix kernels on frontal matrices. However, the frontal matrices are generally smaller and denser than in the frontal method. The classical multifrontal approach (Duff and Reid, 1984) has met with only limited success when the pattern of nonzeros is highly unsymmetric. However, recently a new unsymmetric-pattern multifrontal algorithm has been described by Davis and Duff (1993,1997), and implemented in the code UMFPACK (Davis and Duff, 1995) (version 2.0 of UMFPACK is incorporated into the Harwell Subroutine Library as MA38). In this method, a frontal matrix, consisting of pivot row(s) and column(s), their entries from the original matrix $A$, and contributions to them from previous frontal matrices, is assembled at each stage of the factorization process. The frontal matrix $E_k$ for steps $k$ through $k+g_k-1$ of the LU factorization, where $g_k$ is the number of pivots performed in $E_k$ can be represented as

$$\begin{array}{cc} & \begin{array}{cc} C_k & C'_k \end{array} \\ \begin{array}{c} R_k \\ R'_k \end{array} & \left[ \begin{array}{cc} F_k & B_k \\ T_k & D_k \end{array} \right] \end{array} . \tag{4}$$

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

$$\begin{array}{cc} & \begin{array}{cc} C_k & C'_k \end{array} \\ \begin{array}{c} R_k \\ R'_k \end{array} & \left[ \begin{array}{cc} L_k U_k & U'_k \\ L'_k & D'_k \end{array} \right] \end{array} . \tag{5}$$

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

Zitney *et al.* (1996) have compared the performance of the general-purpose unsymmetric-pattern multifrontal approach (UMFPACK v.1.1) with that of the frontal code (FAMP), as well as with that of the conventional code MA28 (the relative performance of MA48, successor to MA28 in the Harwell Subroutine Library is discussed briefly in Section 3.2). Results on a set of six chemical process simulation problems and five other engineering problems show that the frontal and multifrontal methods are significantly faster than MA28, reflecting in part the use of vectorized dense matrix kernels. Comparing the frontal and multifrontal methods, the frontal method is most effective on problems with good initial matrix orderings, while the multifrontal method is most attractive for problems without a good initial ordering. For process simulation problems, good initial orderings are not uncommon if the equations describing each unit (or equilibrium stage) in a process are kept together, and if adjacent units and streams are numbered consecutively, thus resulting in a nearly block-banded matrix corresponding to the unit-stream nature of the problem. In the general-purpose multifrontal approach, a pivot element is chosen using a Markowitz-style strategy to preserve sparsity. Additional pivots may then be chosen to form a pivot block if they do not cause growth of the assembled frontal matrix beyond a preset limit. In the context of process simulation, the disadvantage of this approach is that it does not take advantage of the good initial structure of the matrix, and may in fact destroy it. The multifrontal algorithm presented below is designed to avoid this difficulty.

### 3.1 The MFA1P algorithm

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

**Algorithm MFA1P**:
For $k = 1 : n$

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

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

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

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

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

### 3.2 Results and discussion

Table 1 compares the performance of MFA1P with that of the frontal solver FAMP, the general-purpose unsymmetric-pattern multifrontal solver UMFPACK, and, to provide a familiar benchmark, the conventional solver MA28. Version 2.0 of UMFPACK was used; this version (Davis and Duff, 1995) incorporates features of the frontal method into the multifrontal solver in order to

| Name | $n$ | $NZ$ | $as$ | FAMP | MA38 | MA28 | MFA1P |
|---|---|---|---|---|---|---|---|
| v3 | 1078 | 16937 | 0.91 | 0.114 | 0.243 | 2.159 | **6.01x10$^{-2}$** |
| v10 | 1148 | 15729 | 0.94 | 0.109 | 0.227 | 1.862 | **6.10x10$^{-2}$** |
| v13 | 834 | 9713 | 0.95 | 6.35x10$^{-2}$ | 0.140 | 0.983 | **4.20x10$^{-2}$** |
| mpex2 | 848 | 11413 | 0.96 | 6.60x10$^{-2}$ | 0.176 | 0.299 | **4.27x10$^{-2}$** |
| mpex3 | 2473 | 46503 | 0.94 | 0.359 | 0.567 | 10.598 | **0.173** |
| mpex4 | 2478 | 44075 | 0.95 | 0.317 | 0.559 | 9.19 | **0.172** |
| mpmult1 | 2023 | 31894 | 0.95 | 0.234 | 0.472 | 6.131 | **0.13** |
| rdist1 | 4134 | 94408 | 0.94 | 0.730 | 1.854 | 30.21 | **0.32** |
| rdist2 | 3198 | 56934 | 0.95 | 0.392 | 0.696 | 16.11 | **0.22** |
| rdist3 | 2398 | 61896 | 0.85 | 0.478 | 1.172 | 32.87 | **0.20** |
| sumb | 523 | 4998 | 0.95 | 3.22x10$^{-2}$ | 8.30x10$^{-2}$ | 0.314 | **2.18x10$^{-2}$** |
| traycalc | 1145 | 20296 | 0.88 | 0.14 | 0.244 | 2.649 | **6.81x10$^{-2}$** |
| uosb | 523 | 4998 | 0.95 | 3.22x10$^{-2}$ | 8.29x10$^{-2}$ | 0.315 | **2.19x10$^{-2}$** |
| userupp | 1269 | 22508 | 0.89 | 0.154 | 0.289 | 3.310 | **7.39x10$^{-2}$** |

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

improve overall efficiency; it is incorporated into the Harwell Subroutine Library as MA38. The default parameter settings were used for each code. Numerical experiments were carried out on a CRAY C90 parallel/vector supercomputer at Cray Research, Inc. in Eagan, Minnesota (USA). The problem set includes 14 steady-state simulation problems solved using ASPEN PLUS (Aspen Technology, Inc.). These problems use the RADFRAC module of ASPEN PLUS. This module does rigorous calculations for all types of fractionation, including absorption, reboiled absorption, stripped, reboiled stripping, and extractive, azeotropic, and three phase distillation, in addition to ordinary distillation.

In Table 1, each matrix is identified by name and order ($n$). In addition, statistics are given for the number of nonzeros ($NZ$), and for a measure of structural asymmetry ($as$). The asymmetry, $as$, is the number off-diagonal nonzeros $a_{ij}$ ($j \neq i$) for which $a_{ji} = 0$ divided by the total number of off-diagonal nonzeros ($as = 0$ is a symmetric pattern, $as = 1$ is completely asymmetric). Run times (in CPU seconds) represent the total time to perform analysis to determine a pivot sequence, to compute the $L$ and $U$ factors of $A$, and to perform the forward and backward substitution to solve $Ax = b$. This execution path (**A**nalyze + **F**actor + **S**olve) is typically used at each iteration in a steady-state simulation. The fastest run time for each problem is shown in bold. Data for the factor only and solve only execution paths are given by Mallya and Stadtherr (1997), along with results for a variety of other problems. Mallya and Stadtherr (1997) also describe a version (MFA2P) of this multifrontal approach in which two pivots are performed in each frontal matrix.

For the critical A + F + S execution path, the MFA1P solver is the best on all problems and on the average is 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 that MA48, the successor to MA28 in the Harwell Subroutine

Library, should perform better than MA28 on these problems, but still not better than the other codes on the A+F+S execution path. For instance, on the *rdist1* problem, Davis and Duff (1995) found that, on a CRAY C90, MA38 was about six times faster than MA48.

# 4 Parallel Frontal Method

The main deficiency with the frontal code FAMP and multifrontal code MFA1P is that there is little opportunity for parallelism beyond that which can be achieved by microtasking the inner loops or by using higher level BLAS in performing the outer product update, which unfortunately usually provides relatively little speedup (Mallya, 1996; Camarda and Stadtherr, 1994). We overcome this problem by using a coarse-grained parallel approach in which frontal elimination is performed simultaneously in multiple independent or loosely connected blocks. This can be interpreted as applying frontal elimination to the diagonal blocks in a bordered block-diagonal matrix form as described below. It can also be interpreted as a coarse-grained multifrontal approach (e.g., Davis and Duff, 1997; Zitney *et al.*, 1996) with large independent pivot blocks factored by frontal elimination. Duff and Scott (1994) have applied this type of approach in solving finite element problems and referred to it as a "multiple fronts" (as opposed to multifrontal) approach.

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

$$A = \begin{bmatrix} A_{11} & & & \\ & A_{22} & & \\ & & \ddots & \\ & & & A_{NN} \\ S_1 & S_2 & \dots & S_N \end{bmatrix} \tag{6}$$

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

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

## 4.1 The PFAMP algorithm

The basic PFAMP algorithm is outlined below, followed by a more detailed explanation of the key steps. For complete details, see Mallya *et al.* (1997a).

**Algorithm PFAMP**:
*Begin parallel computation on P processors*
For $i = 1 : N$, with each task $i$ assigned to the next available processor:

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

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

3. Perform frontal elimination on $A_{ii}$, beginning with the assembly of the first row of $A_{ii}$ into the frontal matrix (see Section 2). The maximum number of variables that can be eliminated is $m_i$, but the actual number of pivots done is $p_i \leq m_i$. We use a partial-threshold pivoting strategy to ensure that the pivot row belongs to the diagonal block $A_{ii}$. We cannot pick a pivot row from the border $S_i$ because border rows may be shared by more than one diagonal block.

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

*End parallel computation*

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

Note that for each block the result of Step 3 is

$$\begin{array}{cc} & \begin{array}{cc} C_i & C_i' \end{array} \\ \begin{array}{c} R_i \\ R_i' \end{array} & \left[ \begin{array}{cc} L_i U_i & U_i' \\ L_i' & F_i \end{array} \right] \end{array} \tag{7}$$

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

$$\begin{array}{cc} & \begin{array}{ccccc} C_1 & C_2 & \dots & C_N & C' \end{array} \\ \begin{array}{c} R_1 \\ R_2 \\ \vdots \\ R_N \\ R' \end{array} & \left[ \begin{array}{ccccc} L_1 U_1 & & & & U_1' \\ & L_2 U_2 & & & U_2' \\ & & \ddots & & \vdots \\ & & & L_N U_N & U_N' \\ L_1' & L_2' & \dots & L_N' & F \end{array} \right] \end{array} \tag{8}$$

where $R' = \bigcup_{i=1}^{N} R'_i$ and $C' = \bigcup_{i=1}^{N} C'_i$. $F$ is the interface matrix that can be assembled from the contribution blocks $F_i$. Note that, since a row index in $R'$ may appear in more than one of the $R'_i$ and a column index in $C'$ may appear in more than one of the $C'_i$, some elements of $F$ may get contributions from more than one of the $F_i$. As this doubly-bordered block-diagonal form makes clear, once values of the variables in the interface problem have been solved for, the remaining triangular solves needed to complete the solution can be done in parallel using the same decomposition used to do the parallel frontal elimination. During this process the solution to the interface problem is made globally available to each processor.

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

### 4.2 Results and discussion

In this section, we present results for the performance of the PFAMP solver on several process optimization and simulation problems. We compare the performance of PFAMP on multiple processors with its performance on one processor and with the performance of the frontal solver FAMP on one processor. The numerical experiments were performed on a CRAY C90 parallel/vector supercomputer at Cray Research, Inc., in Eagan, Minnesota. The timing results presented represent the total time to obtain a solution vector from one right-hand-side vector, including analysis, factorization, and triangular solves. A threshold tolerance of $t = 0.1$ was used in PFAMP to maintain numerical stability, which was monitored using the 2-norm of the residual $b - Ax$. FAMP uses partial pivoting.

In Table 2 each matrix is identified by name and order ($n$). In addition, statistics are given for the number of nonzeros ($NZ$), and for a measure of structural asymmetry ($as$), as defined above. Also given is information about the bordered block-diagonal form used, namely the number of diagonal blocks ($N$), the order of the interface matrix ($NI$), and the number of equations in the largest and smallest diagonal blocks, $m_{i,max}$ and $m_{i,min}$, respectively. $P$ is the number of processors used for evaluating the parallel performance of PFAMP.

The first three problems involve the optimization of an ethylene plant using NOVA, a chemical process optimization package from Dynamic Optimization Technology Products, Inc. NOVA uses an equation-based approach that requires the solution of a series of large sparse linear systems, which accounts for a large portion of the total computation time. The linear systems arising during optimization with NOVA are in bordered block-diagonal form, allowing the direct use of PFAMP for the solution of these systems. Each problem involves a flowsheet that consists of 43 units, including five distillation columns. The problems differ in the number of stages in the distillation columns.

The next five problems have been reordered into a bordered block-diagonal form using the Minimum-Net-Cut (MNC) approach (Coon and Stadtherr, 1995). Two of the problems (*Hydr1c*

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

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

and *Icomp*) occur in dynamic simulation problems solved using SPEEDUP (Aspen Technology, Inc.). The *Hydr1c* problem involves a 7-component hydrocarbon process with a de-propanizer and a de-butanizer. The *Icomp* problem comes from a plantwide dynamic simulation of a plant that includes several interlinked distillation columns. The other three problems are derived from the prototype simulator SEQUEL (Zitney and Stadtherr, 1988), and are based on light hydrocarbon recovery plants, described by Zitney *et al.* (1996). Neither of the application codes produces directly a matrix in bordered block-diagonal form, so a reordering such as provided by MNC is required.

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

Table 3 shows the performance of PFAMP. We note first, that the single processor performance of PFAMP is usually better than that of FAMP. This is due to the difference in the size of the largest frontal matrix associated with the frontal elimination for each method. For solution with FAMP, the variables which have occurrences in the border equations remain in the frontal matrix until the end. The size of the largest frontal matrix increases for this reason, as does the number of wasted operations on zeros, thereby reducing the overall performance. This problem does not arise for solution with PFAMP because when the factorization of a diagonal block is complete, the remaining variables and equations in the front are immediately written out as part of the interface problem and a new front is begun for the next diagonal block. Thus, usually PFAMP is a more efficient serial solver than FAMP. This reflects the advantages of the multifrontal-type approach used by PFAMP, namely smaller and less sparse frontal matrices.

In each of the three ethylene plant matrices, there are five large diagonal blocks, corresponding to the distillation units, with one of these blocks much larger ($m_i = 3337$) than the others ($1185 \leq m_i \leq 1804$). In the computation, one processor ends up working on the largest block, while the remaining four processors finish the other large blocks and the several much smaller ones. The load

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

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

is unbalanced with the factorization of the largest block being the bottleneck. This, together with the solution of the interface problem, results in a speedup (relative to PFAMP on one processor) of less than two on five processors. It is likely that more efficient processor utilization could be obtained by using a better partition into bordered block-diagonal form.

For MNC-reordered SPEEDUP and SEQUEL matrices, the speedup is around two. MNC achieves the best reordering on the *Icomp* problem, for which it finds four diagonal blocks of roughly the same size ($17168 \leq m_i \leq 17393$) and the size of the interface problem is relatively small in comparison to $n$. The speedup observed for PFAMP on this problem was about 2.5 on four processors. While this represents a substantial savings in wallclock time, it still does not represent particularly efficient processor utilization.

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

For none of the problems considered above was the efficiency of processor utilization particularly high. In this context, it should be remembered that even a relatively small serial component in a computation can greatly reduce the efficiency of processor utilization [see Vegeais and Stadtherr (1992) for further discussion of this point]. Improved reorderings to bordered block diagonal form can provide some improvements along these lines (Mallya *et al.*, 1997b). However, because of the unsymmetric and irregular nature of process engineering matrices, achieving extremely efficient processor utilization may be very difficult. Nevertheless, significant speedups can still be achieved, not only by considering the linear algebra component of the problem, but also function evaluation, derivative evaluation, and other aspects of the problem.

# 5 Reliable Computing

It is important to realize that not only does HPC provide the power to increase the speed with which problems can be solved, but also the reliability with which they can be solved. In process optimization a consistent issue concerning reliability is whether or not a global, as opposed to local, optimum has been achieved. In process modeling, especially in the presence of highly nonlinear models, the issue of whether a solution is unique is of concern, and if no solution is found, of whether there actually exists a solution to the posed problem. It is a common misconception that such difficulties cannot be resolved except in special cases. For example, in a section entitled "What is not possible," Dennis and Schnabel (1983) state that "In general, the questions of existence and uniqueness—does a given problem have a solution and is it unique?—are beyond the capabilities one can expect of algorithms that solve nonlinear problems." More recently, in the textbook of Heath (1997) it is stated concerning nonlinear equation systems that "It is not possible, in general, to guarantee convergence to the correct solution or to bracket the solution to produce an absolutely safe method." However, in fact there do exist methods, based on interval computations, that can, given a system of equations with a finite number of solutions in a specified initial interval, find *with mathematical certainty* any and all solutions to a specified tolerance, or can determine *with mathematical certainty* that there are none. Not only do these techniques provide the power to find with certainty all solutions of a system of nonlinear equations, but also to find with total confidence the global minimum of a nonlinear objective function, again provided only that upper and lower bounds are available for all variables. While such methods are not new and date to the pioneering work of Moore (1966), the computational power to efficiently implement these techniques are a relatively recent development, and thus such methods have not yet been widely applied. Schnepper and Stadtherr (1990) suggested the use of these techniques for solving chemical process modeling problems, and recently described both parallel and serial implementations (Schnepper and Stadtherr, 1996). Balaji *et al.* (1995) have also successfully applied these methods to chemical engineering problems. Stadtherr *et al.* (1995), McKinnon *et al.* (1996), and Hua *et al.* (1996a,b) have shown how to use interval computations to reliably solve phase stability problems, and this will be used as an example below. For good introductions to interval computations, including their use in nonlinear equation solving and global optimization, the recent monographs of Neumaier (1990), Hansen (1992), and Kearfott (1996) are available.

## 5.1 Phase stability analysis

The determination of phase stability, i.e., whether or not a given mixture can split into multiple phases, is a key step in phase equilibrium calculations, and thus in the simulation and design of a wide variety of processes, especially those involving separation operations such as distillation and extraction. We use it here as an example of the power of interval computations to reliably solve difficult modeling and optimization problems.

The phase stability problem is frequently formulated in terms of the tangent plane condition (Baker *et al.*, 1982). Minima in the tangent plane distance are sought, usually by solving a system of nonlinear equations for the stationary points (Michelsen, 1982). If any of these yield a negative tangent plane distance, indicating that the tangent plane intersects (or lies above) the Gibbs energy of mixing surface, the phase is unstable and can split (in this context, unstable refers to both the metastable and classically unstable cases). The difficulty lies in that, in general, given any arbitrary equation of state or activity coefficient model, most computational methods cannot find with complete certainty all the stationary points, and thus there is no guarantee that the phase stability problem has been correctly solved.

Standard methods (e.g., Michelsen, 1982) for solving the phase stability problem typically rely on the use of multiple initial guesses, carefully chosen in an attempt to locate all stationary points in the tangent plane distance function. However, these methods offer no guarantee that the global minimum in the tangent plane distance has been found. Because of the difficulties that thus arise, there has been significant recent interest in the development of more reliable methods for solving the phase stability problem (e.g., Nagarajan *et al.*, 1991; Sun and Seider, 1995; Eubank *et al.*, 1992; Wasylkiewicz *et al.*, 1996; McDonald and Floudas, 1995a,b,c,1997). For example, Sun and Seider (1995) apply a homotopy-continuation method, which will often find all the stationary points, and is easier to initialize than Michelsen's approach. However, their technique is still initialization dependent and provides no theoretical guarantees that all stationary points have been found. McDonald and Floudas (1995a,b,c,1997) show that for certain activity coefficient models, the phase stability problem can be reformulated to make it amenable to solution by powerful global optimization techniques, generally involving branch and bound using convex underestimating functions. While this type of approach can offer mathematical guarantees, it does not offer computational guarantees in practice, since it does not deal rigorously with rounding error. As shown by the example given originally by Rump (1988) and also discussed by Hansen (1992), the impact of rounding error is something that should not be taken lightly.

An alternative approach for solving the phase stability problem, based on interval analysis, that provides both mathematical and computational guarantees of global optimality, with resolution limited only by machine precision, was originally suggested by Stadtherr *et al.* (1995), who applied it in connection with activity coefficient models, as later done also by McKinnon *et al.* (1996). This technique, in particular the use of an interval Newton/generalized bisection algorithm, is initialization independent and can solve the phase stability problem with mathematical certainty, while also dealing automatically with rounding error. Recently Hua *et al.* (1996a,b) extended this method to problems modeled with cubic equation of state (EOS) models, in particular the Van der Waals (VDW), Peng-Robinson (PR) and Soave-Redlich-Kwong (SRK) models with standard mixing rules. It should be emphasized however, that the technique is general-purpose and can be applied in connection with any equation of state or excess Gibbs energy model.

According to tangent plane analysis (Baker *et al.*, 1982; Michelsen, 1982), a phase at specified temperature $T$, pressure $P$, and feed mole fraction $\mathbf{z}$ is unstable if the molar Gibbs energy of mixing surface $m(\mathbf{x}, v) = \Delta g_{mix} = \Delta \hat{G}_{mix}/RT$ ever falls below a plane tangent to the surface at $\mathbf{z}$. That is, if the tangent plane distance

$$D(\mathbf{x}, v) = m(\mathbf{x}, v) - m_0 - \sum_{i=1}^{n} \left( \frac{\partial m}{\partial x_i} \right)_0 (x_i - z_i) \tag{9}$$

is negative for any composition $\mathbf{x}$, the phase is unstable. The subscript zero indicates evaluation at $\mathbf{x} = \mathbf{z}$, $n$ is the number of components, and $v$ is the molar volume of the mixture. A common approach for determining if $D$ is ever negative is to minimize $D$ subject to the mole fractions summing to one and subject to the equation of state relating $\mathbf{x}$ and $v$. It is readily shown that the stationary points in this optimization problem can be found by solving the system of nonlinear equations:

$$\left[\left(\frac{\partial m}{\partial x_i}\right) - \left(\frac{\partial m}{\partial x_n}\right)\right] - \left[\left(\frac{\partial m}{\partial x_i}\right) - \left(\frac{\partial m}{\partial x_n}\right)\right]_0 = 0, \quad i = 1, \ldots, n-1 \tag{10}$$

$$1 - \sum_{i=1}^{n} x_i = 0 \tag{11}$$

$$P - \frac{RT}{v-b} + \frac{a}{v^2 + ubv + wb^2} = 0 \tag{12}$$

Equation (12) is the generalized cubic EOS given by Reid *et al.* (1987). With the appropriate choice of $u$ and $w$, common models such as PR ($u = 2$, $w = -1$), SRK ($u = 1$, $w = 0$), and VDW ($u = 0$, $w = 0$) may be obtained. For the example considered here, standard mixing rules, namely $b = \sum_{i=1}^{n} x_i b_i$ and $a = \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j a_{ij}$, are used, with $a_{ij} = (1 - k_{ij})\sqrt{a_{ii} a_{jj}}$. The $a_{ii}(T)$ and $b_i$ are pure component properties determined from the system temperature $T$, the critical temperatures $T_{ci}$, the critical pressures $P_{ci}$ and acentric factors $\omega_i$. If there are multiple real volume roots at the feed composition $\mathbf{z}$, then in evaluating equations (9) and (10), the molar volume $v_0$ at the feed composition must be the root yielding the minimum value of $m_0 = m(\mathbf{z}, v_0)$, the reduced molar Gibbs energy of mixing at the feed.

The $(n+1) \times (n+1)$ system given by equations (10)–(12) above has a trivial root at $(\mathbf{x}, v) = (\mathbf{z}, v_0)$ and frequently has multiple nontrivial roots as well. Thus conventional equation solving techniques may fail by converging to the trivial root or give an incorrect answer to the phase stability problem by converging to a stationary point that is not the global minimum of $D$. This is aptly demonstrated by the experiments of Green *et al.* (1993), who show that the pattern of convergence from different initial guesses demonstrates a complex fractal-like behavior for even very simple models like VDW. We show here that the interval Newton/generalized bisection method can reliably solve the system of equations (10)–(12). The method requires no initial guess, and will find with certainty all the stationary points of the tangent plane distance $D$.

## 5.2 Interval computations

A real *interval*, $X$, is defined as the continuum of real numbers lying between (and including) given upper and lower bounds; i.e., $X = [a, b] = \{x \in \Re \mid a \leq x \leq b\}$, where $a, b \in \Re$ and $a \leq b$. A real interval vector $\mathbf{X} = (X_i) = (X_1, X_2, ..., X_n)^T$ has $n$ real interval components and since it can be interpreted geometrically as an $n$-dimensional rectangle, is frequently referred to as a *box*. Note that in this section lower case quantities are real numbers and upper case quantities are intervals.

Interval arithmetic is an extension of real arithmetic. For a basic arithmetic operation op $\in \{+, -, \times, \div\}$, the corresponding interval operation for intervals $X = [a, b]$ and $Y = [c, d]$ is defined by $X \text{ op } Y = \{x \text{ op } y \mid x \in X, y \in Y\}$. Formulae for computing these basic interval operations are well known, for example $X + Y = [a + c, b + d]$. Such formulae assume that we are able to compute the endpoints of an interval result exactly. Of course, when this is done on a computer, there may be round-off problems, so steps must be taken to ensure that the resulting intervals enclose the exact values. The use of rounded-interval arithmetic solves this problem. Essentially a directed outward rounding is used, so that when a lower endpoint $a$ is computed, it is rounded down to the largest machine-representable number less than or equal to $a$, and when an upper endpoint $b$ is computed, it is rounded up to the smallest machine-representable number greater than or equal to $b$. In this way, when interval arithmetic is used, rounding error is readily accounted for. Outwardly rounded interval extensions of the elementary functions (e.g., exponentiation, logarithm, exponential) are also available.

Of particular interest here are interval Newton/generalized bisection (IN/GB) methods. Consider the solution of the system of real nonlinear equations $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, where it is desired to find all solutions in an specified initial interval $\mathbf{X}^{(0)}$. The basic iteration step in interval Newton methods is, given an interval $\mathbf{X}^{(k)}$, to solve the linear interval equation system

$$F'(\mathbf{X}^{(k)})(\mathbf{N}^{(k)} - \mathbf{x}^{(k)}) = -\mathbf{f}(\mathbf{x}^{(k)}) \tag{13}$$

for a new interval $\mathbf{N}^{(k)}$, where $k$ is an iteration counter, $F'(\mathbf{X}^{(k)})$ is an interval extension of the real Jacobian $f'(\mathbf{x})$ of $f(\mathbf{x})$ over the current interval $\mathbf{X}^{(k)}$, and $\mathbf{x}^{(k)}$ is a point in the interior of $\mathbf{X}^{(k)}$, usually taken to be the midpoint. It can be shown (Moore, 1966) that any root $\mathbf{x}^*$ of the set of equations that is within the current interval, i.e. $\mathbf{x}^* \in \mathbf{X}^{(k)}$, is also contained in the newly computed interval $\mathbf{N}^{(k)}$. This suggests that the next iteration for $\mathbf{X}$ should be the intersection of $\mathbf{X}^{(k)}$ with $\mathbf{N}^{(k)}$, i.e. $\mathbf{X}^{(k+1)} = \mathbf{X}^{(k)} \cap \mathbf{N}^{(k)}$. There are various interval Newton methods, which differ in how they determine $\mathbf{N}^{(k)}$ from equation (13) and thus in the tightness with which $\mathbf{N}^{(k)}$ encloses the solution set of (13).

While the iteration scheme discussed above can be used to tightly enclose a solution, what is of most significance here is the power of equation (13) to provide a test of whether a solution exists within a given interval and whether it is a unique solution. For several techniques for finding $\mathbf{N}^{(k)}$ from equation (13), it can be proven (e.g., Neumaier, 1990) that if $\mathbf{N}^{(k)}$ is totally contained within $\mathbf{X}^{(k)}$, i.e. $\mathbf{N}^{(k)} \subset \mathbf{X}^{(k)}$, then there is a *unique* zero of the set of nonlinear equations $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ in $\mathbf{X}^{(k)}$, and furthermore that Newton's method with real arithmetic will converge to that solution starting from any point in $\mathbf{X}^{(k)}$. Thus, if $\mathbf{N}^{(k)}$ is determined using one of these techniques, the computation can be used as part of a root inclusion test for any interval $\mathbf{X}^{(k)}$:

1. If $\mathbf{X}^{(k)}$ and $\mathbf{N}^{(k)}$ do not intersect, i.e., $\mathbf{X}^{(k)} \cap \mathbf{N}^{(k)} = \emptyset$, then there is no root in $\mathbf{X}^{(k)}$.

2. If $\mathbf{N}^{(k)}$ is totally contained in $\mathbf{X}^{(k)}$, then there is exactly one root in $\mathbf{X}^{(k)}$ and Newton's method with real arithmetic will find it.

3. If neither of the above is true, then no conclusion can be drawn.

In the last case, one could then repeat the root inclusion test on the next interval Newton iterate $\mathbf{X}^{(k+1)}$, assuming it is sufficiently smaller than $\mathbf{X}^{(k)}$, or one could bisect $\mathbf{X}^{(k+1)}$ and repeat the root inclusion test on the resulting intervals. This is the basic idea of IN/GB methods. If $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ has a finite number of real solutions in the specified initial box, a properly implemented IN/GB method can find *with mathematical certainty* any and all solutions to a specified tolerance, or can determine *with mathematical certainty* that there are no solutions in the given box (Kearfott and Novoa, 1990; Kearfott, 1990). The technique used here for computing $\mathbf{N}^{(k)}$ from equation (13) is the preconditioned Gauss-Seidel-like technique developed by Hansen and Sengupta (1981). A detailed step-by-step description of the IN/GB algorithm used here is given by Schnepper and Stadtherr (1996). It should be noted that, prior to executing the interval-Newton procedure outlined above, the root inclusion test first computes an interval extension $\mathbf{F}(\mathbf{X}^{(k)})$ containing the range of $\mathbf{f}(\mathbf{x})$ over $\mathbf{X}^{(k)}$ and tests to see whether it contains zero. Clearly, if $0 \notin \mathbf{F}(\mathbf{X}^{(k)}) \supseteq \{\mathbf{f}(\mathbf{x}) \mid \mathbf{x} \in \mathbf{X}^{(k)}\}$ then there can be no solution of $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ in $\mathbf{X}^{(k)}$and this interval need not be further tested using the interval-Newton procedure. Furthermore, it should be emphasized that since the root inclusion tests can be performed independently on all the intervals generated by bisection, this technique is readily parallelized.

The system of equations (10)–(12) that must be solved here involves $n+1$ variables, the $n$ component mole fractions $\mathbf{x}$ and the molar volume $v$. For the mole fraction variables, initial intervals of [0,1] are suitable. In practice the initial lower bound is set to an arbitrarily small positive number $\varepsilon$ ($10^{-10}$ was used) to avoid taking the logarithm of zero in subsequent calculations. This can be done without the loss of reliability providing a sufficiently small value of $\varepsilon$ is used. The lower limit on the molar volume was taken to be the smallest pure component size parameter $b_i$, and the upper bound was taken to be the ideal gas molar volume for the $T$ and $P$ under investigation. Although it is possible to have compressibility factors greater than one at very high reduced pressure, this was deemed satisfactory for reduced temperature and pressure used in the example below. Our implementation of the IN/GB method for the phase stability problem is based on appropriately modified routines from the packages INTBIS (Kearfott and Novoa, 1990) and INTLIB (Kearfott *et al.*, 1994).

### 5.3 Example

This is a mixture of hydrogen sulfide (1) and methane (2) at 190 K and 40.53 bar (40 atm.). The SRK model was used with parameters calculated from $T_{c1} = 373.2$ K, $P_{c1} = 89.4$ bar, $\omega_1 = 0.1$, $T_{c2} = 190.6$ K, $P_{c2} = 46.0$ bar, $\omega_2 = 0.008$, and a binary interaction parameter $k_{12} = 0.08$.

Several feeds were considered, as shown in Table 4, which also shows the roots (stationary points) found, and the value of the tangent plane distance $D$ at each root. For the $z_1 = 0.5$ case, our results are consistent with those given by Sun and Seider (1995) for this problem. For feeds near the $z_1 = 0.0187$ case, this is known to be a difficult problem to solve (e.g., Michelsen, 1982; Sun and Seider, 1995). As noted by Michelsen and others, if one uses a locally convergent solver, with nearly pure $CH_4$ as the initial guess, convergence will likely be to the trivial solution at $x_1 = z_1 = 0.0187$. And if nearly pure $H_2S$ is the initial guess, convergence will likely be to the local, but not global, minimum at $x_1 = 0.8848$. Using only these initial guesses would lead to the incorrect conclusion that the mixture is stable. This is indicative of the importance of the initialization strategy when conventional methods are used. An important advantage of the IN/GB approach described here is that it eliminates the initialization problem, since it is initialization independent. In this case, it finds all the stationary points, including the global minimum at $x_1 = 0.0767$, correctly predicting, since $D < 0$ at this point, that a mixture with this feed composition is unstable. Michelsen's algorithm, as implemented in LNGFLASH from the IVC-SEP package (Hytoft and Gani, 1996), a code that in general we have found to be extremely reliable, incorrectly predicts that this mixture is stable. As indicated in Table 4, several other feed compositions were tested using the IN/GB approach, with correct results obtained in each case. Note that the presence of multiple real volume roots does not present any difficulty, since the solver simply finds all roots for the given system.

Also included in Table 4 are the number of root inclusion tests performed in the computation and the total CPU time on a Sun Ultra 1/170 workstation. Results for several other problems are available (Hua *et al.*, 1997) We would expect standard approaches to the phase stability problem to be faster, but these methods do not reliably solve the problem in all cases. Thus, as one might expect, to obtain guaranteed reliability some premium must be paid in terms of computation time. While this problem is too small for the use of parallel computing to have much impact, for larger

| Feed $(z_1, z_2)$ | Roots $(x_1, x_2, v)$ | $D$ | Number of Root Inclusion Tests | CPU Time (sec) Sun Ultra 1/170 |
|---|---|---|---|---|
| (0.0115, 0.9885) | (0.0115, 0.9885, 212.8) | 0.0 | 1079 | 0.23 |
| | (0.0237, 0.9763, 97.82) | 0.0137 | | |
| | (0.0326, 0.9674, 78.02) | 0.0130 | | |
| (0.0187, 0.9813) | (0.8848, 0.1152, 36.58) | 0.0109 | 1428 | 0.29 |
| | (0.0187, 0.9813, 207.3) | 0.0 | | |
| | (0.0313, 0.9687, 115.4) | 0.0079 | | |
| | (0.0767, 0.9233, 64.06) | -0.004 | | |
| | (0.4905, 0.5095, 41.50) | 0.0729 | | |
| (0.07, 0.93) | (0.8743, 0.1257, 36.65) | 0.0512 | 1414 | 0.30 |
| | (0.5228, 0.4772, 40.89) | 0.0965 | | |
| | (0.0178, 0.9822, 208.0) | 0.0015 | | |
| | (0.0304, 0.9696, 113.7) | 0.0100 | | |
| | (0.07 , 0.93 , 65.35) | 0.0 | | |
| (0.50, 0.50) | (0.8819, 0.1181, 36.60) | -0.057 | 1416 | 0.29 |
| | (0.0184, 0.9816, 207.5) | -0.079 | | |
| | (0.0311, 0.9689, 114.9) | -0.071 | | |
| | (0.0746, 0.9254, 64.44) | -0.082 | | |
| | (0.50 , 0.50 , 41.32) | 0.0 | | |
| (0.888, 0.112) | (0.888 , 0.112 , 36.55) | 0.0 | 1412 | 0.30 |
| | (0.0190, 0.9810, 207.1) | 0.0026 | | |
| | (0.0316, 0.9684, 116.0) | 0.0103 | | |
| | (0.0792, 0.9208, 63.60) | -0.002 | | |
| | (0.4795, 0.5205, 41.72) | 0.0683 | | |
| (0.89, 0.11) | (0.89 , 0.11 , 36.54) | 0.0 | 1411 | 0.29 |
| | (0.0192, 0.9808, 206.9) | 0.0113 | | |
| | (0.0319, 0.9681, 116.4) | 0.0189 | | |
| | (0.0809, 0.9191, 63.31) | 0.0058 | | |
| | (0.4725, 0.5275, 41.87) | 0.0724 | | |

Table 4: Results for example problem: Hydrogen sulfide (1) and methane (2) at $P = 40.53$ bar and $T = 190$ K, modeled using SRK.

problems and other application areas, the use of high performance computing will be important in using interval computations for global optimization.

## 6 Concluding Remarks

In the first part of this paper, the focus was on using HPC for increasing the speed of process simulation and optimization computations. To better use this leading edge technology in process engineering requires the use of techniques that efficiently exploit vector and parallel processing, and thus it is often necessary to rethink problem solving strategies. We have seen here how a simple multifrontal approach (MFA1P) can be used to efficiently solve, in a vector processing environment, the sparse linear equation systems that arise in the simulation of equilibrium-stage processes. By taking advantage of the problem's structure, the new approach provides significant improvements over both the standard frontal solver FAMP and the general-purpose multifrontal solver MA38 (from the Harwell Subroutine Library). We have also seen that the parallel frontal solver PFAMP can be effective for use in process simulation and optimization on parallel machines 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.

Not only does HPC provide the power to increase the speed with which problems can be solved, but also the reliability with which they can be solved. In the second part of the paper the focus was on an approach, based on interval mathematics, that is capable of guaranteeing the reliable solution of process engineering problems, and that is well suited to parallel processing. As an example of this approach we saw that the interval Newton/generalized bisection algorithm can solve phase stability problems for a generalized cubic equation of state model efficiently and with complete reliability. This work represents an entirely new method for solving these problems, a method that can guarantee with mathematical certainty that the correct solutions are found, thus eliminating computational problems that are frequently encountered with currently available techniques. The method is initialization independent; it is also model independent, straightforward to use, and can be applied in connection with other equations of state or with activity coefficient models. This represents a very powerful problem solving technique that, especially when combined with the power of high performance computing, will find applications in many areas of process modeling and optimization.

**Acknowledgments** – This work has been supported by the National Science Foundation under Grants DMI-9322682 and DMI-9696110, and by the donors of The Petroleum Research Fund, administered by the ACS, under Grant 30421-AC9. We also acknowledge the support of the National Center for Supercomputing Applications at the University of Illinois, Cray Research, Inc., Sun Microsystems, Inc., and Aspen Technology, Inc. We thank Kirk Abbott for providing the ASCEND matrices and the tear_drop reorderings.

## References

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

Baker, L. E., Pierce, A. C. & Luks, K. D. 1982 Gibbs energy analysis of phase equilibria. *Soc. Petrol. Engrs. J.* **22**, 731–742.

Balaji, G. V. & Seader, J. D. 1995 Application of interval-newton method to chemical engineering problems. *AIChE Symp. Ser.* **91**(304), 364–367.

Brüll, L. 1997 Optimization of entire chemical plants. In *Computational Chemistry and Chemical Engineering*, World Scientific, in press.

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

Chen, H. S. & Stadtherr, M. A. 1985 A simultaneous modular approach to process flowsheeting and optimization: I. Theory and implementation. *AIChE J.* **31**, 1843–1856.

Chimowitz, E. H. & Bielinis, R. Z. 1987 Analysis of parallelism in modular flowsheet calculations. *AIChE J.* **33**, 976.

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

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

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

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

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

Dennis, J. E. & Schnabel, R. B. 1983 *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Englewood Cliffs, NJ.

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

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

Eubank, A. C., Elhassan, A. E., Barrufet, M. A., & Whiting, W. B. 1992 Area method for prediction of fluid-phase equilibria. *Ind. Eng. Chem. Res.* **31**, 942–949.

Green, K. A., Zhou, S. & Luks, K. D. 1993 The fractal response of robust solution techniques to the stationary point problem. *Fluid Phase Equilibria* **84**, 49–78.

Hansen, E. R. 1992 *Global Optimization Using Interval Analysis.* Marcel Dekkar, New York, NY.

Hansen, E. R. & Sengupta, R. I. 1981 Bounding solutions of systems of equations using interval analysis. *BIT* **21**, 203–211.

Heath, M. T. 1997 *Scientific Computing: An Introductory Survey.* McGraw-Hill, New York, NY.

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

Hua, J. Z., Brennecke, J. F. & Stadtherr, M. A. 1996a Reliable prediction of phase stability using an interval-Newton method. *Fluid Phase Equilibria* **116**, 52–59.

Hua, J. Z., Brennecke, J. F., & Stadtherr, M. A. 1996b Reliable phase stability analysis for cubic equation of state models. *Comput. Chem. Eng.* **20**, S395–S400.

Hua, J. Z., Brennecke, J. F., & Stadtherr, M. A. 1997 Enhanced interval analysis for phase stability: Cubic equation of state models. Submitted for publication.

Hytoft, G. & Gani, R. 1996 IVC-SEP Program Package. Technical Report SEP 8623, Institut for Kemiteknik, Danmarks Tekniske Universitet, Lyngby, Denmark.

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

Kearfott, R. B. 1990 Interval arithmetic techniques in the computational solution of nonlinear systems of equations: Introduction, examples, and comparisons. *Lectures in Applied Mathematics* **26**, 337–357.

KEARFOTT, R. B. 1996 *Rigorous Global Search: Continuous Problems.* Kluwer Academic Publishers, Dordrecht, Netherlands.

KEARFOTT, R. B., DAWANDE, M. , DU, K.-S. & HU, C.-Y. 1994 Algorithm 737: INTLIB, a portable FORTRAN 77 interval standard function library. *ACM Trans. Math. Software* **20**, 447–459.

KEARFOTT, R. B. & NOVOA, M. 1990 Algorithm 681: INTBIS, a portable interval Newton/bisection package. *ACM Trans. Math. Software* **16**, 152–157.

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

MALLYA, J. U. & STADTHERR, M. A. 1997 A multifrontal approach for simulating equilibrium-stage processes on supercomputers. *Ind. Eng. Chem. Res.* **36**, 144–151.

MALLYA, J. U., ZITNEY, S. E., CHOUDHARY, S. & STADTHERR, M. A. 1997a A parallel frontal solver for large scale process simulation and optimization. *AIChE J.* **43**, 1032–1040.

MALLYA, J. U., ZITNEY, S. E., CHOUDHARY, S. & STADTHERR, M. A. 1997b A parallel block frontal solver for large scale process simulation: Reordering effects. *Comput. Chem Engng* **21**, S439–S444.

MCDONALD, C. M. & FLOUDAS, C. A. 1995a Global optimization for the phase stability problem. *AIChE J.* **41**, 1798–1814.

MCDONALD, C. M. & FLOUDAS, C. A. 1995b Global optimization for the phase and chemical equilibrium problem: Application to the NRTL equation. *Comput. Chem. Eng.* **19**, 1111–1139.

MCDONALD, C. M. & FLOUDAS, C. A. 1995c Global optimization and analysis for the Gibbs free energy function using the UNIFAC, Wilson, and ASOG equations. *Ind. Eng. Chem. Res.* **34**, 1674–1687.

MCDONALD, C. M. & FLOUDAS, C. A. 1997 GLOPEQ: A new computational tool for the phase and chemical equilibrium problem. *Comput. Chem. Eng.* **21**, 1–23.

MCKINNON, K. I. M., MILLAR, C. G., & MONGEAU, M. 1996 Global optimization for the chemical and phase equilibrium problem using interval analysis. In *State of the Art in Global Optimization: Computational Methods and Applications*, eds. C. A. Floudas and P. M. Pardalos, Kluwer Academic Publishers, Dordrecht, Netherlands.

MICHELSEN, M. L. 1982 The isothermal flash problem. Part I: Stability. *Fluid Phase Equilibria* **9**, 1–19.

MOORE, R. E. 1966 *Interval Analysis.* Prentice-Hall, Englewood Cliffs, NJ (1966).

NAGARAJAN, N. R., CULLICK, A. S., & GRIEWANK, A. 1991 New strategy for phase equilibrium and critical point calculations by thermodynamic energy analysis. Part I. Stability analysis and flash. *Fluid Phase Equilibria* **62**, 191–210.

NEUMAIER, A. 1990 *Interval Methods for Systems of Equations.* Cambridge University Press, Cambridge, England.

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

REID, R. C., PRAUSNITZ, J. M., & POLING, B. E. 1987 *The Properties of Gases and Liquids.* McGraw-Hill, New York, NY.

RUMP, S. M. 1988 Algorithm for verified inclusions—theory and practice. In *Reliability in Computing*, ed. R. E. Moore, Academic Press, 109–126.

SCHNEPPER, C. A. & STADTHERR, M. A. 1990 On using parallel processing techniques in chemical process design. Presented at AIChE Annual Meeting, Chicago, IL.

SCHNEPPER, C. A. & STADTHERR, M. A. 1996 Robust process simulation using interval methods. *Comput. Chem. Eng.* **20**, 187–199.

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

STADTHERR, M. A., COFER, H. N. & CAMARDA, K. V. 1993 Computing in 2001—Hardware: Developing the metacomputer. *Chem. Eng. Prog.* **89**(11), 27–37.

STADTHERR, M. A., SCHNEPPER, C. A., & BRENNECKE, J. F. 1995 Robust phase stability analysis using interval methods. *AIChE Symp. Ser.* **91**(304), 356–359.

Sun, A. C. & Seider, W. D. 1995 Homotopy-continuation method for stability analysis in the global minimization of the Gibbs free energy. *Fluid Phase Equilibria* **103**, 213–249.

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

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

Wasylkiewicz, S. K., Sridhar, L. N., Malone, M. F., & Doherty, M. F. 1996 Global stability analysis and calculation of liquid-liquid equilibrium in multicomponent mixtures. *Ind. Eng. Chem. Res.* **35**, 1395–1408.

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

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

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

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

Zitney, S. E., Camarda, K. V. & Stadtherr, M. A. 1994 Impact of supercomputing in simulation and optimization of process operations. In *Proc. Second International Conference on Foundations of Computer-Aided Process Operations* (eds., D. W. T. Rippin, J. C. Hale & J. F. Davis), pp. 463-468. CACHE Corp., Austin, TX.

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

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