Strategies for Equilibrium-Stage Separation Calculations on Parallel Computers

Alfred J. O'Neill, Daniel J. Kaiser, and Mark A. Stadtherr
Dept. of Chemical Engineering, University of Illinois, Urbana, IL 61801

When multicomponent, multistage separation problems are solved on parallel computers by successive linearization methods, the solution of a large sparse linear equation system becomes a computational bottleneck, since other parts of the calculation are more easily parallelized. When the standard problem formulation is used, this system has a block-tridiagonal form. It is shown how this structure can be used in parallelizing the sparse matrix computation. By reformulating the problem so that it has a bordered-block-bidiagonal superstructure, it can be made even more amenable to parallelization. These strategies permit the use of a two-level hierarchy of parallelism that provides substantial improvements in computational performance on parallel machines.

Introduction

The need to solve multicomponent equilibrium-stage separation calculations arises frequently in chemical process analysis and design. The rigorous simulation of separation columns is a computationally intense problem that often represents the largest computational effort in solving flowsheet simulation and optimization problems. Thus, the solution of these problems is an attractive application for parallel computers, since these offer the potential of much higher computational speed than conventional sequential machines. Since, in principle, they involve no upper limit on speed, parallel computing architectures are the inevitable future of high-speed scientific and engineering computing. However, since current methods for solving multicomponent separation problems were developed for use on conventional serial computers, they usually are not capable of taking much advantage of the potential power of parallel machines. Thus, the strategies used to solve these problems must be rethought. We address here the development of new problem-solving strategies for exploiting the power of parallel computing in solving multicomponent equilibrium-stage separation problems.

There are a wide variety of methods available for solving multicomponent separation problems (Henley and Seader, 1981), including equation-tearing methods and simultaneous correction procedures. The latter are desirable since they are more robust for difficult problems, and they allow more flexibility in problem specifications. Furthermore, they are analogous to the equation-based approach for solving the more general process flowsheeting problem, which Vegeais and Stadtherr (1992) have found to have the best potential for exploiting parallel computing. Thus, we concentrate on the use of simultaneous correction methods. In this case, the problem is formulated as a large set of nonlinear equations to be solved by successive linearization, usually using Newton-Raphson or some variation thereof. This involves the solution of a large sparse system of linear equations, which often represents a large fraction of the overall computing time. As noted by Vegeais and Stadtherr (1992), the sparse matrix problem becomes even more of a computational bottleneck on a parallel machine because other parts of the calculation, such as physical property evaluation, are much more readily parallelized than the solution of the sparse matrix.

We present here efficient parallel techniques for the solution of the large sparse linear systems arising in multicomponent equilibrium-stage separation problems. We begin by considering the well-known Naphtali-Sandholm (1971) problem formulation, which leads to a sparse matrix in block-tridiagonal form. A technique is described for exploiting this matrix structure in effectively parallelizing the computation. We then consider how the problem can be reformulated to further enhance parallelization. These techniques facilitate the use of a two-level hierarchy of parallelism, which can be used to improve the overall computational efficiency significantly.

Correspondence concerning this article should be addressed to M. A. Stadtherr.
D. J. Kaiser is presently at Dow Chemical Co., Midland, MI 48667.

Background

Parallel computing

In using parallel computing one can distinguish between what is often called *low-level* (or *small-grained*) and *high-level* (or *large-grained*) parallelism. By low level we mean parallel tasks that can, in principle, be identified and executed without significant user intervention. Usually such tasks must be relatively small, on the DO-loop level or lower, and the parallelization of such tasks is sometimes referred to as *microtasking*. Parallelization can be done on the machine level or by the compiler. Typically, at least today, this may involve some code changes made to enhance the detectability of independence in calculations by parallelizing compilers. These code changes may involve little or no significant change in the algorithm; instead, they cause the compiler to generate a different, and usually more efficient, set of machine-level instructions. Many of today's parallelizing compilers can already identify much of this low-level parallelism without any code changes, and the next generation of such compilers should be even more effective in detecting this independence in operations. However, there are still advantages to switching to algorithms that can take greater advantage of small-grained parallelism.

By high-level parallelism we refer to opportunities for parallelism that usually must be recognized by the algorithm developer, typically based on knowledge of a specific problem or class of problems that cannot be imparted to a compiler. That is, we have in mind an independence of calculations for which explicit provisions can be made in the formulation of the algorithm. The solution method is designed or chosen largely for its amenability to parallelization. Generally, high-level parallel tasks will be larger than on the low level; thus, there are opportunities for low-level parallelism within the high-level tasks. In fact, exploitation of low-level parallelism within the tasks generated by large-grained methods allows a multilevel concurrency that significantly enhances the overall parallel performance of an algorithm. Use of such hierarchical parallelism is often the key in achieving good performance.

In discussing performance on parallel machines, Amdahl's law is often used. This relationship has the form $S_A = P/[P(1-f)+f]$, where f is the fraction of utilized code performed in parallel, and P is the number of processors. The Amdahl's law speedup S_A is defined as the ratio of the time it takes for a computation to execute on one processor of a machine to the time it takes for the same computation to execute with P processors on the same machine. Measurement of S_A on a given number of processors is a commonly used measure of the extent to which an algorithm can be parallelized. There are other performance metrics as well. From a practical standpoint, one should ideally compute speedup relative to the standard serial algorithm for a given problem, not the parallel algorithm running on one processor. We thus define the useful speedup S_U ($\leq S_A$) as the ratio of the time it takes for the standard serial algorithm to execute on one processor of a machine to the time it takes for a parallel algorithm to execute with P processors on the same machine.

Amdahl's law and speedup in this context have been discussed by Vegeais and Stadtherr (1992). A basic lesson of Amdahl's law is that, even with a large number of processors, if there is just a small amount of code that cannot be run in parallel, the potential speedup will be greatly limited. For instance, on a 64-processor machine, even if 90% of a code is able to run in parallel ($f = 0.90$), the maximum speedup possible is about 8.8; if the parallelization is $f = 0.95$, then the speedup is limited to about 15.4. As noted by Gustafson (1988), however, for most applications f increases with problem size. That is, as larger and larger problems are attempted, the speedup tends to increase. Furthermore, the Amdahl's law limits can be bypassed by using a hierarchy of parallelism, as discussed above, since these limits apply only on each level in the hierarchy.

Problem formulation

Consider the Naphtali-Sandholm (1971) model for a vapor-liquid equilibrium-stage separation operation involving N stages and c components. Each stage n (including condensers and reboilers) is represented by an energy balance:

$$E_n = (1+S_n)H_n + (1+s_n)h_n - H_{n+1} - h_{n-1} - h_{f,n} = 0 \quad (1)$$

c material balances:

$$M_{n,m} = (1+S_n)V_{n,m} + (1+s_n)L_{n,m} - V_{n+1,m} - L_{n-1,m} - f_{n,m} = 0 \quad (2)$$

and c equilibrium relationships:

$$Q_{n,m} = \frac{\eta_n K_{n,m} V_n L_{n,m}}{L_n} - V_{n,m} + \frac{(1-\eta_n)V_{n+1,m}V_n}{V_{n+1}} = 0 \quad (3)$$

where $n = 1, 2, \ldots, N$ and $m = 1, 2, \ldots, c$. This provides a total of $N(2c+1)$ equations. The $N(2c+1)$ variables are taken to be the N plate temperatures T_n and the $2Nc$ vapor and liquid component flow rates $V_{n,m}$ and $L_{n,m}$. This assumes that all the feed stream flows $f_{n,m}$ are completely specified, as are the heat loads $h_{f,n}$ for all stages (including partial condenser and partial reboiler), the sidestream ratios S_n and s_n, and the column pressure. If other top and bottom specifications are used or if a total condenser or total reboiler is used, these equations can be easily modified as described by Naphtali and Sandholm. The nonlinear equation system (Eqs. 1-3) is then solved by successive linearization. The bottleneck in parallelizing this computation is the sparse matrix problem.

Sparse matrix problem

When Eqs. 1-3 are grouped by stage and ordered from stage 1 to N, the coefficient matrix for the sparse linear equation system that must be solved takes on a block-tridiagonal (BTD) form, as shown in Figure 1. The block matrices A_n, B_n, and C_n represent the partial derivatives of the equations for the nth stage with respect to the variables on stages $n-1$, n, and $n+1$, respectively. The overall matrix can be viewed as an $N \times N$ block matrix where the elements themselves are $(2c+1) \times (2c+1)$. These blocks themselves have a well-defined substructure, as shown in Figure 2. Note that the A and C blocks each have only $c+1$ nonzero columns. The equation ordering in the substructure shown here differs from that given by Naphtali and Sandholm (1971). The reordering was done to make the B_n, and thus the entire linear system, structurally stable (to have a zero-free diagonal). The advantages of starting

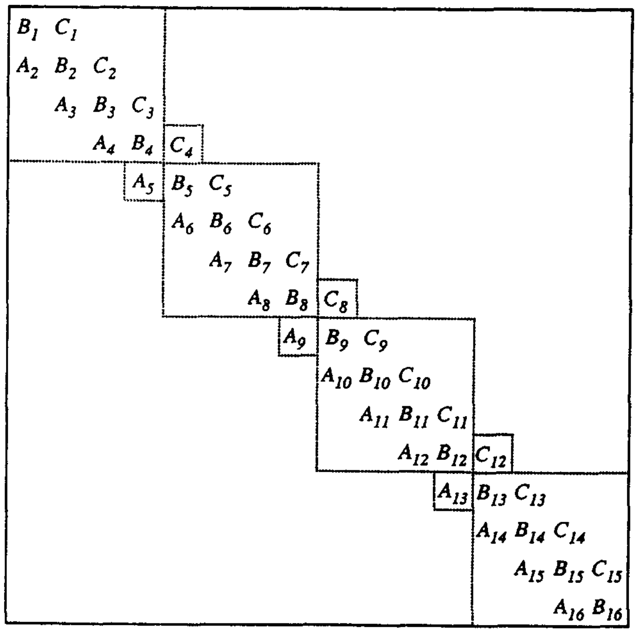

Figure 1. Block-tridiagonal (BTD) matrix structure resulting from the Naphtali-Sandholm problem formulation.

The subblock structure of the A, B and C matrices is shown in Figure 2. The dotted lines indicate the super-BTD structure used in the BTD strategy for parallelization.

with a structurally stable ordering have been discussed by Erisman et al. (1987).

BTD systems of this sort can be solved efficiently on serial machines using block-Gaussian elimination, which is often referred to as the block-Thomas algorithm (Henley and Seader, 1981) in this context. This is efficient since zero to nonzero fill-ins occur only in the B and C blocks. Because there is relatively little fill-in and the zero-nonzero subblock structure is well defined, the procedure can easily be implemented using full matrix code with direct addressing, as opposed to the usual sparse matrix code using indirect addressing that is required in more general sparse matrix problems. The use of full matrix code also makes possible the use of partial pivoting to maintain numerical stability, as opposed to the less effective threshold pivoting schemes used in general sparse matrix codes. In the numerical studies presented below, we use an efficient block-Gaussian elimination code referred to as BGE, which was

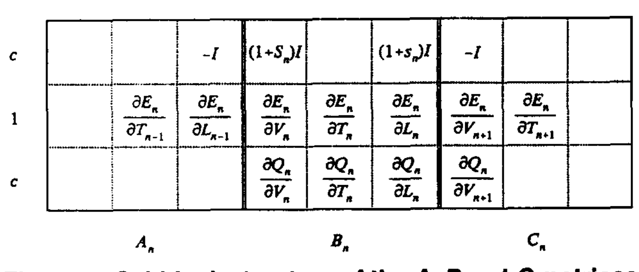
c 1 c c 1 c c 1 c
c
1
c
$-I$
$(1+S_n)I$
$(1+s_n)I$
$-I$
$\frac{\partial E_n}{\partial T_{n-1}}$
$\frac{\partial E_n}{\partial L_{n-1}}$
$\frac{\partial E_n}{\partial V_n}$
$\frac{\partial E_n}{\partial T_n}$
$\frac{\partial E_n}{\partial L_n}$
$\frac{\partial E_n}{\partial V_{n+1}}$
$\frac{\partial E_n}{\partial T_{n+1}}$
$\frac{\partial Q_n}{\partial V_n}$
$\frac{\partial Q_n}{\partial T_n}$
$\frac{\partial Q_n}{\partial L_n}$
$\frac{\partial Q_n}{\partial V_{n+1}}$
A_n
B_n
C_n

Figure 2. Subblock structure of the *A*, *B* and *C* matrices in Figure 1.

Row and column dimensions are indicated at the left and top, respectively.

developed specifically for equilibrium stage separation problems.

We now consider the problem of how to effectively use a two-level hierarchy of parallel processors in solving the sparse matrix problem arising in multicomponent separation calculations.

Low-Level Parallelism

It is well known that the solution of full linear systems can be efficiently parallelized at this level (Dongarra, 1993). Thus, a solution algorithm, such as BGE, that leads to the use of full matrix code on this problem can be expected to be effective here. The parallelization of tasks at this level can be done automatically by a microtasking compiler.

To test the effectiveness of block elimination in exploiting low-level parallelism, experiments were run using the BGE code on the eight processors of an Alliant FX/8 running in a dedicated environment and using the Alliant microtasking Fortran compiler. The test problems are derived from the four-component absorber problem described and solved by Naphtali and Sandholm (1971). Five problems, ranging in size from 24 to 288 equilibrium stages, were solved. Speedups based on total simulation time, including function and derivative evaluations, were determined. Since the algorithm used for parallelization here is the same as the standard serial algorithm for this problem, the Amdahl's law speedup S_A and the useful speedup S_U, as defined above, are the same.

The results showed that the speedup obtained by using BGE on this level was about 3.6, and did not vary significantly with the number of stages. While substantial, this is considerably less than the ideal speedup of 8. As will become clear later, as we see much higher parallelization when other sparse matrix solvers are used, the bottleneck here is not the parallelization of function or derivative evaluations, but of the sparse matrix problem. The difficulty here is the relatively small size, $(2c+1)\times(2c+1)=9\times9$ in this case, of the blocks that must be dealt with in applying block-Gaussian elimination. Because of the small workload per processor, the parallelization overhead due to synchronization and communication cannot be overcome. That the size of the blocks and thus the number of components is the critical factor here can be seen in the fact that the speedup did not increase with the number of stages. This was confirmed by making a similar set of runs on ten-component problems of the same structure. In this case, the average speedup increased to about 4.0 and again was roughly constant with respect to the number of stages. Clearly, for a larger number of components, the larger block sizes lead to larger parallel tasks, and parallelization overhead becomes a less important factor.

An Amdahl's law speedup of about 3.6–4.0, as seen here, corresponds to a parallelization f of about 83-86%. This is comparable to results obtained by Zitney and Stadtherr (1993) for microtasking two more general flowsheeting problems using a frontal method for handling the sparse matrix problem. The frontal method is similar to block-Gaussian elimination in that the sparse matrix calculations are performed using full matrix code; however, the frontal method is a more general approach.

For the fraction parallelized seen here, Amdahl's law indicates that even with an infinite number of processors, a speedup no more than about 7 could be obtained. Thus, for problems with up to ten components only about seven processors can be effectively utilized at this level, at least using this method. To utilize more processors and further increase computational speed, we must use another level in the hierarchy of parallelism, which uses clusters of processors. Low-level parallelism, as discussed here, is then used within each cluster.

High-Level Parallelism

Consider the situation in which there are P clusters of processors and there are P_C processors per cluster. In using high-level parallelism, we seek to parallelize the computation across these P clusters, while using low-level parallelism within each cluster if $P_C > 1$. One way to do this is to take advantage of the block-tridiagonal (BTD) matrix structure resulting from the use of the Naphtali-Sandholm formulation described above.

The method described here is motivated by a technique described by Sameh (1983) and discussed further by Dongarra and Sameh (1984) and Berry and Sameh (1988). The method is designed for the parallel solution of narrow-banded linear systems and BTD linear systems in which the off-diagonal blocks have a very small number of nonzero columns relative to the diagonal blocks. The method is not useful directly in solving the BTD systems arising from the Naphtali-Sandholm formulation because the number of nonzero columns $(c+1)$ in each of the off-diagonal blocks A and C is not sufficiently small compared to the number of columns $(2c+1)$ in the diagonal blocks B. Thus, we make use of the super-block-tridiagonal (s-BTD) structure shown by the dotted lines in Figure 1. Note that now the number of nonzero columns in the off-superdiagonal blocks is much smaller than the number of columns in the superdiagonal blocks, $c+1$ vs. $4(2c+1)$ in this example.

The use of this BTD strategy will be described using the example in Figure 1, which assumes $N=16$ total stages and $P=4$ clusters. An equal number of stages $N_C=N/P$ is then assigned to each cluster. For now, we will assume that N is evenly divisible by P; if it is not then for some clusters N_C will be one more than for others; this situation is discussed further below. Thus, in Figure 1, stages 1–4 (block rows 1–4) are assigned to cluster 1, stages 5–8 to cluster 2, and so on. Each group of stages assigned to a cluster is represented by a superblock row in the coefficient matrix, as indicated by the horizontal dotted lines in Figure 1. In general, stages $1+(k-1)N_C$ to kN_C are assigned to cluster k, and the corresponding block rows constitute superblock row k.

To solve the system, block elimination with partial pivoting is then applied within each cluster to reduce the corresponding superdiagonal blocks to identity matrices. The resulting matrix is shown in Figure 3. This results in the fill-in denoted by the W and Y blocks and leaves a reduced system, denoted by the shaded areas in Figure 3, which must be solved for the variables corresponding to the W and Y blocks. There are six block columns of fill-in here and $2(P-1)$ in general. Note that since the A and C blocks leading to the formation of the Y and W blocks each has only $c+1$ nonzero columns, the reduced system has a total of $6(c+1)$ variables here and $2(P-1)(c+1)$ in general. Thus, each W and Y block actually contributes only $c+1$ equations to the reduced system. For the W blocks these are the last $c+1$ rows, and for the Y blocks the first $c+1$ rows, with the row orders determined by the pivot sequence used in solving the superdiagonal block in each cluster. The parts of the W and Y blocks that contribute to the reduced system are denoted by W_R and Y_R, each of which is $(c+1)\times(c+1)$. In these terms, the reduced system is as shown in Figure 4.

The next step is to solve the reduced system. If it is sufficiently small, the reduced system can be solved by treating it as a full matrix and employing low-level parallel techniques using all available processors. This will yield a high computational rate, but also a number of wasted operations on zeros.

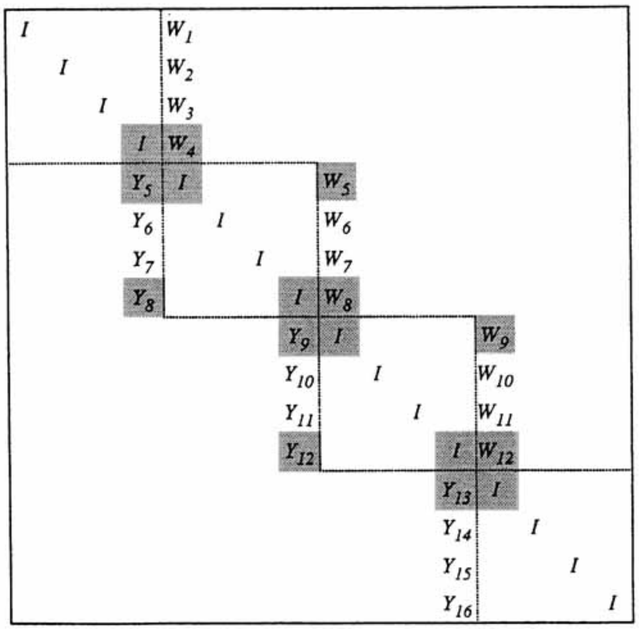

Figure 3. Result of applying the first step of the BTD strategy to the matrix in Figure 1.

The shaded area indicates the reduced system that must be solved next.

$$\begin{array}{c|c|c|c|c|c|c}
 & c+1 & c+1 & c+1 & c+1 & c+1 & c+1 \\
\hline
c+1 & I & W_{R,4} & & & & \\
\hline
c+1 & Y_{R,5} & I & & W_{R,5} & & \\
\hline
c+1 & Y_{R,8} & & I & W_{R,8} & & \\
\hline
c+1 & & & Y_{R,9} & I & & W_{R,9} \\
\hline
c+1 & & & Y_{R,12} & & I & W_{R,12} \\
\hline
c+1 & & & & & Y_{R,13} & I
\end{array}$$

Figure 4. Closeup of reduced system taken from shaded areas of Figure 4.

Row and column dimensions are indicated at the left and top, respectively.

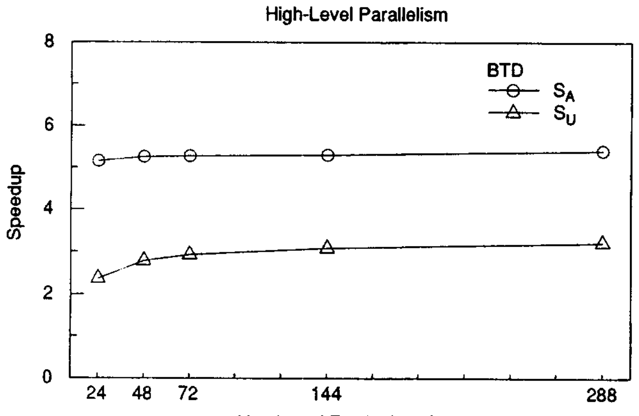

Figure 5. High-level speedup using the BTD strategy.

If the reduced system is larger, use of block elimination may be more attractive. This will yield a lower computational rate, but operations on zeros will be reduced. Still larger reduced systems could be handled by recursive application of the BTD strategy, since the reduced system also has an s-BTD structure, with half the original number of superdiagonal blocks.

Finally, once the reduced system is solved, the remaining variables can be solved for by backsubstitution. This can be done in parallel using the same assignment of block rows to clusters as before.

The BTD strategy was implemented using an Alliant FX/8 running in dedicated mode. To observe the effect of high-level parallelism only, we chose $P=8$ clusters and $P_C=1$ processor per cluster. The BGE code for block Gaussian elimination was used by each cluster to solve the superdiagonal blocks. The reduced system was treated as a full matrix and solved using low-level parallelism with all eight processors. The test problems solved are the same four component absorber problems used above.

Results are shown in Figure 5. The Amdahl's law speedup S_A was determined by comparing the solution time for BTD on eight clusters with the time to do the same computation on one cluster. On these problems S_A averages about 5.3, which corresponds to a fraction parallelized f of about 93%. The largest part of this inefficiency is due to the overhead associated with forming and solving the reduced system. There is additional overhead due to inefficiencies in parallelizing the function and derivative evaluations. Note that S_A increases slightly with problem size, as the workload per processor increases and the relative overhead cost decreases. On other tests with a larger number of components (ten), the workload per processor also increases, and the speedup also increases further. Another inefficiency is that there is an inherent workload imbalance in the method used, since the clusters dealing with the first and last superblock rows encounter less fill-in, a factor considered in more detail by Berry and Sameh (1988). This indicates that in the case that N is not evenly divisible by P, the first two remaining stages should be assigned one each to the first and last superblock rows.

The Amdahl's law speedup S_A is a good measure of the extent to which the BTD method can be parallelized. However, since the standard method for a single processor is BGE, not BTD, a more practical measure of speedup can be obtained by comparing the computation time for BTD on eight processors to the time for BGE on one processor. In this way, we obtain the useful speedup S_U figures in Figure 5. In this case, these mainly reflect the fact that on a single processor BGE is faster than BTD due to the need to compute and eliminate the W and Y blocks. Essentially, there is a tradeoff between the number of processors and operation count, but one that clearly goes in favor of parallelization.

Alternate problem formulations

The size of the reduced system, a critical factor affecting the extent of parallelism, can be decreased by reformulating the problem so that the blocks of fill-in occur in fewer block-columns. There are several ways to do this. To demonstrate the basic reformulation strategy we consider the common situation in which there are no sidestream products, and replace the material and energy balances around stages $j=kN_C+1$, $k=1, 2, \ldots, P-1$ (the first stages in all, but the first, superblock rows) with balances from stage 1 to stage j. Thus, for these stages Eq. 1 is replaced by:

$$\bar{E}_n = H_1 + h_n - H_{n+1} - \sum_1^n h_{f,n} = 0 \tag{4}$$

and Eq. 2 is replaced by:

$$\bar{M}_{n,m} = V_{1,m} + L_{n,m} - V_{n+1,m} - \sum_1^n f_{n,m} = 0. \tag{5}$$

The resulting coefficient matrix for this problem formulation is shown in Figure 6 and the resulting subblock structure is

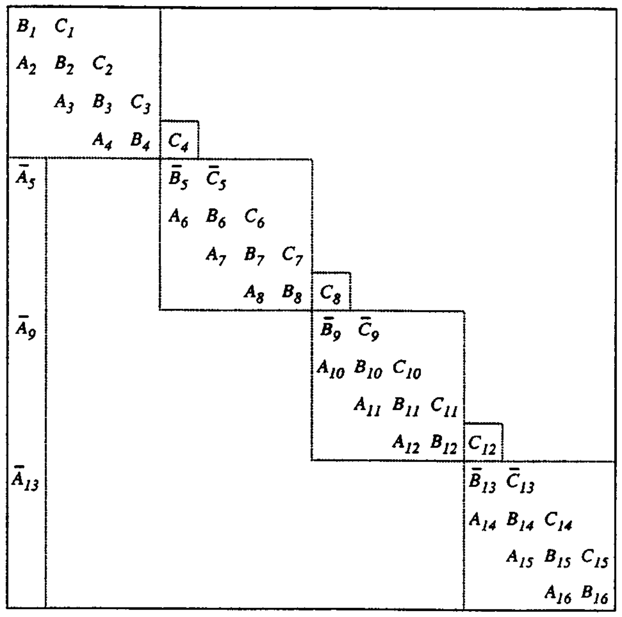
B_1 C_1
A_2 B_2 C_2
A_3 B_3 C_3
A_4 B_4 C_4
$\bar{A}_5$ $\bar{B}_5$ $\bar{C}_5$
A_6 B_6 C_6
A_7 B_7 C_7
A_8 B_8 C_8
$\bar{A}_9$ $\bar{B}_9$ $\bar{C}_9$
A_{10} B_{10} C_{10}
A_{11} B_{11} C_{11}
A_{12} B_{12} C_{12}
$\bar{A}_{13}$ $\bar{B}_{13}$ $\bar{C}_{13}$
A_{14} B_{14} C_{14}
A_{15} B_{15} C_{15}
A_{16} B_{16}

Figure 6. Matrix structure resulting from alternative problem formulation.

The subblock structure of the $\bar{A}$, $\bar{B}$ and $\bar{C}$ matrices is shown in Figure 7. The A, B and C matrices are as in Figure 2. The dotted lines indicate the bordered-block-bidiagonal (BBBD) superstructure used in the BBBD strategy for parallelization.

$$\begin{array}{c|ccc|ccc|ccc|}
 & c & 1 & c & c & 1 & c & c & 1 & c \\
\hline
c & & & & \frac{\partial Q_n}{\partial V_n} & \frac{\partial Q_n}{\partial T_n} & \frac{\partial Q_n}{\partial L_n} & \frac{\partial Q_n}{\partial V_{n+1}} & & \\
1 & \frac{\partial \bar{E}_n}{\partial V_1} & \frac{\partial \bar{E}_n}{\partial T_1} & & & \frac{\partial \bar{E}_n}{\partial T_n} & \frac{\partial E_n}{\partial L_n} & \frac{\partial E_n}{\partial V_{n+1}} & \frac{\partial E_n}{\partial T_{n+1}} & \\
c & I & & & & & I & -I & & \\
\hline
 & & \bar{A}_n & & & \bar{B}_n & & & \bar{C}_n &
\end{array}$$

Figure 7. Subblock structure of the $\bar{A}$, $\bar{B}$ and $\bar{C}$ matrices in Figure 6.
Row and column dimensions are indicated at the left and top, respectively.

shown in Figure 7. The matrix structure in Figure 6 now has a border or "spike" in block column 1. If the stages are assigned to clusters in the same fashion as before, the matrix takes on a superstructure that is bordered-block-bidiagonal (BBBD), as indicated by the dotted lines in Figure 6. A BBBD solution strategy then can be implemented in the same fashion as BTD.

Again, block elimination with partial pivoting is applied within each cluster to reduce the corresponding superdiagonal blocks to identity matrices. The resulting matrix is shown in Figure 8. Again, this results in fill-in, as denoted by the $\bar{W}$ and $\bar{Y}$ blocks, leaving a reduced system, indicated by the shaded areas in Figure 8, to be solved for the variables corresponding to the $\bar{W}$ and $\bar{Y}$ blocks. Note, however, that though the number of filled-in blocks remains the same as in the BTD strategy, now all the $\bar{Y}$ blocks occur in the same block column. Thus, there are only four block columns of fill-in here vs. six for BTD, and in general only P block columns of fill-in vs. $2(P-1)$ for BTD. Since the $\bar{A}$ and C blocks leading to the formation

$$\begin{array}{cccc|cccc|cccc|cccc}
I & & & & \bar{W}_1 & & & & & & & & & & & \\
 & I & & & \bar{W}_2 & & & & & & & & & & & \\
 & & I & & \bar{W}_3 & & & & & & & & & & & \\
 & & & I & \bar{W}_4 & & & & & & & & & & & \\
\hline
\bar{Y}_5 & & & & I & & & & \bar{W}_5 & & & & & & & \\
\bar{Y}_6 & & & & & I & & & \bar{W}_6 & & & & & & & \\
\bar{Y}_7 & & & & & & I & & \bar{W}_7 & & & & & & & \\
\bar{Y}_8 & & & & & & & I & \bar{W}_8 & & & & & & & \\
\hline
\bar{Y}_9 & & & & & & & & I & & & & \bar{W}_9 & & & \\
\bar{Y}_{10} & & & & & & & & & I & & & \bar{W}_{10} & & & \\
\bar{Y}_{11} & & & & & & & & & & I & & \bar{W}_{11} & & & \\
\bar{Y}_{12} & & & & & & & & & & & I & \bar{W}_{12} & & & \\
\hline
\bar{Y}_{13} & & & & & & & & & & & & I & & & \\
\bar{Y}_{14} & & & & & & & & & & & & & I & & \\
\bar{Y}_{15} & & & & & & & & & & & & & & I & \\
\bar{Y}_{16} & & & & & & & & & & & & & & & I
\end{array}$$

Figure 8. Result of applying the first step of the BBBD strategy to the matrix in Figure 6.
The shaded area indicates the reduced system that must be solved next.

$$\begin{array}{c|cccc|}
 & c+1 & c+1 & c+1 & c+1 \\
\hline
c+1 & I & \bar{W}_{R,1} & & \\
c+1 & \bar{Y}_{R,5} & I & \bar{W}_{R,5} & \\
c+1 & \bar{Y}_{R,9} & & I & \bar{W}_{R,9} \\
c+1 & \bar{Y}_{R,13} & & & I \\
\hline
\end{array}$$

Figure 9. Closeup of reduced system taken from shaded areas of Figure 6.
Row and column dimensions are indicated at the left and top, respectively.

of the $\bar{Y}$ and $\bar{W}$ blocks each has only $c+1$ nonzero columns, the reduced system has a total of $4(c+1)$ variables here and $P(c+1)$ in general. Thus, each $\bar{W}$ and $\bar{Y}$ block contributes only $c+1$ equations to the reduced system, namely the first $c+1$ rows, with the row orders determined by the pivot sequence used in solving the superdiagonal block in each cluster. The parts of the $\bar{W}$ and $\bar{Y}$ blocks that contribute to the reduced system are denoted by $\bar{W}_R$ and $\bar{Y}_R$, each of which is $(c+1)\times(c+1)$. In these terms, the reduced system is as shown in Figure 9.

The next step again is to solve the reduced system. Now, however, the reduced system is of order $P(c+1)$ as opposed to $2(P-1)(c+1)$. Not only is the order of the system reduced but it is denser and thus results in fewer wasted operations on zeros when solved as a full matrix. Finally, the remaining variables are solved for by backsubstitution in parallel as done before.

The experiments done above using BTD were repeated using the BBBD strategy. Results are shown in Figure 10. The Am-

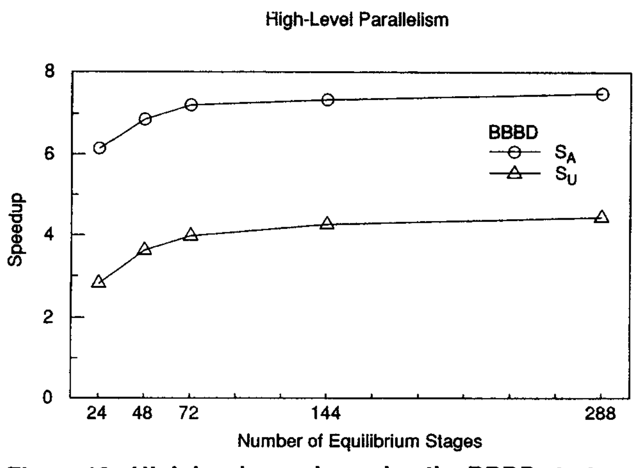

Figure 10. High-level speedup using the BBBD strategy.

dahl's law speedup S_A now averages about 7.0, which corresponds to a fraction parallelized f of about 98%, and on the largest problem it approaches 7.5, or 99% parallelized. Clearly, by decreasing the size of the reduced system a significant improvement in performance has been obtained. However, overhead factors and other inefficiencies as discussed above remain. At this level of parallelization, even very small changes in the fraction parallelized will have a significant effect on performance. Thus, in comparison with the BTD strategy, S_A and S_U for BBBD increase more dramatically with problem size, as the workload per processor increases and the relative overhead cost decreases.

There are many variations of this reformulation strategy that could be used. For instance, in the example being used, instead of replacing the balances on stages 5, 9 and 13 with balances with respect to stage 1, they could be replaced with balances with respect to stage 4. This would lead to a spike in block column 4 instead of block column 1. This might be attractive if there was a sidestream product from any of the first four stages. Similar variations in the strategy could be used in the case of sidestream products in other locations, though this may lead to more than a total of P block columns of fill-in. Another variation on this reformulation strategy would be to write all the balances with respect to stage 1. This again leads to a reduced system as shown in Figure 9. In this case, however, the superdiagonal blocks that must be solved to obtain the reduced system will be block-bidiagonal, not block-tridiagonal. In general, problem reformulations based on balances around groups of stages rather than individual stages can be used to decrease the size of the reduced system and thus enhance parallelization. It should be noted that the choice of problem formulation may affect the convergence behavior in solving the nonlinear system and the numerical stability in solving the linear system. Though no adverse affects along these lines were observed in our tests, these issues have not been thoroughly addressed.

The advantages of using the BTD or BBBD strategy on this problem to implement a two-level hierarchical approach to parallelism can be seen by considering a hypothetical 64-processor system. If low-level parallelism alone is applied across all 64 processors and the observed speedup of 3.6 on eight processors is extrapolated to 64 processors using Amdahl's law, the result is a speedup of only 5.3. On the other hand, if high-level parallelism is applied across eight clusters of eight processors each and a useful high-level speedup of 4.5 is assumed, as in the largest test problem using BBBD, then the overall useful speedup is (4.5)(3.6) = 16.2, over three times what could be achieved using low-level parallelism alone.

In implementing this two-level strategy, a number of issues arise that have not been considered here. For example, in using BTD or BBBD, there is a tradeoff between the number of clusters used and the size of the reduced system that must be solved. For the low-level parallelism within clusters, there is an issue of whether to treat the superdiagonal blocks as BTD and use BGE to solve them, or to simply treat the entire block as full. If the superdiagonal blocks are small enough, the latter may be preferable. The results of decisions of this sort are likely to be hardware-dependent.

Concluding Remarks

Given the current availability of parallel computing technology and its likely importance in the future, the development of strategies to most effectively exploit the capabilities of these machines is a timely problem. The familiar block-tridiagonal matrix structure that arises in multicomponent, multistage separation problems can be used to good advantage in this regard, by using the conventional block-Gaussian elimination solution procedure to exploit low-level parallelism. It has also been demonstrated how this structure can be used to advantage in exploiting high-level parallelism. However, by reformulating the problem so that a bordered-block-bidiagonal superstructure is obtained, further improvements in the use of high-level parallelism can be achieved. Strategies of this sort for using high-level parallelism permit the use of a two-level hierarchy of parallelism and thus lead to substantial improvements in computational performance on parallel machines.

Acknowledgment

This work was supported in part by the National Science Foundation under grant DDM-9024946. We also acknowledge the support of the Center for Supercomputing Research and Development and the National Center for Supercomputing Applications, both at the University of Illinois.

Notation

c	=	number of components
$\underline{E}_n$	=	energy balance around stage n
E_n	=	energy balance around stages 1 to n
f	=	fraction of utilized code performed in parallel (Amdahl's law)
$f_{n,m}$	=	molar feed rate of component m to stage n
h_n	=	enthalpy flow rate from stage n due to liquid
$h_{f,n}$	=	enthalpy flow rate to stage n due to feed; also heat load or loss for tray n
H_n	=	enthalpy flow rate from stage n due to vapor
$K_{n,m}$	=	equilibrium vaporization constant for component m on stage n
L_n	=	total liquid molar flow rate from stage n
$L_{n,m}$	=	liquid molar flow rate of component m from stage n
$\underline{M}_{n,m}$	=	material balance for component m around stage n
$M_{n,m}$	=	material balance for component m around stages 1 to n
N	=	number of stages
N_C	=	number of stages per cluster
P	=	number of processors; or number of clusters in a two-level hierarchy of processors
P_C	=	number of processors per cluster
$Q_{n,m}$	=	equilibrium relationship for component m on stage n
s_n	=	liquid sidestream split ratio from stage n
S_A	=	Amdahl's law speedup
S_U	=	useful speedup relative to standard serial method
S_n	=	vapor sidestream split ratio from stage n
T_n	=	temperature on stage n
V_n	=	total vapor molar flow rate from stage n
$V_{n,m}$	=	vapor molar flow rate of component m from stage n
η_n	=	Murphree efficiency of stage n based on vapor phase

Literature Cited

Berry, M. W., and A. Sameh, "Multiprocessor Schemes for Solving Block Tridiagonal Linear Systems," *Int. J. Supercomputer Appl.*, 2(3), 37 (1988).

Dongarra, J. J., "Performance of Various Computers Using Standard Linear Equations Software," Report CS-89-85, Computer Science Dept., Univ. of Tennessee, Knoxville, (Jan. 23, 1993) (an up-to-date Postscript copy of this report can be obtained on-line from NETLIB by sending the message *send performance from benchmark* to netlib@ornl.gov).

Dongarra, J. J., and A. H. Sameh, "On Some Parallel Banded System Solvers," *Parallel Comput.*, **1**, 223 (1984).

Erisman, A. M., R. G. Grimes, J. G. Lewis, W. G. Poole, Jr., and H. D. Simon, "Evaluation of Orderings for Unsymmetric Sparse Matrices," *SIAM J. Sci. Stat. Comput.*, **8**, 600 (1987).

Gustafson, J. L., "Reevaluating Amdahl's Law," *Comm. ACM*, **31**, 532 (1988).

Henley, E. J., and J. D. Seader, *Equilibrium-Stage Separation Operations in Chemical Engineering*, Wiley, New York (1981).

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

Sameh, A., "On Two Numerical Algorithms for Multiprocessors," *Proc. NATO Adv. Res. Workshop on High-Speed Comput.*, Series F, Computer and Systems Sciences, Springer-Verlag, New York (1983).

Vegeais, J. A., and M. A. Stadtherr, "Parallel Processing Strategies for Chemical Process Flowsheeting," *AIChE J.*, **38**, 1399 (1992).

Zitney, S. E., and M. A. Stadtherr, "Frontal Algorithms for Equation-Based Chemical Process Flowsheeting on Vector and Parallel Computers," *Comput. Chem. Eng.*, **17**, 319 (1993).

Manuscript received Feb. 12, 1993, and revision received May 26, 1993.