Symposium Series No. 92

PROCESS FLOWSHEETING ON SUPERCOMPUTERS

Mark A. Stadtherr and James A. Vegeais
Chemical Engineering Department, University of Illinois, Urbana, Ill.

SUMMARY

The coming availability of relatively cheap supercomputing power will have a major impact in the areas of process flowsheeting and process control. However, because supercomputers differ architecturally from conventional machines, the computational strategies that work well on a conventional machine may not be well suited to the supercomputer, and this seems to be the case in process flowsheeting. This paper considers the development of sparse matrix strategies that are suitable for equation-based flowsheeting, and that take best advantage of supercomputer architecture. An initial appraisal of three such strategies is performed.

INTRODUCTION

A new challenge and opportunity for those working in the area of process simulation, design, and optimization is the increasing availability of supercomputers. These machines differ architecturally from today's "conventional" large mainframe computers, and have the potential to provide very large increases in computational speed relative to the conventional machine. However the amount by which speed can be increased depends on the problem to be solved, on how that problem is formulated, and on what strategies are used to solve the problem. If its architecture is not well exploited, a doubling of speed is all that might be expected; if fully exploited, speed may be increased by a factor of twenty or more. This level of performance comes at a price upwards of $10 million, however.

While recent years have seen an explosion in microcomputer technology, with constant improvements in speed and memory, this has generally not been seen on the other end of the computer size spectrum. Machines ranging from superminis (e.g., the VAX 11/780), through fast mainframes (e.g., the CDC CYBER series, or array processors combined with a VAX), to supercomputers (e.g., the Cray-1) have been on a relatively constant plateau of speed and memory size for several years, despite advances in chip technology. Indications are that this is soon to change, at least with respect to supercomputing. Supercomputers two orders of magnitude faster than current supercomputers seem likely by the end of the decade. Perhaps of even more impact is that the technology now exists to mass-produce supercomputers comparable to the current Cray-1 for under $1 million each. Nobel laureate Kenneth Wilson of Cornell University [1] expects that competition will cause prices for these mass-produced machines to eventually drop to not much more than the prices of the widespread superminis. An explosion in supercomputing technology seems to be on the horizon.

The ready availability of relatively cheap supercomputing power will have a major impact in the areas of process flowsheeting and process control, provided that practitioners are able to take advantage of the different architecture of these machines. From the standpoint of steady-state process flowsheeting and optimization, the use of supercomputing will provide truly interactive design capabilities, most likely involving sophisticated graphics interfaces for input and output. The engineer will be able to make a design change in a complex process and will almost immediately be able to see the simulated result. In terms of actual wall-clock time, such simulations today too often require several minutes even on a relatively fast conventional machine, and perhaps several hours on an engineer's personal micro. Supercomputing will thus mean tremendous increases in the productivity of the design engineer. The supercomputer will also have a major impact on the ability to do dynamic process simulations and design control systems. Real-time, or faster-than-real-time simulations of complex process units or entire plants will be routine. Dynamic simulations of refinery units are apparently already being done using supercomputers in at least one company. While such off-line simulations can be used to improve control system design and plant operability, on-line simulations using supercomputers may provide a capability for on-line optimization of complete plant operations.

It is important to emphasize that computational strategies that work well on the conventional machines may not be the best on a supercomputer. For instance, when Duerre and Bumb [2] implemented the ASPEN flowsheeting system on a Cray-1, they found that it ran only two to three times faster than on an IBM 370, which indicates that little advantage was taken of the supercomputer architecture. In this paper we deal with the solution of process flowsheeting problems on supercomputers, concentrating on the equation-based approach, and on sparse matrix strategies for implementing it that take best advantage of the supercomputer's architecture.

BACKGROUND

We begin this section by briefly discussing some of the basic aspects of supercomputing. A more detailed introduction to the area of supercomputers can be found in a recent article by Levine [3].

The first machine generally regarded as a supercomputer was the ILLIAC IV, a one-of-a-kind machine originally installed in 1972 but now retired. There are now two main families of supercomputers available commercially, the Cray-1 and it descendents, and the CYBER 205, whose descendents will be coming from ETA Systems. Japanese supercomputers are available in Japan, and will undoubtedly soon have a major impact on the international marketplace.

Supercomputers differ architecturally from conventional computers. The most significant architectural difference is the ability of the supercomputer to perform several computational steps concurrently, or in parallel. On the other hand, conventional "sequential" computers typically must carry out steps one at a time, though the fastest sequential computers usually include some rudimentary form of "pipelining", a concept discussed below.

Machines such as the Cray-1, CYBER 205, and ILLIAC IV use a form of parallelism typically known as vector processing; thus these machines are

often called vector machines or vector processors. The basic idea involved in designing vector machines is to speed up the execution of vector operations. One obvious way to do this is to include vector instructions in the instruction set used by the instruction processor. More important however, is the use of "multiprocessing" and "pipelining"; both involve provisions for performing parts of the overall vector operation concurrently.

Pipelining is perhaps best explained using the assembly-line analogy [3]. A floating point operation involves several steps. Without pipelining, all the steps needed to complete one operation would be performed before starting the first step of the next operation. Thus in effect the computer works on one operation at a time. On the other hand, to perform a particular operation in a highly pipelined computer, there are several "stations", each of which performs only one part of the overall operation. Since, as in an assembly line, all stations are working concurrently, the computer can work on more than one similar operation at the same time. The Cray-1 and CYBER 205 are both highly pipelined machines, though each operates somewhat differently. For instance, the CYBER 205 operates most efficiently when operating on very long vectors, while the Cray-1 can operate efficiently even on relatively short vectors.

Multiprocessing basically involves putting several (64 in the case of the ILLIAC IV) data processors in parallel. Thus the ILLIAC IV could add two length-64 vectors in about the same time as one scalar addition, since all 64 scalar addition operations needed to add the vectors could be done concurrently. The Cray X-MP will ultimately employ up to sixteen processors in parallel, each highly pipelined.

To date the most common use of supercomputers has been in the numerical simulation of continuous fields, with applications in areas such as aerodynamics, metereology, and petroleum reservoir simulation. Such problems involve the solution of systems of partial differential equations by discretization. A key subproblem in this process, and one that the supercomputer is very adept at solving, is the solution of a large sparse system of linear equations, usually with specific structural and/or numerical properties, such as symmetry, bandedness, and positive definiteness, that can be readily exploited. The solution of a large sparse system of linear equations is also a key step in the equation-based approach for process flowsheeting. This would suggest that, with the right techniques for solving the sparse matrix problem, the equation-based formulation of the process flowsheeting problem would be well suited to the supercomputer.

A considerable amount of research on solving systems of linear equations using vector processors has concentrated on idealized computers, such as a computer composed of an infinite number of independently operating processing units. Typically this involves the assumption that there are never any internal conflicts of any kind. This research, while finding some bounds on computation times, cannot be directly applied in practice to real general-purpose supercomputers.

Research dealing with the solution of linear equation systems on supercomputers has dealt both with general systems of equations having no regular structure, and with highly structured systems such as those with a tridiagonal or block-diagonal form. Process flowsheeting matrices do not have such a highly structured form, and in general require the use of

general-purpose linear equation solvers. Nevertheless flowsheeting problems do have an easily recognizable block structure that can be exploited to tailor the general-purpose routines for application to flowsheeting [4,5]. General-purpose routines however are not easily "vectorizable", i.e., they are not able to take much advantage of the vector processing capability of the supercomputer. This is clearly seen in some comparisons conducted by Duff and Reid [6]. A general-purpose full matrix routine, compiled and executed on a Cray-1, was found to be roughly 20 to 30 times faster than the same routine compiled and executed on an IBM 3033. On the other hand, when the same comparison was made using a general-purpose sparse matrix routine, it was found to be only about two times faster on the Cray-1, which appears to be due simply to the fact that the scalar speed of the Cray-1 is roughly twice that of the IBM 3033. Clearly very little vectorization of the sparse matrix routine was possible, largely, it appears, because of the amount of indirect addressing in sparse matrix codes. Quoting from [6], "This is very disappointing since there is no easy fix which can give us better vectorization. We are therefore forced to rethink our sparse matrix algorithms with the Cray architecture in mind." Similarly if we are to make the most effective use of supercomputers in equation-based flowsheeting, we must rethink the sparse matrix strategies used.

A variety of ideas [e.g., 7-10] have been suggested for more effectively using supercomputers in general-purpose linear equation solving. In the context of equation-based flowsheeting and optimization, there are three such ideas that appear to be of some promise, namely the use of a frontal approach [e.g., 11], the use of block-oriented methods [e.g., 12], and use of methods involving a search for contiguous nonzero elements [e.g., 13]. In each case the basic idea is to treat parts of the sparse matrix as full to increase the amount of vectorization. Another consideration is the central memory requirement. Although these machines have large central memories, it may still be necessary on very large problems to page parts of the problem repeatedly in and out of central memory. This represents a potential bottleneck in the computation. We discuss in this paper the use of these three ideas in developing methods for solving flowsheeting matrices, and present a critical appraisal comparing the different approaches.

SPARSE MATRIX STRATEGIES

Block-Oriented Approach

One approach to the solution of sparse flowsheeting matrices is the block-oriented approach [12]. The indirect addressing problem that slows down the general-purpose sparse equation solvers is eliminated by treating parts of the sparse matrix as if they were dense blocks of nonzeros. The blocks are so located that the system can be solved by performing block Gaussian elimination. The blocks are given descriptors that identify the size of the block and its position in the matrix. The elements in the block are then stored in a regular way so that no indices are required for each nonzero element. Because the blocks are considered full, the location of all elements is described completely by the block descriptors. This allows the data to be easily accessed since elements in a block are stored contiguously. The system is then solved by block Gaussian elimination. Because of the regular way the matrix is stored, the operations performed in this approach are vector operations.

One advantage of this approach is that it can exploit the natural block structure of the matrices that arise in process flowsheeting. This is because normally, equation-based flowsheeting packages are set up to generate the entire set of equations for each unit operation simultaneously. The block descriptors could be generated at the same time the equations for the blocks are generated. The only preprocessing necessary would be to locate blocks where fill-in will occur and generate the descriptors for this block fill-in.

There are drawbacks to this method as well, however. First, a high execution rate may be misleading with this method. Although a huge number of operations per second may be performed, many of these operations are on zero elements and do not have to be done. The result could be a program that runs at a high rate of speed but that takes longer to run than a slower program. Another drawback is the difficulty involved in pivoting to maintain numerical stability. In order to perform threshold pivoting it is necessary to search through the blocks to see if the blocks contain elements in the same row. If it becomes necessary to exchange columns for reasons of numerical stability, all the elements of the two exchanged columns must be exchanged, which could involve transfers among a number of the blocks, and the formation of new blocks if the exchange affects the location of the fill-in that will arise when these blocks are eliminated. This could slow down the overall performance of the block-oriented solver considerably.

Continuous Backsubstitution Approach

Another approach to increasing the amount of vectorization when process flowsheeting matrices are solved on supercomputers is to locate and perform vector operations on contiguous nonzero elements when they arise. Though not originally developed with supercomputers in mind, the continuous backsubstitution algorithm (CBS) is a solution method that tries to exploit the presence of contiguous nonzero elements, and which could be used advantageously on the supercomputer. A description of this method is given by Stadtherr and Wood [5]. The CBS algorithm limits fill-in in the matrix to certain columns, called spike columns. Because these spike columns normally become completely filled-in in the CBS algorithm, they can be stored as a full vector. This means that indirect addressing in the spike columns can be eliminated and that computations with the spike columns can be done as vector operations.

One advantage of this method is that it operates almost exclusively on nonzeros. This method also limits the amount of fill-in that can occur. Unneeded operations are not performed on zero elements, so there is no tradeoff between more operations and a faster rate of operations, as there is in the block-oriented approach.

One disadvantage to this method is that the increased speed only occurs in the spike columns. The elements below the diagonal are still indirectly indexed and cannot be operated on as vectors. Another disadvantage of this method is that, when column exchange is necessary in order to maintain numerical stability, the spike column must be put into indexed form and the pivot column must be "unindexed" into a contiguous vector.

Frontal Approach

Another way of taking advantage of vectors is to use a variation of the frontal approach. The frontal approach [11] was developed for use in finite element problems. It takes advantage of the fact that each variable only appears in a few equations and that pivoting on a variable will only affect a small number of equations and variables. This can be exploited by storing only a small submatrix, called the frontal matrix, at any time during the solution of the sparse matrix. Basically, this method is designed to take advantage of a banded type of matrix structure.

In the original frontal codes, the input is by finite element. Equations and variables enter the front as their elements are encountered. As soon as all the elements containing a particular variable are in the frontal matrix, that variable may be used as a pivot. Since we are not trying to solve finite element problems, we use a somewhat more general-purpose approach in our version of the frontal approach. In this version, the input is by equation. Variables enter the frontal matrix the first time they are encountered in an equation. After the equation that contains the last occurrence of the variable has been entered into the frontal matrix, the variable is eliminated.

As an example, in order to solve the matrix

	1	2	3	4	5
1	x		x	x	
2				x	x
3	x	x	x		
4	x	x			x
5		x			x

the first equation is entered into the frontal matrix. In order to do this, variables 1, 3, and 4 must be entered into the front. The frontal matrix now looks like:

	1	3	4
1	x	x	x

None of these elements may be pivoted on because none of the variables are completely within the frontal matrix. Thus we proceed to the second equation. Because the second equation contains the fifth variable, variable 5 must enter the frontal matrix. The frontal matrix now looks like:

	1	3	4	5
1	x	x	x	
2			x	x

The fourth variable is now completely in the frontal matrix. A nonzero in that column is now chosen as a pivot. After eliminating the other nonzeros in the column the pivot row and column are removed. If the second equation is chosen as the pivot row the following frontal matrix is obtained:

	1	3	5
1	x	x	f

The f here represents a zero element that has become a nonzero element during elimination (a fill-in). Note here that the first equation could also have been chosen as the pivot row. Now the third row is entered into the frontal matrix. Since the third equation contains the second variable, variable 2 must be entered into the frontal matrix. The matrix now looks like:

	1	3	5	2
1	x	x	f	
3	x	x		x

The third variable is now completely in the frontal matrix and may now be chosen as a pivot. Eliminating the third equation leaves the matrix:

	1	5	2
1	x	f	f

The fourth equation is now entered, and the frontal matrix is:

	1	5	2
1	x	f	f
4	x	x	x

The first variable may now be used as a pivot. Pivoting on the first equation and entering the final equation to the frontal matrix results in the final frontal matrix

	5	2
4	x	x
5	x	x

which is now solved by simple Gaussian elimination.

The major advantage of this method is that the frontal matrix is fairly dense and can be operated on as a full matrix. This allows the use of vector operations during elimination. Another advantage of this method is that the amount of storage necessary for the frontal matrix and other needed arrays is small. In fact, except for very small matrices, the storage required is less than that required for the original matrix and much less than the storage required for the CBS algorithm. This can also be seen from the data in Table 1. The Table contains the total amount of array storage necessary for the solution of flowsheeting matrices using the frontal method. The matrices used were among the examples used by Stadtherr and Wood [4,5]. The frontal method requires as little as a tenth of the storage needed by the CBS algorithm.

The frontal method takes advantage of the fact that in flowsheeting matrices elements tend to be concentrated in a band near the diagonal, though without the intraband structure of matrices of the sort arising in finite element problems. For a banded matrix the total amount of array storage needed for the frontal method is N + 2B*B + 9B - 16, where N is the number of equations and B is the bandwidth of the matrix. The array needed for the frontal matrix is B*(2B - 1). While the matrices that arise in process flowsheeting have a predominately banded structure, however, they do have nonzero blocks off the diagonal band that must also be considered.

To do this, we note that another desirable form for the frontal approach is the triangular form. This is desirable because as each row enters the matrix there is one pivot variable available. Because of this the frontal matrix need only be of dimension 1 x N and the total array storage needed is only 6N + 2. Flowsheeting matrices are not simply triangular of course. Some nonzeros will always be found above the diagonal, in the spike columns. But if we can keep those nonzeros relatively close to the diagonal (i.e., in a band) then the frontal matrix can be kept reasonably small. For a triangular matrix with spikes, the storage needed for the front is N times the maximum number of local spikes (this term is defined in [4] and is related to the proximity to the diagonal of the topmost nonzero in a spike). Also, since the triangular portion of flowsheeting matrices is not full, the storage needed for the frontal method will actually be less than N times the maximum number of local spikes.

It is interesting to note here that the desired row-column ordering for the frontal method is the opposite of the order for the CBS method. With the CBS method, it is desirable to have small rows first and small columns last. For the frontal method it is desirable to process small columns first and small rows last. Because of this, matrices to be solved by the frontal approach may be fairly efficiently ordered by taking an a priori reordering method intended to reduce fill-in in general sparse matrix solvers and reversing the order of the rows. This can be seen in the results in Table 2, which shows the size of the frontal matrix required using several different sparse matrix reordering algorithms on the six example matrices used above. A description of the reordering methods SPK1, SPK2, HP30, and BLOKS can be found in [4]. The notation REV in this table indicates the use of a reverse ordering. It can be seen from example 1 in Table 2 that the reverse ordering results in a much more compact frontal matrix than the unreversed ordering. This is also true for the other example problems, though this data is not listed. It can also be seen that in all cases the BLOKS reordering performs very well. This reordering algorithm attempts to keep the inherent block structure associated with process flowsheeting matrices. Interestingly, this reordering is also the fastest reordering algorithm of those tested.

Because a variable does not enter the frontal matrix until it is encountered in an equation, it does not matter at all in what order the variables occur. The order of the equations is the only thing that will affect the size of the frontal matrix. Because of this it may be possible to find a reordering algorithm that is more efficient than the current methods, and which produce a better equation ordering for the frontal approach.

Because the frontal matrix is operated on as a full matrix and because pivoting only affects those elements in the frontal matrix, it is not necessary to limit pivoting to threshold pivoting. Instead, partial pivoting may be performed in the column with no penalty of additional fill-in. This may help the numerical stability of the solution algorithm.

A disadvantage to this method is that, like the block-oriented approach, operations are done on zero elements. This can be seen in Table 3, which shows the percentage of the total operations that are performed as vector operations, and the percentage that are wasted due to operations on zeros. Though about 90% of the operations performed are vector

operations, it can also be seen that about 30% of the operations are unnecessary. So while the operations are performed at a much faster rate, some of this speed is wasted on unnecessary operations.

CONCLUSIONS

In order to fully exploit the computing potential of supercomputers in equation-based process flowsheeting and optimization it is necessary to develop new strategies for the solution of the sparse matrices that arise. Three approaches--the block-oriented approach, continuous backsubstitution, and the frontal approach--appear to have some promise. Each exploits a different feature of the sparse flowsheeting matrix to allow vectorization to occur during the solution of the matrix. As discussed in the initial appraisal above, the three approaches have their advantages and disadvantages. The potential difficulties involved in pivoting in the block-oriented approach and continuous backsubstitution have caused us to concentrate initially on the frontal approach. A more detailed appraisal of the block-oriented approach and continuous backsubstitution is still needed. Our initial appraisal of the frontal approach makes it appear quite attractive. Its performance could be further enhanced by the development of reordering algorithms for flowsheeting matrices that are designed to produce an ordering tailored specifically to the frontal approach.

REFERENCES

1. K. Wilson, Supercomputing: Cooperation with industry, SIAM News, 18(5), 6 (1984).

2. K. H. Duerre and A. C. Bumb, Implementing ASPEN on the Cray computer, Los Alamos Report LA-UR-81-3528, Los Alamos National Laboratory, Los Alamos, New Mexico (1981).

3. R. D. Levine, Supercomputers, Scientific American, 246(1), 118 (1982).

4. M. A. Stadtherr and E. S. Wood, Sparse matrix methods for equation-based chemical process flowsheeting--I. Reordering phase, Comput. Chem. Eng., 8, 9 (1984).

5. M. A. Stadtherr and E. S. Wood, Sparse matrix methods for equation-based chemical process flowsheeting--II. Numerical phase, Comput. Chem. Eng., 8, 19 (1984).

6. I. S. Duff and J. K. Reid, Experience of sparse matrix codes on the Cray-1, Comput. Phys. Comm., 26, 293 (1982).

7. W. P. Peterson, Vector Fortran for numerical problems on Cray-1, Comm. ACM, 26, 1008 (1983).

8. D. A. Calahan, Performance of linear algebra codes on the Cray-1, SPE J., 21, 558 (1981).

9. D. A. Calahan, Direct solution of linear equations on the Cray-1, Cray Channels, 3(2), 2 (1981).

10. D. Heller, A survey of parallel algorithms in numerical linear algebra, SIAM Review, 20, 740 (1978).

11. I. S. Duff and J. K. Reid, The multifrontal solution of unsymmetric sets of linear equations, AERE Harwell Report CSS 133 (1983).

12. D. A. Calahan, A block-oriented equation solver for the Cray-1, SEL Report No. 136, University of Michigan, Ann Arbor (1980).

13. D. A. Calahan and W. G. Ames, Vector processors: Models and applications, IEEE Trans. on Circuits and Systems, CAS-26, 715 (1979).

TABLE 1.

EXAMPLE	EQUATIONS	NONZEROS	STORAGE-FRONTAL	STORAGE-CBS
1	372	3253	5872	12453
2	814	7448	10190	*
3	1060	6254	3973	28040
4	1564	9369	4538	39889
5	2224	13577	6038	62487
6	2878	17772	7427	74530

*INSUFFICIENT STORAGE

TABLE 2.

EXAMPLE	EQUATIONS (NONZEROS)	REORDERING METHOD	MAXIMUM EQUATIONS	MAXIMUM VARIABLES
1	372 (3253)	SPK1	160	160
		REV SPK1	55	160
		SPK2	160	160
		REV SPK2	44	160
		REV BLOKS	31	160
		REV HP30	40	166
2	814 (7448)	REV SPK1	87	244
		REV SPK2	74	278
		REV BLOKS	47	263
		REV HP30	51	267
3	1060 (6254)	REV SPK1	203	490
		REV SPK2	25	155
		REV BLOKS	20	161
		REV HP30	30	182
4	1564 (9369)	REV SPK1	168	428
		REV SPK2	39	183
		REV BLOKS	17	140
		REV HP30	30	198
5	2224 (13577)	REV SPK1	174	466
		REV SPK2	49	245
		REV BLOKS	17	180
6	2878 (17772)	REV SPK1	142	413
		REV SPK2	43	299
		REV BLOKS	17	215

TABLE 3.

EXAMPLE	EQUATIONS	% VECTOR OPERATIONS	% WASTED OPERATIONS
1	372	93.8	30.9
2	814	95.8	38.9
3	1060	91.6	32.5
4	1564	89.4	27.7
5	2224	89.9	27.6
6	2878	88.8	31.2