Analysis of Parallelism in Modular Flowsheet Calculations

An analysis of parallelism for modular process flowsheet calculations is given. The central idea is to examine various algorithms that allow one to decouple and decentralize flowsheet calculations for computation in a multiprocessor environment. Specific algorithms are evaluated which make use of quasi-Newton updates to converge the flowsheet equations. From the analysis presented and example problems solved, it is clear that parallel processing can be used to enhance the computational speed of modular flowsheet calculations. In particular, various formulations of the simultaneous modular approach are analyzed to show how parallelism may be taken advantage of in flowsheet solutions.

E. H. Chimowitz, R. Z. Bielinis
Department of Chemical Engineering
University of Rochester
Rochester, NY 14627

Introduction

The development of parallel processing algorithms has been the subject of intensive research in the computer sciences for at least the past decade. Put simply, the concept behind parallel processing is that of doing more than one thing at a time. This means breaking down a computational task into independent segments that can be executed simultaneously, i.e., in parallel. If this can be done, the computational task is said to contain inherent parallelism. In contrast to parallelism is the concept of sequential processing, where a computational task is done by breaking it down into segments that are solved serially. Here the output from one segment is used as input for another. Sequential processing concepts are widely used in current hardware/software computing systems; however, more advanced computer designs are expected to take advantage of the possibilities offered by parallel processing ideas (Lord et al., 1984), and prototype parallel machines like the BUTTERFLY are currently in use (Brown et al., 1985).

One of the major advantages of parallel processors over sequential processors is their greatly increased speed. Execution speed enhancement in parallel using N processors, can be of order N compared to the same task done on a serial processor (Heller, 1978). Heller (1978) and Schendel (1984) provide reviews of various aspects of numerical computing on parallel computers.

The purpose of this work is to illustrate possible advantages of parallelism in the modular process flowsheeting problem. The underlying numerical strategy is similar to the simultaneous modular flowsheeting approach discussed by Jirapongphan (1980), Mahalec et al. (1979), Biegler (1983), Chen and Stadtherr (1985a, b, c); however, these previous studies have not addressed the issue of parallelism as it arises in simultaneous modular flowsheeting, yet it is clear that some formulations of this flowsheet strategy are well suited for parallel computers.

Flowsheets and Parallelism

The process units in a chemical process flowsheet are the various operations (e.g., chemical reactors, distillation columns) required to process raw feed materials into final products. The manner in which these units are interlinked is generally quite complex and the major purpose of process flowsheet programs is to provide a solution to the equations describing the steady state (or dynamic) behavior of the process. The standard computational structure for solving flowsheeting problems is sequential (these are called sequential modular systems) and their use is widespread throughout the chemical/petroleum industry; calculations are done serially, the output results from one unit being used as input for the succeeding one. Other well-known approaches that have been proposed for solving the flowsheet equations are the simultaneous-modular and equation-oriented methods. Rosen (1980), Biegler (1983), Shacham et al. (1982), and Chen and Stadtherr (1985a) discuss these strategies and their relative merits.

The sequential modular strategy employs a classic serial approach to the problem; the flowsheet is partitioned and a tear set found such that once the tear streams are torn, all the remaining streams related to this group can be computed in a serial manner. The serial information flow in this traditional approach precludes much use of parallelism. The equation-

Correspondence concerning this paper should be addressed to E. H. Chimowitz.
The current address of R. Z. Bielinis is Alfa-Laval Inc., North Chili, NY 14514.

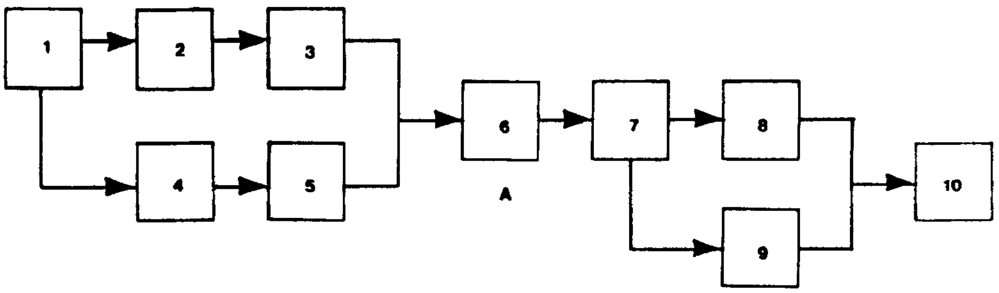

Figure 1a. Flowsheet structure without recycles.

based approach, while offering the potential for speed and flexibility, has acquired a poor reputation for reliability (Chen and Stadtherr, 1985a). In addition, from a parallel processing viewpoint the large storage requirements and complexity of the computing routines needed by this approach (such as the sparse matrix routines) will require considerable software engineering to implement. The simultaneous modular approach on the other hand combines some of the better features of these other methods. The Jacobian matrix (or an approximation of it) of the flowsheet equations will be smaller than that required by the equation-based approach. Furthermore, already existing modules can be used to perform the module-level calculations, requiring little additional investment in software. Finally, the simultaneous modular formulation can be shown to accommodate parallelism in a straightforward manner, making it an attractive strategy for implementation on a multiprocessor computer system. This strategy forms the basis of the approach taken in this work.

Preliminary Concepts

Figure 1a illustrates the idea of parallelism in a simple acyclic process flowsheet. Each block (subtask) represents a process unit (e.g., distillation column, reactor). Figure 1b shows time units associated with the calculations for each block. The latent parallelism in Figure 1a is easily seen. Given two processors, segments 2-3, 4-5, and 8-9 can be done in parallel. If only one processor were available, the total time units required to accomplish the overall computing task would be the sum of times for each subtask, equal to 25 units. If the above-mentioned segments are executed in parallel on two processors, the overall task can be accomplished in 19 units, which is the time required by the slowest path through the network (1-2-3-6-7-9-10).

B

BLOCK	TIME
1	2
2	3
3	1
4	2
5	1
6	3
7	4
8	3
9	4
10	2
	25

Figure 1b. Time associated with calculations of flowsheet in Figure 1a.

The structure shown in Figure 1a represents probably the simplest form of parallelism that may be encountered in flowsheet structures. A prevalent feature of chemical process flowsheets is the presence of recycle streams. In general, recycle streams arise because of a need to recycle unreacted material from a chemical reactor for further reaction, or because of energy recovery within the process which requires the matching of hot and cold process streams (Linnhoff et al., 1983). Recycle streams couple units downstream with those upstream, complicating the usual notion of parallelism, which requires independent subtasks for computation on separate processors. One conceptual problem then becomes how to "find" parallelism in a recycle structure, where unit calculations are clearly coupled.

Parallelism with a Modular Flowsheet Description

The flowsheet corresponding to Cavet's (1963) test problem will be used to illustrate concepts. The flowsheet is shown in Figure 2 and has a fairly complicated recycle structure. Streams *S1* and *R1* are a tear set for this problem (Upadhye and Grens, 1975; Chen and Stadtherr, 1985b), belonging to the same irredundant family as the tear set *R4*, *R2*, *R1*. With *S1* and *R1* as tear streams, a traditional sequential modular algorithm would solve this flowsheet by converging the tear stream variables using direct substitution in combination with an acceleration technique, for example the method of Wegstein (1958). There is a limited degree of obvious parallelism to these calculations. The calculations on unit *F2* proceed first on processor 1. The results from *F2* allow the calculations in *F1* and *A2* to be started simultaneously on separate processors (1 and 2). After this step though, the calculations in unit *A1* require the output from *F3* before they can proceed. If units *F1* and *A2* take the same amount of time to calculate, this would imply that the processor devoted to the *S1* loop would remain idle until after *F3* was calculated. Idle time usually has an adverse effect on parallel algorithm efficiency.

For the purpose of clarity, assume that each unit takes on the order of τ time units to calculate. With the *S1*, *R1* tear set, a traditional serial algorithm on a single processor would take 6τ time units for each iteration. However, using two processors and the parallel scheme described previously, the time for each iteration is 4τ, an improvement of 33%, since the units *F1*, *A2* and *A1*, *F4* can be done in parallel. If, however, *A2* dominated the calculations, say taking 10τ, then the traditional single-proces-

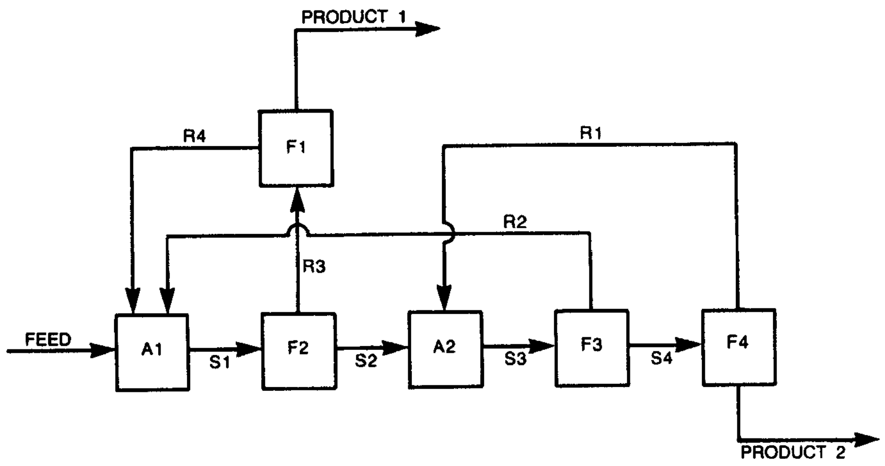

Figure 2. Cavett's problem.

sor approach would take 15τ while the two-processor scheme takes 13τ, an improvement of only 13%. This shows how a dominant unit calculation can lead to increases in processor idle time and a deterioration in parallel algorithm performance and efficiency.

Consider these calculations with the tear set *S1*, *R1* and two processors with the *F1* calculation increased to say 3τ (the other units remaining the same). The parallel scheme still takes 13τ between iterations. If however, *F1* remained at τ and *F3* were increased to 3τ (all others remaining the same), the parallel scheme would take 15τ between iterations, i.e., the increase in *F3* lies along the slow path *S2*, *S3*, *R2*, *S1* connecting the recycles *R3*, *R4*, *S1* and *S3*, *S4*, *R1*, and this increase is taken up by the algorithm. Thus parallel schemes can be quite sensitive to where the increased computing load occurs within the structure. The simple flowsheet structure shown in Figure 3 illustrates a situation where the sequential modular scheme is restricted to pure serial processing. If say $\underline{x}$ is the tear stream, during each iteration the flowsheet units would be calculated successively, prior to an acceleration step. There is no parallelism to this algorithm, no way in which multiple processors can be used to advantage.

This discussion shows how even in simple flowsheets the conventional sequential modular approach can be quite resistant to the advantages afforded by parallelism. In modular flowsheets our objective is to devise parallel algorithms whose speed per iteration is largely determined by τ_{max}, the dominant unit computational time, regardless of the flowsheet structure.

Now consider the simultaneous modular approach. Chen and Stadtherr (1985a) give an excellent discussion of three basic formulations of this strategy which we briefly review. In formulation I all connecting streams in the flowsheet are torn and treated as two separate streams, one input and one output, resulting in an unnecessarily large system of nonlinear equations to be solved. In formulation II, each connecting stream is treated as one input stream and the model equations are substituted into the stream connection equations, resulting in a large reduction in the number of flowsheet-level equations compared to formulation I. In order to further reduce the number of equations to be solved simultaneously, and thereby allow the use of full matrix methods for the system solution, an appropriate set of connecting streams is torn, leading to the formulation III, which was the basis for the work of Chen and Stadtherr (1985a, b ,c). Calculation of the Jacobian matrix for formulation III is a central issue. Direct difference approximation is straighforward but requires modules within each loop to be repeatedly calculated in a manner analogous to the sequential modular approach, with all the attendant disadvantages as far as parallelism is concerned. As the chemical process to be simulated becomes larger or more complicated, the number of tear streams will generally increase, exacerbating the problem associated with this approach. These considerations led Chen and Stadtherr to the conclusion that Jacobian evaluation should be done using block perturbation methods usually associated with formulations I and II. It is worth considering this in more detail.

Using the description of Chen and Stadtherr (1985a), mathematically the flowsheet-level equations for formulation II can be partitioned as:

$$f(\underline{p}, \underline{q}) = 0 \tag{1}$$

$$g(\underline{p}, \underline{q}) = 0 \tag{2}$$

Here Eq. 1 represents the nontear connecting stream equations and Eq. 2 represents the tear stream connection equations and design specifications. The vector $\underline{q}$ represents tear stream variables and free design variables and $\underline{p}$ nontear stream variables. Formulation III can be obtained from these equations by using Eq. 1 to determine $\underline{p}$ as a function of $\underline{q}$, reducing the above equations to: $g(\underline{p}, \underline{q}) = \overline{(g(\underline{p}(\underline{q}), \underline{q})} = \overline{g}(\underline{q})$.

Euler's identity allows the Jacobian for formulation III to be obtained from:

$$\left[\frac{dg}{d\underline{q}}\right] = \left[\frac{\partial g}{\partial \underline{q}}\right] - \left[\frac{\partial g}{\partial \underline{p}}\right]\left[\frac{\partial f}{\partial \underline{p}}\right]^{-1} + \left[\frac{\partial f}{\partial \underline{q}}\right] \tag{3}$$

The quantities in Eq. 3 are those obtained in the normal course of things for formulation II; however, the matrix manipulations involved in Eq. 2 at each iteration could prove to be expensive for large flowsheets with large stream-variable vectors, notwithstanding the fact that the final Jacobian is smaller in size.

More importantly from a parallel processing perspective is the subtle restriction imposed by the implicit serial information flow associated with formulation III used by Chen and Stadtherr. Consider Eq. 3 and the resultant flowsheet solution procedure. The various subsystem Jacobians represented by the terms $\partial f/\partial \underline{q}$, $\partial f/\partial \underline{p}$, $\partial g/\partial \underline{p}$, $\partial g/\partial \underline{q}$ can be calculated in parallel if the block perturbation approach is used. (Relative to the direct difference scheme, the block perturbation approach is advantageous from a parallel algorithm viewpoint, since the individual block calculations can be routed to separate processors working in parallel.) After these subsystem calculations, the reduced flowsheet Jacobian can be calculated using Eq. 3. However, this updated Jacobian allows only the tear stream variables to be directly updated. Prior to the next iteration and reevaluation of the flowsheet subsystem Jacobians, nontear stream variables have to be updated. This requires a complete serial pass through the flowsheet, in a manner entirely analogous to a sequential modular iteration. Hence, at each iteration the simultaneous modular strategy corresponding to formulation III is restricted from exploiting parallelism in a way similar to the sequential modular approach described earlier.

In line with this discussion, the parallel strategy we are proposing seeks to maximize the use of parallelism in modular flowsheet calculations. The problem formulation consists of a modular flowsheet description similar to that solved by conventional sequential modular or simultaneous modular approaches, i.e., where existing special-purpose modules are used to solve individual flowsheet subsystem models. Each module is assigned a separate processor, a reasonable assumption given that a current parallel computer like the BUTTERFLY has access to 256 coprocessors and clearly this number will increase as this technology advances. The flowsheet-level equations that result after each parallel step are similar in structure to formulation II of the simultaneous modular approach described by Chen and Stadtherr (1985a). Although this leads to a larger tear set (Jacobian) than formulation III, it exploits parallelism to a greater extent. This trade-off is discussed in a later section. Another important difference is that the flowsheet Jacobian or subsystem Jacobians are approximated directly using quasi-Newton updates like the method of Broyden (1965) or that of Schubert (1970). This eliminates the need to compute these various matrices by perturbation of unit modules, as required by the full block perturbation technique used by Chen and Stadtherr. Our proposed strategy is intended to lead to algorithms whose computation time per iteration is largely limited by τ_{max}, the dominant unit calculation time in the flowsheet. It is likely to be most advantageous for flowsheets comprising many computationally intensive module calculations. Without "penetrating" and changing the code in modules themselves, τ_{max} represents a lower bound per interation for any parallel scheme applied to modular flowsheets. Our proposed parallel strategy is then of the following general form.

Algorithm 1 (parallel processing)—General form

1. Partition flowsheet subsystems (blocks) connected by forward and/or backward (recycle) streams.
2. Tear connecting streams and initialize connecting stream variables and system Jacobian matrix or its inverse.
3. Calculate all blocks simultaneously on a multiprocessor system producing new estimates of connecting stream variables.
4. Update all connecting stream variable values simultaneously using the Newton-type iteration or a variation thereof. Update system Jacobian (or its inverse) using a quasi-Newton formula. For small problems the system Jacobian (or its inverse) can be approximated directly by a full matrix update. For large problems either the system Jacobian could be approximated directly using a sparse update like Schubert's method, or else the various subsystem Jacobians could be directly found using a quasi-Newton update.
5. Check for convergence; if so, print final results and stop, otherwise return to step 3 using as input to the blocks the values obtained in step 4.

For the flowsheet in Figure 3, with partitioning along the dotted lines, the equations to be solved are Eqs. 4, 5, and 6 cast in the following form:

$$\underline{f}_1 = \underline{z} - \underline{g}(\underline{y}) = 0 \tag{4}$$

$$\underline{f}_2 = \underline{x} - \underline{h}(\underline{z}) = 0 \tag{5}$$

$$\underline{f}_3 = \underline{y} - \underline{f}(\underline{x}) = 0 \tag{6}$$

A schematic of algorithm 1 using a quasi-Newton method to directly converge the connecting stream variables is now given.

Algorithm 2, for flowsheet in Figure 3

1. Partition flowsheet from Figure 3 into three blocks as indicated by vertical broken lines in the figure.
2. Set iteration counter $K = 1$. Initialize connecting stream variables $\underline{x}^{(K)}$, $\underline{y}^{(K)}$, $\underline{z}^{(K)}$ and Jacobian matrix inverse to $H^{(K)}$ (often $H^{(1)} = I$).
3. Evaluate blocks in parallel, $\underline{f}^{(K)}$, $\underline{g}^{(K)}$, $\underline{h}^{(K)}$ and obtain values for $\underline{f}_1^{(K)}, \underline{f}_2^{(K)}, \underline{f}_3^{(K)}$.
4. Estimate new values $\underline{x}^{(K+1)}$, $\underline{y}^{(K+1)}$, $\underline{z}^{(K+1)}$ using unit step Newton iteration formula:

$$\begin{pmatrix} \underline{z}^{(K+1)} \\ \underline{x}^{(K+1)} \\ \underline{y}^{(K+1)} \end{pmatrix} = \begin{pmatrix} \underline{z}^{(K)} \\ \underline{x}^{(K)} \\ \underline{y}^{(K)} \end{pmatrix} - (H^{(K)}) \begin{pmatrix} \underline{f}_1^{(K)} \\ \underline{f}_2^{(K)} \\ \underline{f}_3^{(K)} \end{pmatrix}$$

5. Check for convergence of $\underline{x}^{(K+1)}$, $\underline{y}^{(K+1)}$, $\underline{z}^{(K+1)}$. If converged, print final results and stop, otherwise proceed to step 6.
6. Evaluate blocks in parallel, $\underline{f}$, $\underline{g}$, $\underline{h}$ and obtain values for $\underline{f}_1^{(K+1)}, \underline{f}_2^{(K+1)}, \underline{f}_3^{(K+1)}$.
7. Update estimate of inverse Jacobian matrix using quasi-Newton formula (i.e., Broyden 1965) $H^{(K+1)} \leftarrow H^{(K)}$.
8. Increment iteration counter and proceed to step 4.

Basis for Algorithm Time Comparisons

The processing time for a simulation using the sequential modular technique on a serial processor can be written as:

$$T_{sm} = \left(I_{sm} \sum_{i=1}^{N} \tau_i \right) + \tau_o \tag{7}$$

where:

I_{sm} = Number of sequential modular iterations

τ_i = Processing time requirement of the ith unit operation (which we assume remains constant from iteration to iteration, generally a reasonable assumption)

N = Number of unit operations in the convergence path

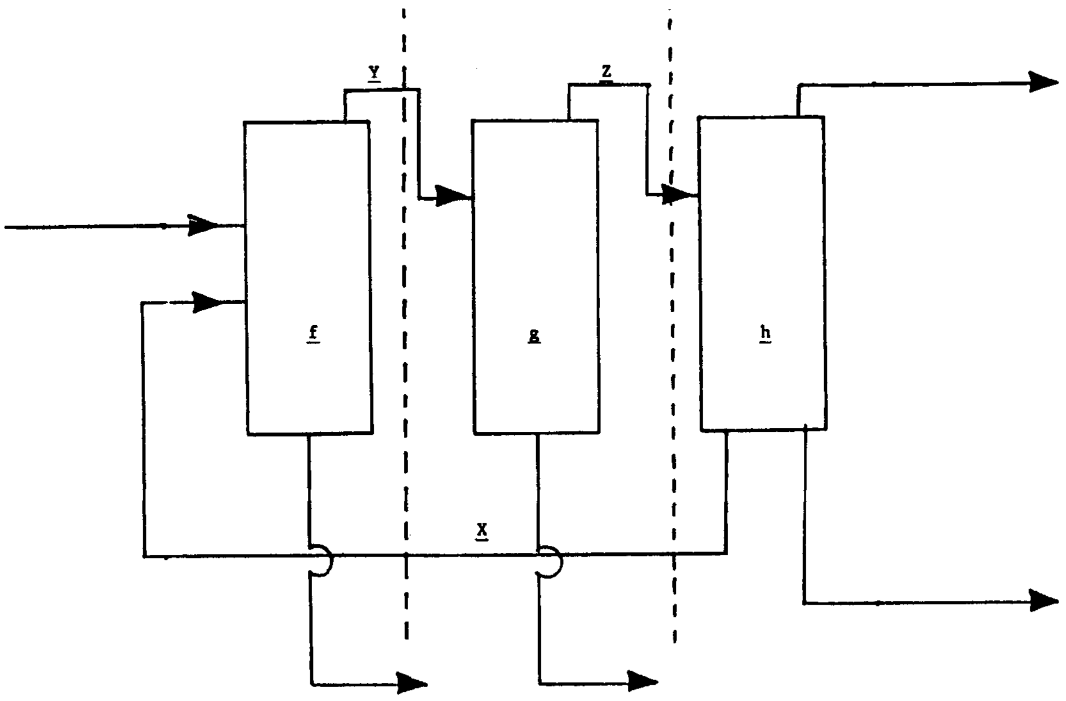

Figure 3. Simple process flowsheet.

τ_o = Overhead processing time (reading files, calling routines, etc.)

For the purposes of this study, overhead processing time will be neglected since the unit operation simulations dominate the computational load. Therefore, for the sequential modular technique, processing time can be written as

$$T_{sm} = I_{sm} \sum_{i=1}^{N} \tau_i \tag{8}$$

When applying an acceleration technique to the sequential modular approach the amount of processing time required to carry out the acceleration step must be included. Thus the processing time required by a sequential modular simulator using acceleration after every iteration can be written as

$$T_{asm} = I_{asm} \left[\sum_{i=1}^{N} (\tau_i) + \tau_a \right] \tag{9}$$

where τ_a is the amount of time required to execute the acceleration step. In many instances the acceleration is not carried out after each iteration, and Eq. 9 must be modified to account for this. If τ_a is small with respect to the term $\Sigma_{i=1}^{N} \tau_i$ then Eq. 9 takes the form of Eq. 8. This will be the case when the number of tear variables is relatively small or when time-consuming unit operation simulations dominate the flowsheet.

The processing time required for the general parallel algorithms described earlier when executed on a multiprocessor machine is somewhat more difficult to express, depending mainly upon how the Jacobian is evaluated. First consider the case where all derivatives of the flowsheet subsystems are calculated using a finite-difference technique, equivalent to the block perturbation method. A flowsheet with N subsystems and m independent connecting variables per stream requires $N(m + 1)$ unit evaluations in order to calculate the Jacobian. If the computational load is distributed over N processors, the Jacobian calculation time can be written as

$$T_{J,fd} = (m + 1)\tau_{i,max} \tag{10}$$

where $\tau_{i,max}$ is the dominant unit computational time. Using this relationship, along with the fact that the parallel algorithm allows unit operations to be distributed over multiple processors, leads to the following expression for processing time for the parallel algorithm using finite differences for Jacobian approximation:

$$T_{p,fd} = I_{p,fd}[(m + 1)\tau_{i,max})] \tag{11}$$

where $I_{p,fd}$ is the number of iterations required by the parallel algorithm. It should be noted that if Nm processors were available, the Jacobian calculation time could be decreased to $2\tau_{i,max}$. However, Nm can be a significantly large number of processors to schedule. One of the advantages of using quasi-Newton methods for Jacobian evaluation is that only N processors are required to keep Jacobian evaluation time of order $\tau_{i,max}$ if the update time is small by comparison. This should be evident from the following argument.

Consider the processing time required by the parallel algorithm using N processors with quasi-Newton methods for derivative estimation. Neglecting overhead, processing time can be expressed as:

$$T_{p,QN} = I_{p,QN}[\tau_{i,max} + \tau_u] + \tau_{Jo}$$

where τ_u is the time required to update the tear stream vectors and the Jacobian or its inverse with the quasi-Newton update, and τ_{Jo} is the time required to initialize the Jacobian. If the initial Jacobian estimate is made using finite differences,

$$T_{p,QN} = I_{p,QN}[\tau_{i,max} + \tau_u] + (m+1)\tau_{i,max} \tag{12}$$

If however, the initial Jacobian is set equal to the identity matrix, τ_{J_0} can be neglected and the processing time becomes

$$T_{p,QN} = I'_{p,QN}[\tau_{i,max} + \tau_u] \tag{13}$$

where $I'_{p,QN}$ and $I_{p,max}$ will in general be different. It is clear that rigorous estimation of the initial Jacobian may be a stiff penalty, and it may be advantageous to avoid such a calculation. The time required to update the system Jacobian is included, since the update is executed after the unit module evaluations are completed and therefore cannot be executed in parallel with the unit operation simulations. It is possible that large values of τ_u could lead to significant algorithm inefficiency while small values will not have an adverse impact, something that will be elaborated on later in the context of column-dominated flowsheets.

In the above relationships we see that the iteration numbers and the relationship of $\tau_{i,max}$ to $\Sigma_{i=1}^{N} \tau_i$ are the most important parameters when comparing serial and parallel algorithm speed. Clearly these parameters will vary from flowsheet to flowsheet. It can be argued, however, that when the parallel algorithms use derivative information to converge multiple tear streams simultaneously, for flowsheets of complex topology I_p should be less than I_{sm}. Evidence supporting this argument can be seen from the results of Chen and Stadtherr (1985b). The terms that the iteration numbers multiply are obviously important. While it is true that $\Sigma_{i=1}^{N} \tau_i$ is greater than $\tau_{i,max}$, it is also true that for some flowsheets the difference may not be great. Thus the parallel approach may significantly enhance the execution speeds of some flowsheets but not others.

Having described the general parallel approach in previous sections, we now solve two problems that illustrate the advantages of parallelism in calculations. The parallel algorithms were simulated on a VAX 11/780 system and initial conditions and convergence tolerances on each respective problem were the same for each numerical method.

Absorber-Stripper Flowsheet

A flowsheet for separating a hydrocarbon stream into lean and heavy fractions is shown in Figure 4. It consists of two interconnected adiabatic flash columns, the first used for lean component recovery, the second for solvent regeneration. Pressures in both columns are specified and the objectives of the calculation are to compute product recoveries and make-up solvent flow rate. Since pressures are specified, the vectors $\underline{x}$ and $\underline{y}$ for this problem are defined as

$$\underline{x} = [F_{1,x}, F_{2,x}, F_{3,x}, F_{4,x}, F_{5,x}, T_x]$$

$$\underline{y} = [F_{1,y}, F_{2,y}, F_{3,y}, F_{4,y}, F_{5,y}, T_y]$$

where $F_{i,x}$ referes to the flow rate of component i from the stripper, and $F_{i,y}$ the flow rate of component i from the absorber; T_x and T_y refer to the temperatures in the respective units.

At steady rate the following set of equations must hold

$$\underline{x} - \underline{A}(\underline{y}) = 0 \tag{14}$$

$$\underline{y} - \underline{S}(\underline{x}) = 0 \tag{15}$$

where $\underline{A}(\underline{y})$ and $\underline{S}(\underline{x})$ are vector functions defining the flash units and are not analytic, but calcuated by subroutine modules. The Jacobian matrix of this equation set has the following structure:

$$J = \begin{bmatrix} 1 & 0 & \cdots & 0 & \left(\frac{\partial A_1}{\partial y_1}\right) & \left(\frac{\partial A_1}{\partial y_2}\right) & \cdots & \left(\frac{\partial A_1}{\partial y_6}\right) \\ 0 & 1 & \cdots & 0 & \left(\frac{\partial A_2}{\partial y_1}\right) & \left(\frac{\partial A_2}{\partial y_2}\right) & \cdots & \left(\frac{\partial A_2}{\partial y_6}\right) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & \left(\frac{\partial A_6}{\partial y_1}\right) & \left(\frac{\partial A_6}{\partial y_2}\right) & & \left(\frac{\partial A_6}{\partial y_6}\right) \\ \left(\frac{\partial S_1}{\partial x_1}\right) & \left(\frac{\partial S_1}{\partial x_2}\right) & \cdots & \left(\frac{\partial S_1}{\partial x_6}\right) & 1 & 0 & \cdots & 0 \\ \left(\frac{\partial S_2}{\partial x_1}\right) & \left(\frac{\partial S_2}{\partial x_2}\right) & \cdots & \left(\frac{\partial S_2}{\partial x_6}\right) & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \left(\frac{\partial S_6}{\partial x_1}\right) & \left(\frac{\partial S_6}{\partial x_2}\right) & \cdots & \left(\frac{\partial S_6}{\partial x_6}\right) & 0 & 0 & \cdots & 1 \end{bmatrix}$$

It is this Jacobian or its inverse that is required, and a number of numerical methods for approximating it have been tested.

For the parallel algorithms, both forward and back recycle connecting streams were torn and the unit calculations done in parallel. For the comparative serial calculation (equivalent to an accelerated sequential modular approach) the exit stream from the stripper was torn and converged. This flowsheet was solved for many different initial conditions on the $\underline{x}$ and $\underline{y}$ vectors. The final steady state results are shown in Table 1. The performance of the various algorithms is shown in Table 2 using two quasi-Newton updates (Broyden, 1965; Schubert, 1970) for the parallel methods and both a Broyden and a direct substitution update

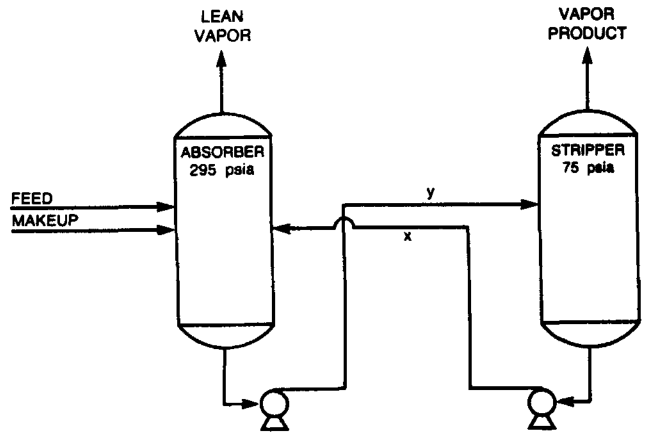

Figure 4. Absorber-stripper process.

Table 1. Absorber-Stripper Process Steady State

Stream	Temp. K	Press. kPa	Molar Flow Rate, kmol/h				
			F_{CH_4}	$F_{C_2H_6}$	$F_{C_3H_8}$	$F_{C_4H_{10}}$	$F_{C_7H_{16}}$
Feed	308	2,070	70.0	16.0	9.00	5.00	0.00
Make-up	298	2,070	0.00	0.00	0.00	0.00	2.52
Vapor product	343	517.0	4.31	2.52	2.20	1.52	0.880
Lean product	333	2,030	65.7	13.5	6.83	3.46	1.64
Stripper feed	334	2,030	4.87	3.70	5.00	6.72	48.3
Recycle	343	157.0	0.514	1.20	2.80	5.14	47.4

for the serial algorithms. These results are based upon 20 different initial conditions for all algorithms.

The reliability is expressed as the percentage of successful simulations and the mean number of iterations required to obtain the solution is also shown. It is clear that the sequential algorithms are the most reliable methods. Schubert's method showed a poor level of reliability, especially with the identity as the initial estimate for the Jacobian. The quasi-Newton methods require fewer iterations with an accurate Jacobian matrix initialized by finite differences. This is consistent with the results and discussion of Dennis and More (1977).

Cavett's Problem (1963)

This standard flowsheet test problem was used because it admits the possibility of using six processors in parallel and has an intricate recycle structure. Furthermore, this is a 16-component problem and leads to larger Jacobian matrices for the quasi-Newton updates. The flowsheet was discussed previously and is shown in Figure 2; final steady state conditions are shown in Table 3. It should be noted that mixing operations are modeled by adiabatic flashes, since this is a way to monitor the possibility of phase separation after mixing. This is done routinely in flowsheeting systems, where it is not always evident whether or not phase separation occurs.

The variables and equations modeling the flowsheet are given by

$$\underline{S}_1 = [F_{1,S1}, F_{2,S1}, \ldots, F_{16,S1}]$$

where $F_{i,S1}$ represents the flow rate of component i in stream $S1$. The other stream variables follow analogously, i.e.

$$\underline{R}_2 = [F_{1,R2}, F_{2,R2}, \ldots, F_{16,R2}]$$

Table 2. Absorber-Stripper Convergence Results

Method	J	J_o	Reliability, %	No. Iterations
Seq. Mod.	n.a.	n.a.	100	22
Seq. Mod. (B)	Broyden	I	100	16
Parallel	F.D.	F.D.	95	8
Parallel	Broyden	F.D.	75	12
Parallel	Broyden	I	80	18
Parallel	Schubert	F.D.	45	9
Parallel	Schubert	I	15	20

F.D., finite differences
Seq. Mod. (B), sequential modular with Broyden acceleration

The equations to be solved at steady state are

$$\underline{f}_1 = \underline{S}_1 - A1(\underline{R}_2, \underline{R}_4) = \underline{0} \quad \text{(set a, 16 equations)}$$
$$\underline{f}_2 = \underline{S}_2 - F2(\underline{S}_1) = \underline{0} \quad \text{(set b, 16 equations)}$$
$$\underline{f}_3 = \underline{S}_3 - A2(\underline{S}_2, \underline{R}_1) = \underline{0} \quad \text{(set c, 16 equations)}$$
$$\underline{f}_4 = \underline{S}_4 - F3(\underline{S}_3) = \underline{0} \quad \text{(set d, 16 equations)}$$
$$\underline{f}_5 = \underline{R}_1 - F4(\underline{S}_4) = \underline{0} \quad \text{(set e, 16 equations)}$$
$$\underline{f}_6 = \underline{R}_2 - F3(\underline{S}_3) = \underline{0} \quad \text{(set f, 16 equations)}$$
$$\underline{f}_7 = \underline{R}_3 - F2(\underline{S}_1) = \underline{0} \quad \text{(set g, 16 equations)}$$
$$\underline{f}_8 = \underline{R}_4 - F1(\underline{R}_3) = \underline{0} \quad \text{(set h, 16 equations)}$$

The Jacobian matrix of this system has the following structure:

$$\begin{array}{c|cccccccc|}
 & \underline{S}1 & \underline{S}2 & \underline{S}3 & \underline{S}4 & \underline{R}1 & \underline{R}2 & \underline{R}3 & \underline{R}4 \\
a & I & 0 & 0 & 0 & 0 & [\,] & 0 & [\,] \\
b & [\,] & I & 0 & 0 & 0 & 0 & 0 & 0 \\
c & 0 & [\,] & I & 0 & [\,] & 0 & 0 & 0 \\
d & 0 & 0 & [\,] & I & 0 & 0 & 0 & 0 \\
e & 0 & 0 & 0 & [\,] & I & 0 & 0 & 0 \\
f & 0 & 0 & [\,] & 0 & 0 & I & 0 & 0 \\
g & [\,] & 0 & 0 & 0 & 0 & 0 & I & 0 \\
h & 0 & 0 & 0 & 0 & 0 & 0 & [\,] & I
\end{array}$$

The blocks in parentheses in the above matrix are (16 × 16) submatrices representing Jacobian elements determined by unit operation functionality on inlet streams. Other entries are also (16 × 16) submatrices and are either the identity matrix or all zeros. The Jacobian is thus a (128 × 128) matrix. Once again the parallel algorithms require this Jacobian to be estimated, and a number of methods for doing so have been studied. Results for Cavett's problem are shown in Table 4.

Each algorithm was executed with five different initial estimates for tear variables. The sequential algorithms used the tear set {$R1$, $R2$, $R4$}. Using this set, Rosen and Pauls (1977) solved this problem using Wegstein acceleration after every four iterations. They reported that their program would not converge if acceleration was applied after each iteration. We also found that the Broyden sequential algorithm converged best if acceleration was not applied after each iteration. The results in Table 4 are for an acceleration after every fourth iteration. Here we see a high level of reliability for all parallel methods except when

Table 3. Cavett's Problem Steady State

S	Molar Flow Rate F kmol/h	Temp. T K	Press. P kPa	Mole Fractions C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
Feed	2,734	322	440	0.013	0.182	0.012	0.110	0.088	0.084	0.022	0.056	0.029	0.041	0.065	0.095	0.067	0.061	0.031	0.044
S1	40,330	318	1,961	0.010	0.224	0.019	0.097	0.119	0.133	0.027	0.061	0.025	0.034	0.047	0.066	0.046	0.042	0.020	0.030
S2	23,303	322	1,961	0.001	0.119	0.015	0.021	0.074	0.139	0.035	0.083	0.038	0.052	0.078	0.111	0.077	0.070	0.034	0.051
S3	25,546	303	440	0.001	0.124	0.016	0.021	0.079	0.154	0.037	0.087	0.037	0.051	0.073	0.103	0.072	0.065	0.032	0.047
S4	16,746	308	440	0.000	0.028	0.007	0.001	0.022	0.101	0.036	0.094	0.049	0.069	0.107	0.156	0.110	0.100	0.049	0.072
R1	1,843	307	191	0.000	0.187	0.034	0.012	0.138	0.346	0.065	0.129	0.033	0.030	0.016	0.008	0.002	0.000	0.000	0.000
R2	8,900	308	440	0.003	0.304	0.034	0.057	0.185	0.254	0.039	0.074	0.016	0.018	0.009	0.005	0.001	0.000	0.000	0.000
R3	16,526	322	1,961	0.023	0.375	0.024	0.206	0.184	0.125	0.015	0.028	0.006	0.007	0.004	0.002	0.001	0.000	0.000	0.000
R4	4,089	311	5,620	0.007	0.330	0.029	0.096	0.184	0.200	0.032	0.063	0.017	0.019	0.013	0.008	0.002	0.001	0.000	0.000
P1	12,438	311	5,620	0.029	0.390	0.022	0.242	0.184	0.100	0.010	0.016	0.003	0.003	0.001	0.000	0.000	0.000	0.000	0.000
P2	14,903	302	191	0.000	0.008	0.003	0.000	0.008	0.070	0.032	0.089	0.051	0.073	0.119	0.174	0.123	0.112	0.055	0.081

C1 Nitrogen; C2 Carbon Dioxide; C3 Hydrogen Sulfide; C4 Methane; C5 Ethane; C6 Propane; C7 Isobutane; C8 *n*-butane; C9 Isopentane; C10 *n*-pentane; C11 Hexane; C12 Heptane; C13 Octane; C14 Nonane; C15 Decane; C16 Undecane

using the Schubert method with the identity matrix as the initial Jacobian approximation. Note that good initial estimates of the Jacobian matrix significantly aid the speed of convergence of the quasi-Newton methods. Also note that the parallel method using finite-difference calculations for all derivative estimates represents a base case analysis, since no attempt was made to find an optimal search direction (i.e., line search, dogleg strategy, etc.). This, along with roundoff errors incurred during perturbation could account for the unusually large number of iterations required by this method.

Comparison of Serial and Parallel Algorithm Performance

In order to analyze the results, assumptions were made for each problem. The following assumptions were made when considering the absorber-stripper process:

1. Each unit operation requires the same amount of processing time.
2. The update times, τ_u and τ_a, are negligible.

The first assumption is easily justifiable since both units are characterized by the same type and number of modeling equations, and the same flash algorithm was used to solve them. The second assumption is justifiable when using the quasi-Newton updates since the Jacobians are small and relatively few matrix-vector operations are used for the updates.

For Cavett's problem however, assumption 1 above is clearly inappropriate. Because the process is modeled with two different types of unit operations (adiabatic and isothermal flashes), it is evident that all of the τ_i are not equal. Even so, a simplifying approximation must be made in order to estimate parallel algorithm efficiency. It was found from our calculations with Cavett's problem that the adiabatic flash units required on average four iterations of the isothermal flash algorithm. Since the process contains four isothermal flashes and two adiabatic flashes, each iteration of the sequential modular technique required the time equivalent of twelve isothermal flash calculations. Thus for Cavett's problem we assumed that $\Sigma\tau_i = 12\tau_I$ and $\tau_{i,max} = 4\tau_I$, where τ_I is the average processing time associated with one isothermal flash evaluation.

Estimating the processing time requirements of the Jacobian matrix updates is a more complex task. First, consider the processing time required to accelerate tear variables using the Broyden technique during a sequential modular simulation. The acceleration time, τ_a, is directly related to the number of variables being accelerated. For the three tear stream sequential modular approach to Cavett's problem, the Jacobian is a (48 × 48) matrix. Actual CPU time measurements indicated that τ_a for Cavett's problem was small relative to $\Sigma\tau_i$ (i.e., $\tau_a < 0.1\Sigma\tau_i$). Furthermore, the acceleration step was performed on every fourth iteration. Thus this term was neglected when algorithm processing time requirements were calculated. Next, consider the time required to update the Jacobian associated with the parallel algorithms. The magnitude of τ_u is also a function of the Jacobian size, which can in general be described as a matrix of dimension ($ab \times ab$) where a is the number of unspecified independent variables per tear stream and b is the number of tear streams in the process. An analysis of the quasi-Newton updates leads to the following expressions for the number of FLOPS (floating point operations) performed per update; see Bielinis (1986) for details:

$$\text{Broyden} \approx 9a^2b^2 + 2ab - 1 \tag{16}$$

$$\text{Schubert} \approx \frac{a^3b^3}{3} + \frac{11a^2b^2}{2} + \frac{7ab}{6} \tag{17}$$

The processing time requirement per update is proportional to the number of FLOPS performed and we can therefore see that

Table 4. Cavett's Problem Convergence Results

Method	J	J_ϕ	Reliability, %	No. Iterations
Seq. Mod.	n.a.	n.a.	100	50
Seq. Mod. (B)	Broyden	I	100	29
Parallel	F.D.	F.D.	100	30
Parallel	Broyden	F.D.	100	10
Parallel	Broyden	I	100	43
Parallel	Schubert	F.D.	100	8
Parallel	Schubert	I	20	14

F.D., finite differences
Seq. Mod. (B), sequential modular with Broyden acceleration

$\tau_{u,B}$ (update time for Broyden) increases with the square of a and b, while $\tau_{u,S}$ increases with the cube of a and b. It is instructive to examine the ratio

$$\frac{\tau_u}{\tau_{i,max}} = \alpha \tag{18}$$

to estimate the impact of the update on the parallel algorithms. A large value of α (i.e., greater than 1.0) would imply that the update requires a significant amount of the total processing time, while a small value (i.e., less than 0.1) would justify neglecting this term.

As mentioned earlier, τ_u was neglected in the analysis of the absorber-stripper process because of the small values of a and b. However, the values of a and b in Cavett's problem are 8 and 16, respectively. Furthermore, since $\tau_{i,max}$ is the time associated with only a single adiabatic flash calculation, the ratio given by α takes on a relatively large value. CPU time comparisons associated with Cavett's problem yielded the following average values of α:

$$\alpha_{\text{Broy}} \approx 0.250$$

$$\alpha_{\text{Schu}} \approx 18.0$$

In the light of this discussion Table 5 was developed. The results shown there illustrate processing time requirements of the various algorithms. The results on this problem indicate that significant speedup factors can be realized by implementing modular flowsheet calculations on multiprocessor computers. If one classifies the best serial algorithm as the Broyden accelerated sequential modular technique, and the best parallel algorithm to be the Broyden-based method, with J_o calculated using finite differences, the speed-up factors and efficiences for the two test problems are:

Problem	Speed-up Factor	Efficiency
Absorber-stripper	1.68	0.84
Cavett ($\tau_u = 0$)	3.22	0.54
Cavett ($\tau_u = \alpha\tau_{i,max}$)	2.95	0.49

Efficiency here is defined as speed-up factor/number of processors used (Schendel, 1984). It is interesting to note that Chen and Stadtherr (1985b) also solved Cavett's problem with their simultaneous modular algorithm. Their best results, reported in terms of equivalent sequential modular iterations indicated that their algorithm was almost exactly equivalent to the sequential modular results of Rosen and Pauls (1977) on this problem. However, Chen and Stadtherr's algorithm is well suited for parallelism since their subsystem Jacobians could be calculated in parallel, and viewed from that perspective, a parallel implementation of their algorithm would show a significant speed-up factor relative to the serial algorithm of Rosen and Pauls.

Estimates for ($\tau_u/\tau_{i,max}$) Column-Dominated Flowsheets

Consider a flowsheet where $\tau_{i,max}$ is the processing time required to simulate a distillation column using the well known

Table 5. Average Processing Time Requirements for Examples

Method	J	J_o	Absorber-Stripper	Cavett $\tau_U = 0$	Cavett[5] $\tau_U = \alpha\tau_{I,max}$
Seq. Mod.	n.a.	n.a.	$44\tau_A$	$600\tau_I$	n.a.
Seq. Mod. (B)[1]	Broyden	I	$32\tau_A$	$348\tau_I$	n.a.
Parallel[2]	F.D.	F.D.	$56\tau_A$	$2{,}040\tau_I$	n.a.
Parallel[3]	Broyden	F.D.	$19\tau_A$	$108\tau_I$	$118\tau_I$
Parallel[4]	Broyden	I	$18\tau_A$	$172\tau_I$	$283\tau_I$
Parallel[3]	Schubert	F.D.	$16\tau_A$	$100\tau_I$	$676\tau_I$
Parallel[4]	Schubert	I	$20\tau_A$	$56\tau_I$	$1{,}064\tau_I$

Seq. Mod. (B), sequential modular with Broyden acceleration
F.D., finite differences
τ_A, mean processing time of an adiabatic flash in absorber-stripper process
τ_I, mean processing time of an isothermal flash in Cavett's problem
Processing time calculated by:
1. Eq. 9
2. Eq. 11
3. Eq. 12
4. Eq. 13
5. $\tau_u = \alpha(4\tau_I)$

Naphtali-Sandholm (1969) technique. The Naphtali method requires the explicit solution of the MESH equations using the Newton-Raphson technique, and therefore the column Jacobian and resulting set of linear equations must be evaluated and solved iteratively. An order of magnitude analysis of the Naphtali algorithm (Bielinis, 1986) yields the following expression for the number of FLOPS per column solution:

$$\text{Naphtali} \approx N_I[{}^{46}\!/_3 NC^4 + {}^{320}\!/_3 NC^3 + {}^{84}\!/_3 NC^2 + {}^{32}\!/_3 NC - 20C^2 + {}^{4}\!/_3 N] \tag{19}$$

where N_I is the number of iterations required for the column solution, N is the number of stages in the column, and C is the number of components being distilled. Since processing time is proportional to the number of FLOPS performed, the following approximations for α can be written by assuming $a = c$ and neglecting insignificant terms in Eqs. 16, 17, and 19.

$$\alpha_{\text{Broy}} \approx \frac{9b^2}{N_I C^2\left[\dfrac{46}{3}N + \dfrac{320N}{3C}\right]} \tag{20}$$

$$\alpha_{\text{Schu}} \approx \frac{\dfrac{b^3}{3} + \dfrac{11b^2}{2C}}{N_I C\left[\dfrac{46}{3}N + \dfrac{320N}{3C}\right]} \tag{21}$$

From these relationships it is clear that α become significantly larger with inceasing number of streams. It is also clear that for large values of b, α_{Schu} is much larger than α_{Broy}. Because large values of α lead to significant delay times between iterations of the parallel algorithm, Schubert-based parallel algorithms become more inefficient for flowsheets with large numbers of tear streams. In order to better appreciate this, α for both updates was plotted against the parameter b in Figures 5 through 8. For all plots, N_I was assumed to be equal to 10, which is a reasonable average number for column calculations. Figures 5 and 6 illustrate the dependence of α_{Broy} and α_{Schu} on b for flowsheets com-

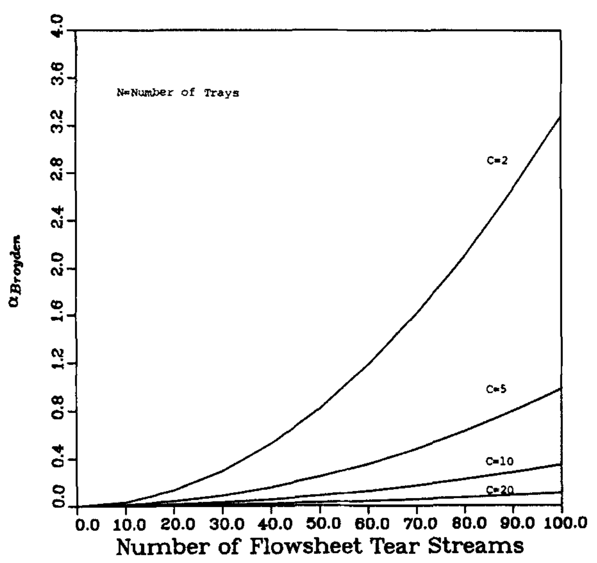

Figure 5. Performance index for Broyden's method in column-dominated flowsheet, $N = 10$.

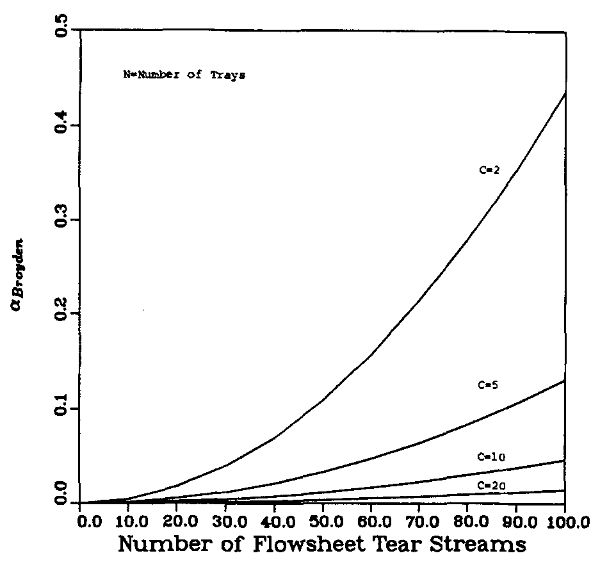

Figure 7. Performance index for Broyden's method in column-dominated flowsheet, $N = 75$.

putationally dominated by a 10-stage distillation column with various values of C. Figure 7 and 8 illustrate the same relationships when a 75-stage column dominates the flowsheet. Inspection of the plots indicates that the Broyden update time is almost always negligible with respect to the column simulation time. The only cases when the Broyden update time is significant are for large flowsheets dominated by small columns with a small number of components. The values of α_{Schu} indicate that the update time, $\tau_{u,S}$, can often be large with respect to the column simulation time and therefore can have a substantial detrimental effect on algorithm efficiency.

Of course, this previous analysis does not account for the fact that the update calculations could be done in parallel. In this sense the analysis represents a worst case scenario for the effect of τ_u on the parallel algorithm efficiency.

Conclusions

An analysis is presented showing how parallelism can benefit the implementation of modular flowsheeting strategies on multiprocessor computers. A particular strategy was proposed and studied whereby the flowsheet is partitioned into individual subsystems which are then routed to separate processors working in parallel. The results from these parallel computations are then used to converge the global flowsheet equations with the aid of quasi-Newton methods.

On the problems solved, the Broyden-based parallel algorithm with the initial Jacobian calculated by finite differences was the best method, considering both speed-up and reliability. In comparison to its serial counterpart the parallel Broyden method yielded a speed-up factor of 1.68 on a two-unit flowsheet, and approximately a factor of 3 on a six-unit flowsheet.

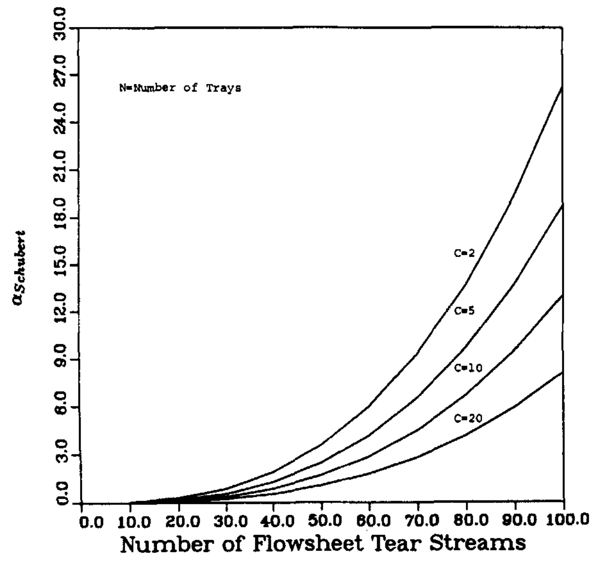

Figure 6. Performance index for Schubert's method in column-dominated flowsheet, $N = 10$.

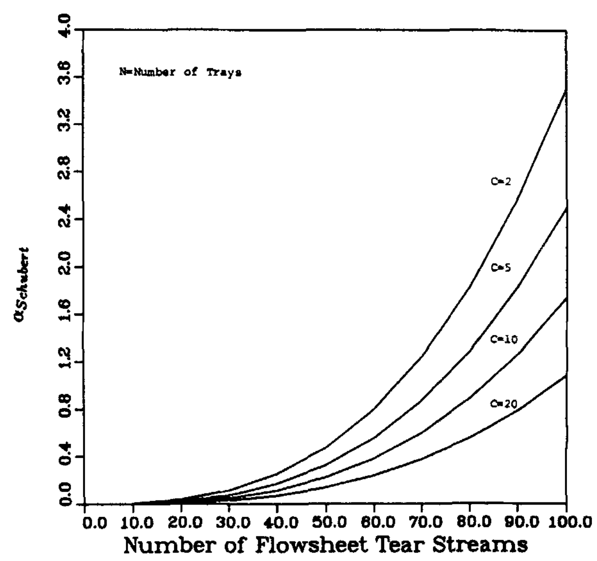

Figure 8. Performance index for Schubert's method in column-dominated flowsheet, $N = 75$.

With a processor dedicated to each unit module calculation, algorithm efficiencies were 84% and 50% on the example flowsheets, respectively. It is not necessary however to assign each unit operation an independent processor; the "fast" tasks could be executed on a single processor, while the "slow" tasks could be assigned independent processors. This would improve algorithm efficiency in asynchronous flowsheets, requiring an algorithm to schedule processor tasks for a given flowsheet. This is a standard problem in computer science.

It is conceivable that speed enhancements for a parallel implementation of the simultaneous modular approach will be an order of magnitude for realistic flowsheets, as can be seen from Eqs. 9 and 13 with reasonable choices for I_{asm}, $I_{p,QN}$, τ_i, $\tau_{i,max}$, and N. Obviously, parallel algorithms will be least advantageous for flowsheets where $\tau_{i,max}$ is significantly larger than $\Sigma_{i=1}^{N-1}\tau_i$ for the remaining flowsheet units.

Acknowledgment

The authors with to acknowledge support of this work by the National Science Foundation, Grant No. CPE-8404252.

Notation

a =	number of unspecified independent variables per tear stream
b =	number of tear streams in a process flowsheet
$\underline{f}$ =	unit module function calculated by a subroutine
$\underline{g}$ =	unit module function calculated by a subroutine
$\underline{h}$ =	unit module function calculated by a subroutine
m =	number of unspecified independent variables per connecting stream
$\underline{p}$ =	vector of nontear stream variables
$\underline{q}$ =	vector of tear stream variables and free design variables
$\underline{x}$ =	vector of connecting stream variables
$\underline{y}$ =	vector of connecting stream variables
$\underline{z}$ =	vector of connecting stream variables
A =	absorber model
C =	number of chemical components in a process model
$F_{i,j}$ =	molar flow rate of component i from unit j
H =	inverse Jacobian matrix
I =	identity matrix
I_{sm} =	number of sequential modular iterations required to simulate a flowsheet
$I_{p,fd}$ =	number of parallel iterations required for simulation when Jacobian is estimated using finite differences
$I_{p,QN}$ =	number of parallel iterations required for simulation when Jacobian is estimated using a quasi-Newton method after a finite-difference initialization
$I'_{p,QN}$ =	number of parallel iterations required for simulation when Jacobian is estimated using a quasi-Newton method after an identitly matrix initialization
J =	Jacobian matrix
K =	iteration counter
N =	number of unit operations in a convergence path
N_I =	number of iterations required to simulate a column using Naphtali technique
$\underline{R}_i$ =	vector of unspecified independent stream variables in recycle stream i of Cavett's problem
S =	stripper model
$\underline{S}_i$ =	vector of unspecified independent stream variables in connecting stream i of Cavett's problem
T_{asm} =	processing time required by an accelerated sequential modular algorithm
T_i =	temperature of stream i
T_{sm} =	processing time required by a sequential modular algorithm
$T_{J,fd}$ =	processing time required by multiple processors to estimate a Jacobian using finite differences
$T_{p,fd}$ =	processing time required by a parallel algorithm using finite differences for Jacobian estimation
$T_{p,QN}$ =	processing time required by a parallel algorithm using quasi-Newton methods to update the Jacobian

Greek letters

α =	parallel algorithm performance index
τ_a =	processing time of an acceleration step
τ_i =	processing time requirement of unit operation i
τ_o =	overhead processing time
τ_u =	processing time required to update flowsheet tear streams and system Jacobian or its inverse using a quasi-Newton method
$\tau_{i,max}$ =	processing time required to evaluate the dominant unit module in a flowsheet

Literature cited

Biegler, L. T., "Simulataneous Modular Simulation and Optimization," *Proc. 2nd Foundations Computer-Aided Process Design*, A. W. Westerberg and H. H. Chien, ed., AIChE, Snowmass, CO (1983).

Bielinis, R., "Chemical Process Simulation in a Parallel Environment," MS Thesis, Univ. Rochester, NY (1986).

Brown, C. M., C. S. Ellis, J. A. Feldman, T. J. LaBlanc, and G. I. Peterson, "Research with the BUTTERFLY Multicomputer." *Comput. Sci. Eng. Res. Rev.*, Univ. Rochester, NY (1985).

Broyden, C. G., "A Class of Methods for Solving Nonlinear Simultaneous Equations," *Math. Comp.*, **19**, 577 (1965).

Cavett, R. H., "Application of Numerical Methods to the Convergence of Simulated Processes Involving Recycle Loops," *Am. Pet. Inst., Repr. No. 04-63* (1963).

Chen, H. S., and M. A. Stadtherr, "A Simultaneous Modular Approach to Process Flowsheeting and Optimization. I: Theory and Implementation," *AIChE J.*, **31**, 1843 (1985a).

———, "A Simultaneous Modular Approach to Process Flowsheeting and Optimization. II: Performance on Simulation Problems," *AIChE J.*, **31**, 1857 (1985b).

———, "A Simultaneous Modular Approach to Process Flowsheeting and Optimization. III: Performance or Optimization Problems," *AIChE J.*, **31**, 1868 (1985c).

Dennis, J. R., Jr., and J. J. More, "Quasi-Newton Methods, Motivation and Theory," *SIAM Rev.*, **19**, 46 (1977).

Heller, D., "A Survey of Parallel Algorithms in Numerical Linear Algebra," *SIAM Rev.*, **20**, 740 (1978).

Jirapongphan, S., "Simultaneous Modular Convergence Concept in Process Flowsheet Optimization," Ph.D. Thesis, MIT (1980).

Linnhoff, B., H. Dunford, and R. Smith, "Heat Integration of Distillation Columns into Overall Processes," *Chem. Eng. Sci.*, **38**, 1175 (1983).

Lord, N., P. Giragosian, R. Quellette, and R. Clerman, *Advanced Computers: Parallel and Biochip Processors*, Butterworths, Woburn, MA (1984).

Mahalec, V., H. Kluzik, and L. B. Evans, "Simultaneous Modular Algorithms for Steady State Flowsheet Simulation and Design," *CACE 1979 Conf. Proc.*, Eur. Fed. Chem. Eng. (1979).

Naphtali, L. M., B. P. Sandholm, "Multicomponent Separation Calculations by Linearization," *AIChE J.*, **17**, 148 (1971).

Rosen, E. M., "Steady State Chemical Process Simulation: A State-of-the-Art Review," *Computer Applications to Chemical Engineering*, R. G. Squires and G. V. Reklaitis, eds., Am. Chem. Soc., Washington, DC, 3 (1980).

Rosen, C. M., and A. C. Pauls, "Computer-Aided Chemical Process Design: The FLOWTRAN System," *Comput. Chem. Eng.*, **1**, 11 (1977).

Schendel, U., *Introduction to Numerical Methods for Parallel Computers*, Wiley, New York (1984).

Schubert, L. K., "Modification of a Quasi-Newton Method for Nonlinear Equations with a Sparse Jacobian," *Math. Comp.*, **24**, 27 (1970).

Shacham, M. S., S. Macchietto, L. F. Stutzman, and P. Babcock, "Equation-Oriented Approach to Process Flowsheeting," *Comp. Chem. Eng.*, **6**, 79 (1982).

Wegstein, J. H., *Commun. Assoc. Comput. Mach.*, **1**, 9 (1958).

Upadhye, R. S., and E. A. Grens, "Selection of Decompositions for Process Simulation," *AIChE J.*, **21**, 136 (1975).

Manuscript received Feb. 3, 1986, and revision received Dec. 22, 1986.