Interval Computations
No 4, 1993

Application of a Parallel Interval Newton/Generalized Bisection Algorithm to Equation-Based Chemical Process Flowsheeting

Carol A. Schnepper and Mark A. Stadtherr

We describe here the application of a new parallel interval Newton/generalized bisection algorithm for solving the large, sparse, nonlinear algebraic equation systems arising in chemical process flowsheeting. The algorithm is based on the simultaneous application of root inclusion tests to multiple interval regions, and it is designed for implementation on MIMD computers with a combination of local and shared memory. The algorithm was tested successfully on several relatively small flowsheeting problems. The tests were performed using between 2 and 32 nodes of a BBN TC2000 parallel computer.

Применение параллельного интервального метода Ньютона с обобщенным делением пополам для решения уравнений в картах химико-технологических процессов

К. А. Шнеппер, М. А. Штадтхерр

Описывается применение нового параллельного интервального метода Ньютона с обобщенным делением пополам для решения больших разреженных систем нелинейных алгебраических уравнений, возникающих при составлении карт химико-технологических процессов. Данный алгоритм основан на одновременном применении критериев включения корня к нескольким интервальным областям. Он разработан для реализации на компьютерах типа MIMD, совмещающих локальную и распределенную память. Алгоритм был успешно протестирован на нескольких относительно небольших задачах картирования химических процессов с использованием от 2-х до 32-х узлов параллельного компьютера BBN TC2000.

1 Introduction and motivation

Computer-aided process simulation, design, and optimization are important tools in the design and control of manufacturing processes in the chemical and petroleum industries. The diagram of a chemical process, showing the units and the connections between them, depicts the flow of chemical components through the system and thus is often referred to as a process flowsheet. Especially in the chemical engineering literature, the simulation, design, or optimization problems associated with flowsheets are referred to as process flowsheeting problems [33].

In general, equation-based process simulation requires the solution of large, sparse, differential-algebraic equation (DAE) systems. These systems arise from the algebraic equations and the ordinary differential equations that model the process units, the connections between them, and the design specifications. If partial differential equations are included in the model, some type of spatial discretization is usually necessary, and the resulting equations are incorporated into the flowsheet system. Thus the general DAE system may be written

$$g(\dot{x}, x, t) = 0, \qquad t_0 \le t \le t_f \tag{1}$$

with a set of consistent initial conditions

$$x(t_0) = x_0. \tag{2}$$

Here, the nonlinear equations g and the time dependent variable set x are assumed to be sufficiently differentiable. In order to solve the system in equations (1) and (2), it is reduced to the form

$$f(x) = 0 \tag{3}$$

and this system of nonlinear algebraic equations f is solved at various time steps. The focus of this work is on the solution of the nonlinear algebraic system (3), so only steady-state simulation and design problems are considered here. However, this work is also relevant to more general flowsheeting problems since solution procedures for dynamic problems and optimization problems frequently include steps requiring the solution of nonlinear algebraic systems.

Most published work on general purpose equation-based flowsheeting has concentrated on using local methods, specifically Newton-like methods, to solve (3) [20]. Such algorithms are not always reliable unless the starting point is sufficiently close to a solution, and they are not designed for finding multiple roots, if they exist. The ability to locate multiple solutions is especially important since models involving complex nonlinearities are being incorporated into flowsheeting as new physical insights are gained. For example, models for exothermic reactors [17] and interlinked distillation towers [31] may give rise to multiple steady states and singularities, and even when simple models are used, multiple steady states may exist for a process as relatively uncomplicated as two-product distillation [10].

Various techniques for addressing the limitations inherent in Newton-like methods have been studied for application to equation-based flowsheeting. These approaches include trust region methods [2, 3], homotopy methods [17, 25, 31, 32], and methods based on imbedded linear programming problems [29]. Bisection algorithms based on interval analysis have not received serious consideration in this context because the number of variables involved in process flowsheeting makes such approaches infeasible on serial computers. However, by taking advantage of parallel computer architectures, interval bisection techniques can be made feasible, providing a globally convergent solution method capable of locating multiple roots.

In this paper, a parallel interval Newton/generalized bisection algorithm is developed and tested for application to equation-based flowsheeting. The problem now becomes the solution of

$$F(X) = 0 \tag{4}$$

where $F(X)$ is a suitable interval extension of the large, sparse system of nonlinear algebraic equations in (3). When efficiently implemented on a parallel computer, interval Newton/generalized bisection methods have characteristics which are especially beneficial for equation-based flowsheeting applications. First and foremost, these methods can be guaranteed to locate all real solutions, within a given region, to a system of nonlinear algebraic equations [11]. In addition, in process flowsheeting the variables represent physical quantities and phenomena, and intervals provide a natural way for engineers to define the space in which useful solutions may be located, excluding regions which are physically meaningless.

In this paper, real vectors and matrices are written as lower-case and upper-case letters, respectively, in regular typeface. Interval scalars are written as bold, lower-case letters, and interval vectors and matrices are represented with bold, upper-case letters. An N-dimensional interval vector is considered to represent an N-dimensional box in space.

2 Solution techniques and parallel algorithm

2.1 Interval nonlinear solution method

The method chosen here for solving (4) is based on the interval Newton/generalized bisection algorithm discussed in [12]. According to [11], a generalized bisection algorithm combines a geometrical bisection process with a root inclusion test. In this context, the interval Newton method serves as an existence and uniqueness test.

For an interval Newton method, the nonlinear algebraic system (4) is linearized to form

$$F'(X^k)(X^{k+1} - x^k) = -f(x^k) \tag{5}$$

where the superscript denotes an iteration counter, $F'(X^k)$ represents an appropriate interval extension of the Jacobian matrix of f evaluated for X^k, and x^k is the midpoint of the interval vector X^k.

The interval Gauss-Seidel linear solution technique was selected here to solve (5) for the updated interval vector X^{k+1}. Iterating according to (5) is most efficient for isolating the roots of (3) when the widths of the components of X^{k+1} are as small as possible, and the interval Gauss-Seidel linear solution technique is known to perform well in this regard [8, 19]. In most cases, the interval Gauss-Seidel method is also known to achieve smaller widths for X^{k+1} when (5) is first multiplied by a preconditioning matrix to produce

$$Y^k F'(X^k)(X^{k+1} - x^k) = -Y^k f(x^k). \tag{6}$$

Then, applying the interval Gauss-Seidel solver involves calculating the components of X^{k+1} on a row-by-row basis according to

$$x_i^{k+1} = x_i^k - \frac{b_i + \sum\limits_{\substack{j=1 \\ j \neq i}}^{n} M_{ij}(x_j^k - x_j^k)}{M_{ii}} \tag{7}$$

where $b = Y^k f(x^k)$ and $M = Y^k F'(X^k)$.

The interval Gauss-Seidel linear solver possesses properties which allow it to function as an existence and uniqueness test and to couple the interval Newton method to a bisection procedure. These desirable characteristics of the Gauss-Seidel iteration follow from a proof presented in [19] and have been studied by others, most notably [8]. The resulting interval Newton/generalized bisection technique is then capable of locating all real roots to (3). The critical properties of the Gauss-Seidel method and their application in the solution procedure are as follows. First, if $X^{k+1} \subseteq X^k$ after applying a Gauss-Seidel iteration, then (3) has a unique solution in X^{k+1}, and the traditional point Newton method starting from any point in X^{k+1} will converge to that solution. In this case, the result from applying (7) is considered to be a "true" response to an existence and uniqueness test, or root inclusion test. If, on the other hand, $x_i^{k+1} \cap x_i^k = \emptyset$ for any i, then there are no solutions to (3) in X^k, and the Gauss-Seidel method, acting as a root inclusion test, returns a value of "false". Finally, if the Gauss-Seidel method returns neither "true" nor "false," then the root inclusion test is inconclusive, and the procedure returns a value of "unknown." When the Gauss-Seidel method returns "unknown," either the value of X^k is set equal to that of X^{k+1} and another Gauss-Seidel iteration is performed, or one of the coordinate intervals of X^{k+1} is bisected to form two subboxes. The Gauss-Seidel iteration is subsequently applied to each of the two sub-boxes. In this manner, a bisection process enables an interval Newton method to isolate all of the solutions to (3) contained in the initial interval $X^{(0)}$.

2.2 Parallel algorithm

The key to designing efficient parallel algorithms is identifying independent tasks, having approximately equal computational requirements, for concurrent execution. For the parallel algorithm presented here, independent tasks at the subroutine programming level arise from application of the bisection process. In the serial implementation of an interval Newton/generalized bisection algorithm [12, 15], after a bisection step is completed, one of the sub-boxes is stored on a stack for later consideration while the program continues processing the other sub-box according to the interval Gauss-Seidel method. However, the calculations performed on any given sub-box are independent of the calculations performed on any other sub-box, and the boxes on the stack may be considered in any order. Therefore, on computers with the appropriate multiple-processor architecture, each available processor may execute the Gauss-Seidel method on a different box from the stack.

The following parallel algorithm, developed here for solving large, sparse flowsheeting systems, is based on the serial algorithm implemented in the INTBIS program for small, dense, polynomial systems[12, 15]. For the algorithm's initialization stage, the user must furnish an initial interval $X^{(0)}$ and values for the parameters ϵ and ϵ_F. The value of ϵ is heuristically chosen to represent the smallest allowable box dimension and is used to distinguish between roots that are close together. In the implementation here, roots that are closer than ϵ to each other may lie in the same small box. Similarly, the parameter ϵ_F is necessary for Step III (j), which provides a mechanism for the program to stop treating boxes near solutions at which the Jacobian matrix is singular. For the test problems described in Section 3, we chose $\epsilon = 10^{-5}$ and $\epsilon = 10^{-10}$. More information concerning the computational uses of ϵ and ϵ_F may be found in [12].

Algorithm

```
I. Initialization
    (a) Set $L = 1$. $L$ is the stack counter, indicating the number of boxes
        placed on the stack since the last execution of Step II (b(i)).
    (b) Set $X = X^{(0)}$.
    (c) Input values for $\epsilon$ and $\epsilon_F$.
    (d) Push $X$ onto the stack of boxes to be considered.
    (e) Set $STKPTR = 1$. $STKPTR$ is the stack pointer, indicating the
        location of the most recent box added to the stack.
II. Check stack. Executed on controlling processor.
    (a) If the stack is empty, then
        (i) If any other processors are active, then
            (1) Wait for one other processor to finish.
            (2) Return to Step II (a).
```

(ii) If no other processors are active, then go to Step IV.

(b) If the stack is not empty, then

(i) Set $LP = L$. LP represents the number of processors, at one per box, that are necessary for concurrently executing Step III on all L boxes.

(ii) Set $L = 0$.

(iii) Activate up to $LP - 1$ additional processors for executing Step III in parallel.

(iv) Execute Step III.

(v) Return to Step II (a).

III. Root inclusion test. Executed on all processors, one processor per box.

(a) If all parallel tasks generated at Step II (b(iii)) are complete, then remain idle until reactivated by the controlling processor at Step II (b(iii)).

(b) If any parallel tasks remain unfinished, then continue to Step III (c).

(c) Remove a box X from the stack.

(d) Set $STKPTR = STKPTR - 1$.

(e) Determine $d(X)$, the maximum value among the widths of the elements of X.

(f) If $d(X) < \epsilon/4$ and X has nonnull intersection with a box X' in the list of solution-containing boxes $\underline{L}$ such that $d(X) + d(X') \leq \epsilon/4$, then return to Step III (a). Otherwise, continue to Step III (g).

(g) Perform the Gauss-Seidel root inclusion test $T(X)$.

(h) If $T(X) =$ "unknown" and $d(X) \geq \epsilon/16$, then go to Step III (l).

(i) If $T(X) =$ "false", then return to Step III (a).

(j) If $T(X) =$ "true" or $|f(x)| < \epsilon_F$, then

(i) Store X in list $\underline{L}$.

(ii) If X has nonnull intersection with a box X' in list $\underline{L}$ such that $d(X) + d(X') \leq \epsilon/4$, then delete X' from list $\underline{L}$.

```
        (iii) Return to Step III (a).
    (k) If $T(X) =$ "unknown" and $d(X) < \epsilon/16$, then (adjust for roots
        on a boundary)
        (i) Replace $X$ by the box $X_b$ with the same midpoints as $X$ but
            four times as large.
        (ii) Delete from list $\underline{L}$ all $X' \in \underline{L}$ for which $X' \cap X_b \neq \emptyset$ and
             $d(X_b) + d(X') \leq \epsilon/4$.
        (iii) Store $X_b$ in list $\underline{L}$.
        (iv) Return to Step III (a).
    (l) Subdivide $X$ as follows:
        (i) Bisect $X$ to form sub-boxes $X_1$ and $X_2$.
        (ii) Store $X_1$ on the stack of boxes for later consideration.
        (iii) Set $L = L + 1$. Set $STKPTR = STKPTR + 1$.
        (iv) Store $X_2$ on the stack of boxes for later consideration.
        (v) Set $L = L + 1$. Set $STKPTR = STKPTR + 1$.
        (vi) Return to Step III (a).
IV. Final point solution approximation.
    (a) Execute traditional point Newton method for each box stored in
        solution-containing list $\underline{L}$.
    (b) Print as output the solution(s) and algorithm performance infor-
        mation.
```

In this implementation, a controlling processor executes Step II (b(iii)), which generates a parallel task indicating that Step III should be performed $LP - 1$ times, and that specific value of $LP - 1$ becomes associated with a particular parallel task. When fewer than $LP - 1$ processors are available, all of the available processors work on the task until they have executed the subroutine implementing Step III $LP - 1$ times. Step II (b(iii)) uses the value of $LP - 1$ rather than LP because the controlling processor itself, which is responsible for activating the other processors, executes Step III at Step II (b(iv)). The controlling processor may generate additional parallel tasks without first waiting for previously generated tasks to finish, thus activating as many processors as possible.

For this work, the preconditioning matrix Y^k in (6) was the inverse of the Jacobian matrix of f evaluated at the midpoint of the interval vector X^k. This preconditioner should be a sufficient approximation to the inverse of the matrix of midpoints of the elements of $F'(X^k)$, which the literature [19] frequently suggests as a good preconditioner. While other preconditioners for improving the performance of the Gauss-Seidel method are the subjects of current research [13, 16], they are even more computationally expensive than an inverse Jacobian matrix.

For selecting an appropriate coordinate interval for bisection, the "maximum smear" strategy suggested in [12] was used. This bisection approach attempts to determine the coordinate direction in which the gradient of the functions in the system (3) changes most rapidly. It accomplishes this by computing, for every component of X^{k+1},

$$s_j = (b_j - a_j) \cdot \max_i(|J_{i,j,1}|, |J_{i,j,2}|) \tag{8}$$

where a_j and b_j are the left and right endpoints, respectively, of the j-th component of X^{k+1} and where $[J_{i,j,1}, J_{i,j,2}]$ is the interval element located at row i and column j in the interval Jacobian matrix $F'(X^k)$. Then the coordinate interval j of X^{k+1} corresponding to the maximum s_j is bisected. For flowsheeting problems, this "maximum smear" scheme has been shown to be superior to both bisecting the widest coordinate interval and bisecting the widest coordinate interval after scaling with an infinity norm [23].

Techniques for efficiently storing and manipulating large, sparse matrices [6] were also incorporated into the program. For the solution of the large, sparse, linear systems of real equations required in computing Y^k, a two-pass approach, involving a reordering stage followed by a numerical stage, was implemented [9, 27, 28]. Sparse matrix techniques were also applied in the adaptation of the point Newton method [2, 3] required for Step IV (a).

2.3 Efficiency of parallel algorithm

The parallel algorithm described here is designed for a MIMD (Multiple-Instruction, Multiple-Data) computer, defined as a multiple-processor computer capable of simultaneously performing different sets of instructions on many different data streams [7]. For our algorithm, the computer should also support both globally shared memory, which any processor may access, and local memory, which is specific and private to each processor. For Step III, each processor is provided with its own private copy of all of the variables and arrays except for the stack of boxes awaiting the root inclusion test, the stack counter L, the stack pointer $STKPTR$, and the list of solution-containing boxes $\underline{L}$, which are stored in global memory. Each processor must be able to change the values of its own variables without influencing the other processors, but all of the processors must access the same stack and stack-related variables. Additionally, the list of boxes $\underline{L}$ must be shared among the processors so that all of the boxes in the list are available for comparison at Steps III (f), III (j(ii)), and III (k(ii)).

Several conditions must be met in order for the parallel algorithm to perform efficiently. First, several bisections are necessary for the algorithm to achieve its full speedup potential, where speedup is defined as the time it takes to complete a job on one processor divided by the time the job requires on P processors. Some problems require only one call to the root inclusion test subroutine, and while the parallel algorithm would successfully isolate their solutions, it would not offer any speedup over a sequential implementation since it would execute on only one processor. In addition, the algorithm assumes that several bisections are performed early in a program run. When the program executes Step III the first time, only one interval box is on the stack, and only the controlling processor is active. However, if every box results in a value of "unknown" after the first three passes through Step III, the stack will contain eight boxes, each of which may be tested on a different processor. On the other hand, if hundreds or thousands of processors are available, the startup time necessary to activate all of the processors would be unacceptable. In that case, the initialization procedure in Step I should also include a breadth-first partitioning of $X^{(0)}$ so that each processor could remove a box from the stack during the first pass through Step III. Finally, the binary tree of sub-boxes must grow in width as well as depth. A binary tree with depth-only growth would be produced if, after every interval bisection, one of the sub-boxes always resulted in a root inclusion test value of "true" or "false" while the other sub-box always resulted in a test value of "unknown." The algorithm would achieve a maximum speedup of two on this type of problem, regardless of the number of available processors. In contrast, a more balanced binary tree would activate many processors after only a few root inclusion tests.

The maximum speedup an algorithm can attain is clearly an important issue. One of the best-known models for estimating an upper bound on speedup for a parallel algorithm is based on an adaptation of Amdahl's law [1] to multiprocessor systems:

$$\textit{Maximum Speedup} = \frac{P}{P(1-f)+f} \tag{9}$$

where f is the fraction of code which may be performed in parallel on P processors. Equation (9) represents an upper bound on performance because it assumes that negligible time is spent on processor startup, interprocessor communication, synchronization, and other factors which degrade program performance. From (9), it is easily seen that a significant fraction of code must run in parallel in order to approach maximum speedup, and this effect is amplified as the number of processors increases.

Our algorithm may underutilize the available processors when the rate of interval bisection is insufficient for maintaining a suitably large stack, thereby reducing the fraction of operations executing in parallel and decreasing the maximum speedup it can realize based on (9). Fortunately, several steps in the algorithm present the opportunity for further parallelization, and the fraction of concurrently executing code could be increased by implementing such finer-grained parallelism in addition to the large-grained, subroutine-level parallelism incorporated here. For instance, the individual steps of the Gauss-Seidel procedure could be spread across many processors [14], and another level of parallelism could be exploited by implementing an appropriate parallel sparse linear solver [4, 5, 35] for computing the inverse point Jacobian preconditioning matrix. In addition, the point Newton method in Step IV (a) offers several opportunities for finer-grained parallelism [30].

3 Flowsheeting test problems

For our computational experiments, we used five flowsheeting problems, all of which are derived from problems appearing in the chemical engineering literature. These problems include models for an ethylene plant, an interconnected system of mixer and divider units, an ammonia plant, a two-component flash, and a single-stage distillation column. The flowsheeting modeling equations are based on the formulations from SEQUEL-II, a prototype equation-based flowsheeting system [26, 34].

The types of functions used in the flowsheet models are related to material balances, energy balances, and phase-equilibrium calculations. Equations describing material balances are nearly linear since they indicate only that the amount of a given material flowing into a process unit is the amount that also must flow out of it. Energy, or enthalpy, balances, on the other hand, are complex nonlinear functions of temperature. For these test problems, we used ideal vapor heat capacities and heat capacities estimated with the corresponding states model attributed to Sternling and Brown [22] to calculate the enthalpies of the vapor and liquid streams, respectively. Phase-equilibrium calculations, for computing the vapor and liquid fractions of a material, are even more complex. Equations for determining phase equilibria are functions of pressure and temperature and are tightly coupled to the energy balance equations. For the equilibrium calculations performed here, we implemented Raoult's Law with Antoine's formula for computing saturation pressures.

In these types of functions, the main variables represent flowrates for entire streams and individual chemical components, pressures, temperatures, and enthalpies. Other variables, added for specific process units, represent parameters such as heat duties, mole fractions, and the fraction of material flowing into each stream located downstream from a splitter.

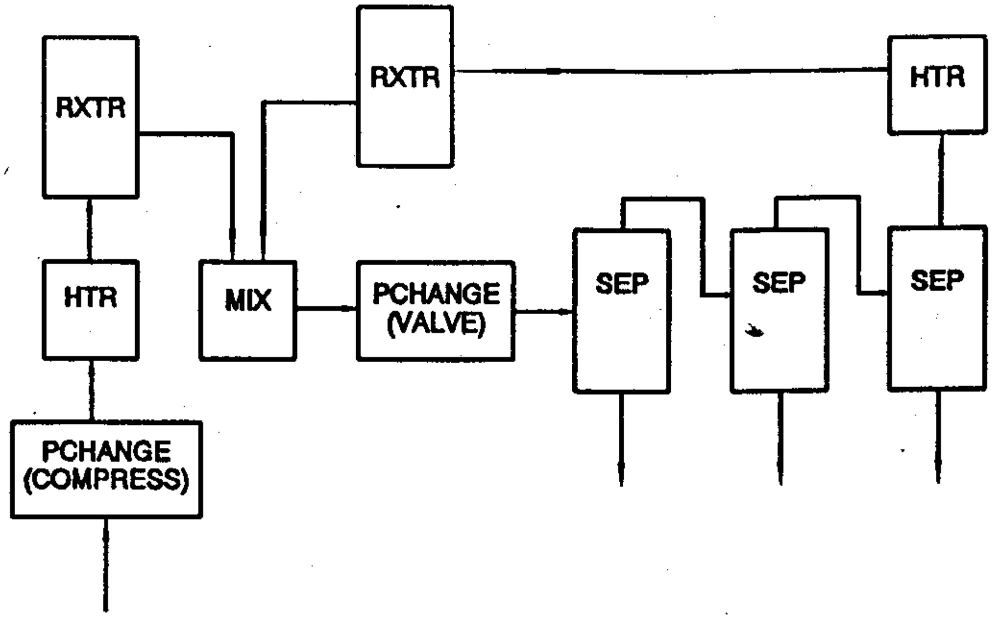

Figure 1: Ethylene plant flowsheet

The first test problem is an ethylene plant model adapted from Sample Problem #3 in [18]. The flowsheet for this problem is shown in Figure 1. In this process, chemical reactions convert ethane and propane to ethylene and various byproducts. The reaction in the first reactor (RXTR) is

$$3C_2H_6 + 6C_3H_8 \rightarrow 4H_2 + 4CH_4 + 5C_2H_4 + 2C_3H_6 + C_4H_{10} \qquad (10)$$

and the reaction in the second reactor is

$$4C_2H_6 \rightarrow 2H_2 + 2CH_4 + 3C_2H_4. \qquad (11)$$

For this problem, only material balances are written, and thus the resulting system of equations is nearly linear.

The second problem involves only mixing and dividing units, and its flowsheet is shown in Figure 2. This problem, like the first problem, was adapted from [18] and is considered interesting in the chemical engineering literature because it contains multiple material recycle and feed-forward loops. The equation system includes both energy and material balances, but it includes few complex nonlinear equations because no phase-equilibrium calculations are necessary.

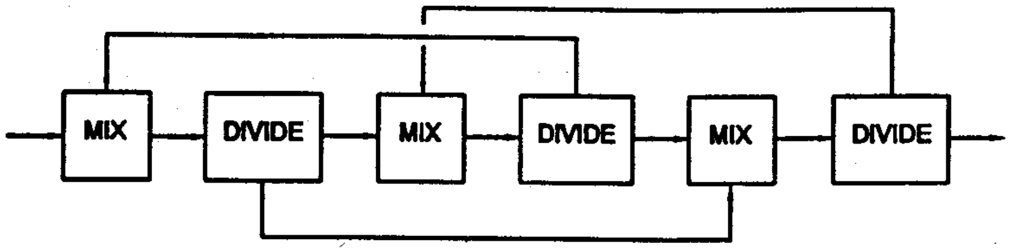

Figure 2: Flowsheet for the interconnected system of mixers and dividers

The third problem is an ammonia plant model, adapted from [24], describing the flowsheet in Figure 3. This process converts hydrogen and nitrogen to ammonia via the chemical reaction

$$3H_2 + N_2 \rightarrow 2NH_3. \qquad (12)$$

Both energy and material balances are written for this flowsheet, and phase-equilibrium calculations are required for the flash separation units.

The fourth problem is a two-component phase-equilibrium problem taken from Example 8–14 in [21]. Its flowsheet is shown in Figure 4. In this

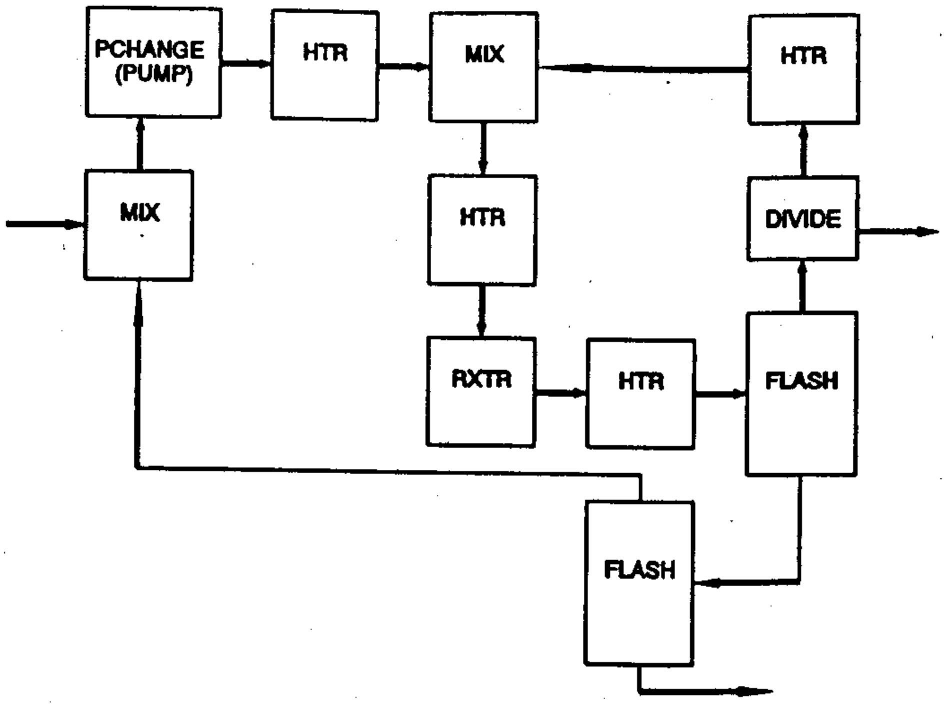

Figure 3: Ammonia plant flowsheet

problem, a mixture of ethane and n-heptane at a fixed temperature and pressure is allowed to separate into vapor and liquid phases, and the amount and chemical composition of each phase is computed. The only difference between the problem stated in the reference and the problem we solved is the equation of state used.

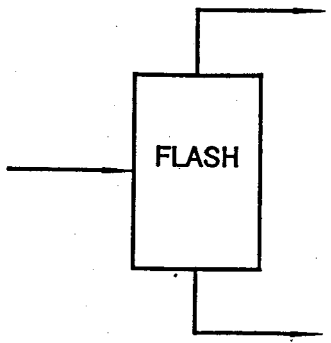

Figure 4: Flowsheet for the two-component flash problem

The fifth flowsheet represents a single-stage distillation column with a total condenser. This problem is adapted from [10], and its flowsheet is shown in Figure 5. The problem involves separating methanol from 1-propanol, and is similar to problem 4 except that it includes a material recycle loop and a phase change in the cooling unit.

Statistics describing the size of these problems are summarized in Table 1. For each of the five problems, the table lists the number of variables, the number of nonzero coefficients in the occurrence matrix, the number of unit operations, the number of process streams connecting the units, and the number of chemical components. Note that all of the problems are relatively small for process flowsheets.

The interval components of the initial interval boxes were set to the widest possible values representing reasonable physical limits for all of the problems except the ammonia plant model. For example, the initial intervals for the unknown variables in the two-component flash problem were $[0., 1000.]$ for flowrates, $[310., 500.]$ for temperatures, $[0., 1000.]$ for pressures, $[-100000., 100000.]$ for the heat duty and enthalpies, and $[0., 800.]$ for the ratios of vapor mole fractions to liquid mole fractions. For the ammonia plant problem, the initial interval box was based on estimates derived from values of the design specifications.

4 Results from computational experiments

The results from applying the parallel algorithm to solve the test problems described in the previous section are discussed here. The sequential runs were performed on one processor of a BBN TC2000 multiple-processor computer, and the parallel runs were executed on multiple processors of a BBN TC2000. The algorithm was implemented with the Uniform System programming model. We selected the Uniform System because it provides memory and process management tools which efficiently map our algorithm onto the hardware configuration of the BBN TC2000.

Results from running a sequential implementation of the algorithm on only one processor are given in Table 2. These runs were performed mainly to establish a basis for measuring the performance of the multiple-processor runs, but a few comments concerning the application of the interval Newton/generalized bisection technique to flowsheeting problems are in order here. First, the problem size as gauged by the number of variables does not appear to be the most important factor in determining the number of

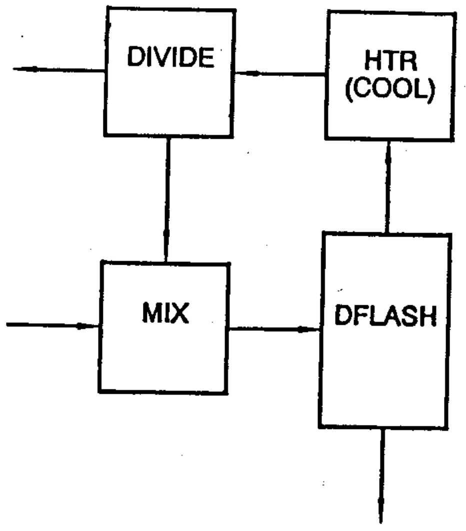

Figure 5: Flowsheet for the single-stage distillation column

Problem	Number of Variables	Number of Nonzeros	Number of Units	Number of Streams	Number of Components
Ethylene Plant	163	497	10	14	7
Mixers and Dividers	146	509	6	10	10
Ammonia Plant	177	659	11	15	5
Two-Component Flash	21	57	1	3	2
Single-Stage Column	50	136	4	7	2

Table 1: Statistics for test problems

root inclusion tests required to locate the solutions. Rather, the degree of difficulty the method experiences depends heavily on the complexity of the nonlinearities and the amount of coupling between the equations. Note

	Problem #				
	Ethylene Plant	Mixers and Dividers	Ammonia Plant	Two-Component Flash	Single-Stage Column
# of roots found	1	1	1	3	0*
Maximum binary tree level reached	1	7	16	27	81
Maximum stack depth reached	0	6	14	21	38
# of root inclusion test calls	1	7	108	569	100001
CPU time (sec)	3.1	21.8	1003.5	49.0	21789.6

* — Reached the maximum number of root inclusion test calls before finishing. Problem is known to have at least one solution.

Table 2: Results from running a sequential implementation of the algorithm

that the ethylene plant model, the second largest problem tested, was the easiest problem for the method to solve, while the relatively small single-stage distillation problem was the most difficult. However, recall that the model for the ethylene plant is nearly linear, in contrast to the single-stage distillation problem which includes highly nonlinear, tightly coupled equations arising from the equilibrium flash unit and the material recycle loop. Other factors influencing the required number of root inclusion test calls include the number of solutions located in the initial interval box and the size of the initial box. For example, the two-component flash problem presented here required a relatively large number of root inclusion test calls to isolate the three solutions, only one of which represents a physically attainable condition. This demonstrates that even relatively simple problems may have multiple solutions. While a traditional, point method may converge to a single nonphysical root, the interval Newton/generalized bisection algorithm presented here successfully locates them all. It should be noted that a different formulation of the problem, not included in this paper, requires only one root inclusion test call. The alternative formulation involves a similar amount of complexity and coupling but has only one root. Finally, the size of the initial box dramatically affected the method's behavior on the single-stage distillation problem. For the problem reported here, the elements of the initial interval box were unnecessarily wide, and the method was unable to locate any solutions, even after more than six hours of CPU time and $100,000$ root inclusion test calls. On the other hand, only one root inclusion test is required to locate a solution to this problem when the initial box is based on estimates more accurately reflecting the character of the process.

For the parallel runs on multiple processors, the parallel implementation of the algorithm was run on increasing numbers of processors until the speedups appeared to approach some maximum value. With the exception of the maximum stack depth, the statistics recorded in Table 2 are the same for both the sequential and parallel programs. This is expected since there are no differences in the numerical techniques between the programs. The difference lies in the order in which interval boxes are stored on and removed from the stack. Multiple processors performing simultaneous root inclusion tests will handle the stack differently for every run, even when the same problem is retested on the same number of processors, owing to small differences between the individual processors and in the interprocessor communications. It should also be mentioned here that the number of root inclusion test calls executed for the single-stage distillation problem is a minor exception since the program halts before the entire initial interval is processed. The sequential program halts as soon as the number of root inclusion test calls exceeds the user-supplied maximum, but the parallel program allows all active processors to complete root inclusion tests in progress at the time the maximum is reached. Thus for the single-stage distillation problem, the number of root inclusion test calls the parallel program executes may vary from run to run and is usually slightly greater than the number performed by the sequential program.

For parallel runs solving four of the flowsheeting problems studied here, Figure 6 shows plots of speedup against numbers of processors, and Table 3 lists an approximation of the fraction of operations executed in parallel for each of these four problems. For each problem, the fraction of concurrently executing operations was estimated by substituting the speedup gained on the maximum number of processors used into equation (9) and solving for f. The ethylene plant model was not solved with the parallel program since it requires only one root inclusion test call and therefore would not benefit from the parallel algorithm developed here.

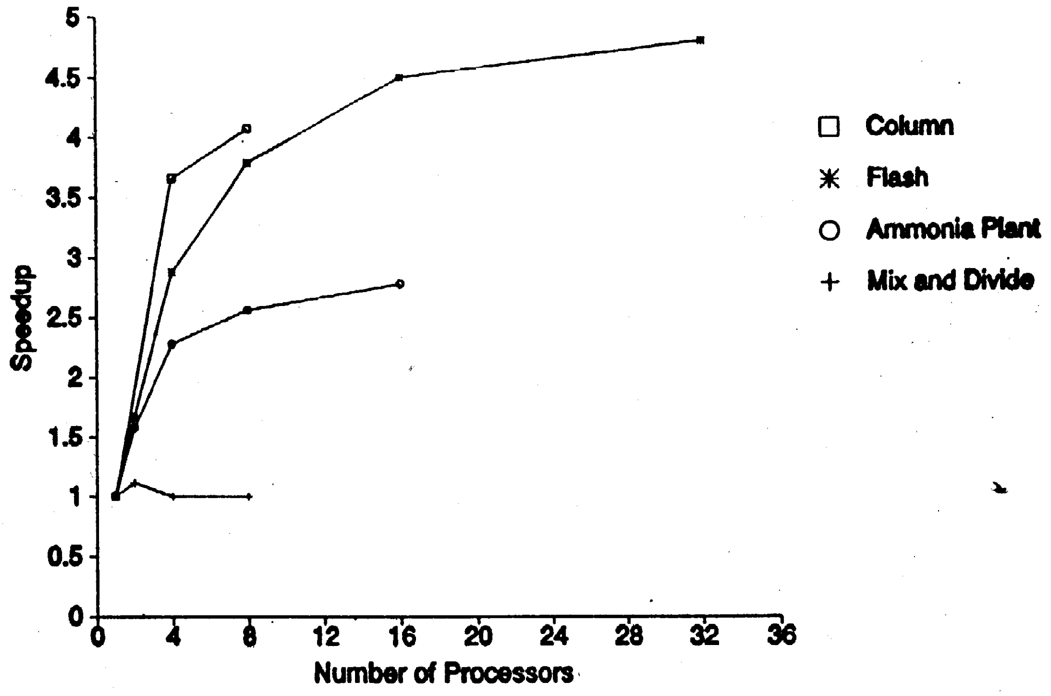

Figure 6: Parallel speedup preformance

The maximum speedups the parallel program achieved for the four flowsheeting problems ranged from one to 4.8, and the fraction of concurrently executing operations ranged from zero to 0.86. The maximum speedups seem most strongly related to the number of root inclusion test calls required and to the extent of branching in the binary search tree arising from interval bisections. Of the four problems studied here, the interconnected system of mixers and dividers required the fewest root inclusion tests, at seven, and Table 2 shows that the sequential program reached level seven in the binary search tree and level six on the stack of interval boxes. This

Problem	Fraction in Parallel
Mixers and Dividers	0.00
Ammonia Plant	0.68
Two-Component Flash	0.82
Single-Stage Column	0.86

Table 3: Fraction of operations executed in parallel

information indicates that the binary search tree for the system of mixers and dividers branches according to our worst-case scenario, in which only two boxes are present at each level of the binary tree. For this problem, the parallel algorithm would therefore achieve a speedup of only two under the best circumstances, and here it fails to produce any speedup at all. Different computational requirements for each of the seven root inclusion tests and the startup overhead costs associated with activating multiple processors account for this performance. In contrast, the remaining three problems require sufficient numbers of root inclusion test calls and subsequent binary search tree branching to utilize reasonable numbers of processors. For these three problems, the fraction of concurrently executing operations increases as the number of root inclusion test calls increases. The speedups seem to follow the same trend, even though the two-component flash problem results in a greater speedup than the single-stage distillation problem while requiring fewer test calls. In comparison with the flash problem, the distillation problem resulted in a smaller speedup value from a greater fraction of concurrently executing operations because the program achieved the speedup on fewer processors.

5 Discussion and conclusions

When applied to process flowsheeting problems, the implementation of our large-grained parallel algorithm on the BBN TC2000 is capable of achieving speedups approaching five with approximately 70 to 85 percent of the operations executing concurrently. This performance does not justify running the program on more than a few processors, and the single-stage distillation example illustrates that using an arbitrarily large initial interval box is not always feasible with this algorithm. Fortunately, more impressive speedups and processor utilization could be achieved on larger numbers of processors by exploiting the smaller-grained parallelism within the large-grained tasks of our algorithm.

In conclusion, our studies show that this interval Newton/generalized bisection technique is applicable as a global nonlinear solver for isolating all solutions to process flowsheeting models. The large-grained parallel approach described here will be most effective when combined with smaller-grained parallel tasks and implemented on a very large number of processors.

Acknowledgements

This work was funded in part by the National Science Foundation under grant DDM9024946. The authors thank the Mathematics and Computer Science Division of Argonne National Laboratory for providing time on the BBN TC2000 and Halliburton Energy Services for encouraging publication of this work.

References

[1] Amdahl, G. M. *Validity of the single-processor approach to achieving large scale computing capabilities.* AFIPS Conf. Proc. **30** (1967), p. 483.

[2] Chen, H.-S. and Stadtherr, M. A. *A modification of Powell's dogleg method for solving systems of nonlinear equations.* Comput. Chem. Engng. **5** (3) (1981), pp. 143–150.

[3] Chen, H.-S. and Stadtherr, M. A. *On solving large sparse nonlinear equation systems.* Comput. Chem. Engng. **8** (1) (1984), pp. 1–7.

[4] Coon, A. B. and Stadtherr, M. A. *Parallel implementation of sparse LU decomposition for chemical engineering applications.* Comput. Chem. Engng. **13** (8) (1989), pp. 899–914.

[5] Coon, A. B. and Stadtherr, M. A. *Parallel sparse matrix solvers for equation-based process flowsheeting.* I. Chem. E. Symp. Ser. **114** (1989), pp. 165–175.

[6] Duff, I. S., Erisman, A. M., and Reid, J. K. *Direct methods for sparse matrices.* Clarendon Press, Oxford, 1986.

[7] Flynn, M. J. *Very high-speed computing systems.* Proc. IEEE **54** (12) (1966), pp. 1901–1909.

[8] Hansen, E. and Sengupta, S. *Bounding solutions of systems of equations using interval analysis.* BIT **21** (1981), pp. 203–211.

[9] Hellerman, E. and Rarick, D. *The partitioned preassigned pivot procedure (P^4).* In: Rose, D. J. and Willoughby, R. A. (eds) "Sparse matrices and their applications", Plenum Press, New York, NY, 1972.

[10] Jacobsen, E. W. and Skogestad, S. *Multiple steady states in ideal two-product distillation.* AIChE J. **37** (4) (1991), pp. 499–511.

[11] Kearfott, R. B. *Abstract generalized bisection and a cost bound.* Math. Comput. **49** (179) (1987), pp. 187–202.

[12] Kearfott, R. B. *Some tests of generalized bisection.* ACM Trans. Math. Softw. **13** (3) (1987), pp. 197–220.

[13] Kearfott, R. B. *Preconditioners for the interval Gauss-Seidel method.* SIAM J. Numer. Anal. **27** (3) (1990), pp. 804–822.

[14] Kearfott, R. B., Bayoumi, M., Yang, Q., and Hu, C. *The all-row preconditioned interval Newton method on an MIMD machine.* The Second International Conference on Industrial and Applied Mathematics, Washington, D.C., 1991.

[15] Kearfott, R. B. and Novoa III, M. *INTBIS, a portable interval Newton/bisection package.* ACM Trans. Math. Softw. **16** (2) (1990), pp. 152–157.

[16] Kearfott, R. B, Hu, C., and Novoa III, M. *A review of preconditioners for the interval Gauss-Seidel method.* Interval Computations **1** (1) (1991), pp. 59–85.

[17] Kuno, M. and Seader, J. D. *Computing all real solutions to systems of nonlinear equations with a global fixed-point homotopy.* Ind. Eng. Chem. Res. **27** (1988), pp. 1320–1329.

[18] Motard, R. L. and Lee, H. M. *CHESS: chemical engineering simulation system, user's guide.* University of Houston, Houston, TX, 1971.

[19] Neumaier, A. *Interval methods for systems of equations.* Cambridge University Press, Cambridge, 1990.

[20] Perkins, J. D. *Equation-oriented flowsheeting.* In: Westerberg, A. W. and Chien, H. H. (eds) "Proceedings of the Second International Conference on Foundations of Computer-Aided Process Design", CACHE, 1983.

[21] Reid, R. C., Prausnitz, J. M., and Poling, B. E. *The properties of gases and liquids.* 4th edition, McGraw-Hill, New York, NY, 1987.

[22] Reid, R. C., Prausnitz, J. M., and Sherwood, T. K. *The properties of gases and liquids.* 3rd edition, McGraw-Hill, New York, NY, 1977.

[23] Schnepper, C. A. *Large grained parallelism in equation-based flowsheeting using interval Newton/generalized bisection techniques.* Ph.D. Dissertation, University of Illinois, Urbana, IL, 1992.

[24] Seader, J. D., Seider, W. D., and Pauls, A. C. *FLOWTRAN simulation: an introduction.* 2nd edition, CACHE, Cambridge, MA, 1977.

[25] Seider, W. D., Brengel, D. D., and Widagdo, S. *Nonlinear analysis in process design.* AIChE J. **37** (1) (1991), pp. 1–38.

[26] Stadtherr, M. A. and Hilton, C. M. *Development of a new equation-based process flowsheeting system: numerical studies.* Computer-Aided Process Design and Analysis, AIChE Symp. Ser. **78** (214) (1982), pp. 12–28.

[27] Stadtherr, M. A. and Wood, E. S. *Sparse matrix methods for equation-based chemical process flowsheeting — I: Reordering phase.* Comput. Chem. Engng. **8** (1) (1984), pp. 9–18.

[28] Stadtherr, M. A. and Wood, E. S. *Sparse matrix methods for equation-based chemical process flowsheeting — II: Numerical phase.* Comput. Chem. Engng. **8** (1) (1984), pp. 19–33.

[29] Swaney, R. E. and Wilhelm, C. E. *Robust solution of flowsheeting equation systems.* In: Siirola, J. J., Grossman, I. E., and Stephanopoulos, G. (eds) "Proceedings of the Third International Conference on Foundations of Computer-Aided Process Design", CACHE, 1990.

[30] Vegeais, J. A. and Stadtherr, M. A. *Parallel processing strategies for chemical process flowsheeting.* AIChE J. **38** (9) (1992), pp. 1399–1407.

[31] Wayburn, T. L. and Seader, J. D. *Solution of systems of interlinked distillation columns by differential homotopy-continuation methods.* In: Westerberg, A. W. and Chien, H. H. (eds) "Proceedings of the Second International Conference on Foundations of Computer-Aided Process Design", CACHE, 1983.

[32] Wayburn, T. L. and Seader, J. D. *Homotopy continuation methods for computer-aided process design.* Comput. Chem. Engng. **11** (1) (1987), pp. 7–25.

[33] Westerberg, A. W., Hutchinson, H. P., Motard, R. L., and Winter, P. *Process flowsheeting.* Cambridge University Press, 1979.

[34] Zitney, S. E. and Stadtherr, M. A. *Computational experiments in equation-based chemical process flowsheeting.* Comput. Chem. Engng. **12** (12) (1988), pp. 1171–1186.

[35] Zitney, S. E. and Stadtherr, M. A. *Frontal algorithms for equation-based chemical process flowsheeting on vector and parallel computers.* Comput. Chem. Engng. **17** (4) (1993), pp. 319–338.

C. A. Schnepper
Halliburton Services
Research Center
P.O. Drawer 1431
Duncan, OK 73536
USA

M. A. Stadtherr
University of Illinois @ Urbana-Champaign
Department of Chemical Engineering
R.A.L. — Box C–3
1209 W. California Ave.
Urbana, IL 61801
USA