IMPLEMENTING NESTED DISSECTION

By

W. Morven Gentleman
Department of Computer Science
University of Waterloo
Waterloo, Ontario

Research Report CS-82-03
March, 1982

Implementing Nested Dissection

W. Morven Gentleman

University of Waterloo
Computer Science Department
Waterloo, Ontario, Canada

I. THE PROBLEM

When solving sparse linear systems by Gaussian elimination, the principal concern is to minimize fillin, the occurrence of nonzeros in the matrix factors where the original matrix had zeros. More fillin requires more storage, as sparse matrix codes explicitly represent only nonzeros, and more fillin is also strongly correlated with requiring more arithmetic, since operations on elements known to be zero can often be avoided. The amount of fillin in a particular linear system depends on the order in which the unknowns are eliminated when solving the system. For a positive definite symmetric system, if symmetry is preserved throughout the elimination, only half the nonzeros need be represented, and numerical stability is assured for any ordering.

A particularly important class of sparse linear systems arise from finite element problems in two dimensions. In such a problem, a region of the plane is subdivided by a mesh into subregions called finite elements. Corresponding

to each vertex of the mesh is a value, and an equation relating this value to the values at other nodes on the edges of elements which this node is on the edge of. Typically the values on the boundary of the original region are known, and the values in the interior of the original region are unknown, giving the linear system to be solved.

For this class of problem, it has been shown that the ordering known as nested dissection (1,5,6,7,8,) is asymptotically optimal, both with respect to fillin and with respect to the amount of arithmetic to be performed, in the sense that both these are of the same order of N, the number of unknowns, as lower bounds which have been proved to hold for any ordering.

In general terms, a nested dissection ordering is obtained by finding a small set of nodes C, called a separator, such that the remaining nodes are divided into two disjoint sets A and B, where there are no nonzero coefficients coupling the values corresponding to nodes in set A with values corresponding to nodes in set B. Ideally, the number of nodes in the separator is as small as possible, and the number of nodes in each of the sets A and B are roughly equal. The ordering sought numbers the nodes of the separator after the nodes of sets A and B. Recursively, separators are then found that partition each of the sets A and B, and these separators are numbered after the other nodes in the sets A and B, etc.

II. THE ALGORITHM

At this point we will restrict our attention to regions which are rectangles, and meshes which are rectangular grids. Ultimately we want to consider a grid with 2^m+1 nodes on each side, so the nodes with unknown values form a 2^m-1 by 2^m-1 array, but the presentation is somewhat simpler if we initially consider the less efficient situation where there are unknowns at all nodes, so the nodes with unknown values form a 2^m+1 by 2^m+1 array. We will order the perimeter nodes last, so the nested dissection only applies to the ordering of the interior nodes. The separators used will effectively be the "+" separators of George, but considered as two parts: part V, which is the vertical column of nodes that separates the square array of nodes with unknowns into two identical arrays, each roughly twice as high as wide, and part H, which is the horizontal row of nodes that separates one of these rectangular arrays of nodes with unknowns into two identical square arrays. Figure 1 illustrates these separators for m=3.

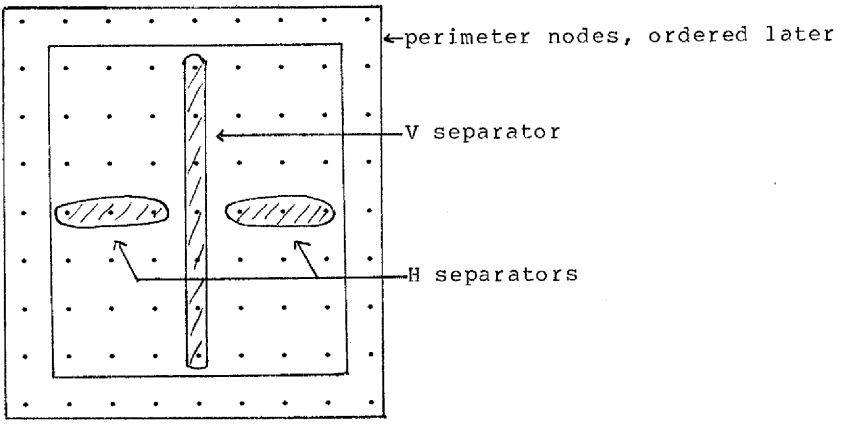

Figure 1 H and V separators

The V separator must be ordered after the H separators. Notice that the nodes not in the perimeter, V separator, or H separators form four arrays, each $2^{m-1}-1$ by $2^{m-1}-1$, for which H and V separators can be found. The perimeters of these smaller square arrays are made up of parts of the original perimeter, and H separators or V separator of larger arrays which, by definition, are ordered later, so the situation is exactly as existed for the 2^m-1 and by 2^m-1 array. The procedure can thus be repeated recursively, subdividing each array down until m=1, where the interior set is a single node. The complete set of separators for m=4 is shown in Figure 2: all isolated nodes are ordered first, all V separators before any H separator of the same length, all shorter separators before any longer separator, and finally the perimeter last.

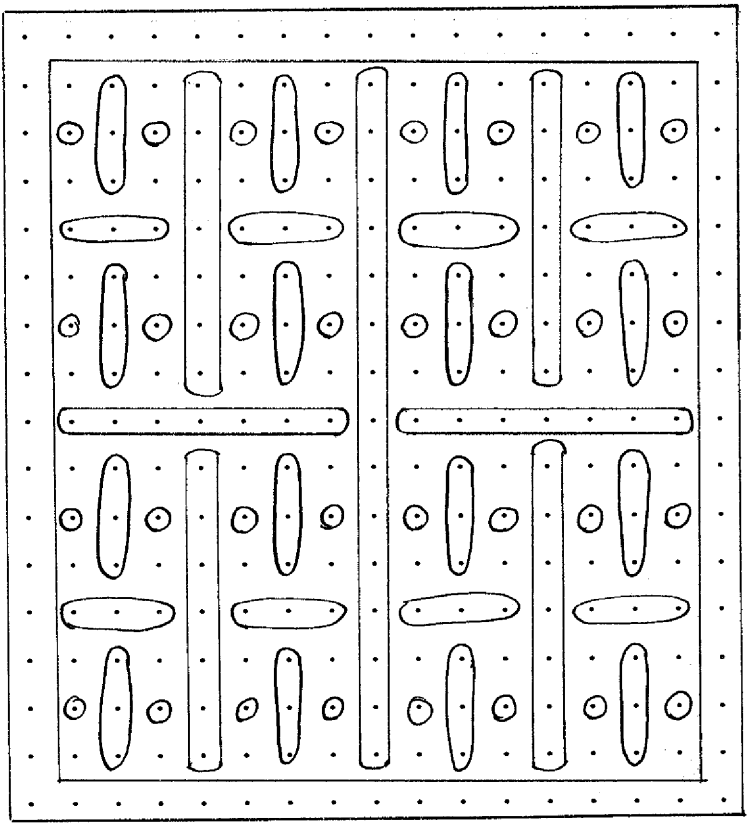

Figure 2 All separators for m=4

The foregoing description has been top down, describing how the set of nodes is partitioned to obtain the desired ordering. An equivalent bottom up description can be given describing how adjacent "generalized" elements (starting with the original finite elements) can be merged into larger "generalized" elements by eliminating the unknowns corresponding to the H or V separators which form their common edge. These generalized elements have nodes along their

edges as well as at the corners, and indeed at intermediate stages of the merge have interior nodes along the separator being eliminated.

This bottom up description performs the elimination for a 2^m+1 by 2^m+1 grid in m stages (the first of which is degenerate), followed by the elimination for the 2^m by 2^m generalized element which is the original region with nodes only on the perimeter. For s=2,3,...,m, stage s consists of forming a 2^s by 2^s generalized element from four 2^{s-1} by 2^{s-1} generalized elements by first eliminating the H separator which forms the common edge between two of the smaller elements juxtaposed vertically, and then eliminating the V separator which forms the common edge between two of thr newly formed rectangular elements. Figures 3 and 4 illustrate the H separator elimination phase and the V separator elimination phase for stage s=3.

Stage s=1 is slightly different from the subsequent stages, because although it consists of forming a 2 by 2 generalized element from four 1 by 1 elements, there are not distinct H and V separator phases, as there is only one node whose unknown is eliminated at this stage.

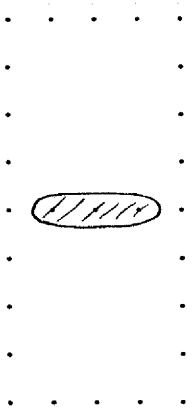

Figure 3 H separator elimination phase

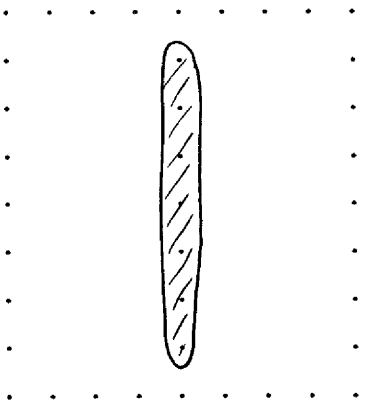

Figure 4 V separator elimination phase

Notice that for the 2^m+1 by 2^m+1 grid, at stage s the elimination of the H separator must be done for each pair of 2^{s-1} by 2^{s-1} elements, a total of $2 \times 2^{(m-s)} \times 2^{(m-s)} = 2^{2(m-s)+1}$ replications. Similarly the elimination of the V separator at stage s must be done for $2^{(m-s)} \times 2^{(m-s)} = 2^{2(m-s)}$ replica-

tions. Clearly the operations performed for each replication are identical, and we shall show below that they can be made independent. They thus can be done in parallel, indeed by an SIMD parallel computer.

In the foregoing discussion, we have frequently used the phrase "eliminating an unknown" from the system of equations. It is time to look closer at this. The basic identity that is referred to, since the system is assumed to be positive definite and symmetric, is equation (1).

$$\begin{bmatrix} L_i D_i L_i^T & U_i \\ U_i^T & B_i \end{bmatrix} = \begin{bmatrix} L_i & 0 \\ U_i L_i^{-T} D_i^{-1} & I \end{bmatrix} \begin{bmatrix} D_i & 0 \\ 0 & B_i - U_i L_i^{-T} D_i^{-1} L_i^{-1} U_i \end{bmatrix} \begin{bmatrix} L_i^T & D_i^{-1} L_i^{-1} U \\ 0 & I \end{bmatrix} \quad (1)$$

This identity yields an important interpretation when considered in the context of a system of linear equations like equation (2).

$$\begin{bmatrix} L_i D_i L_i^T & U_i \\ U_i^T & B_i \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} b_i \\ c_i \end{bmatrix} \quad (2)$$

A little manipulation yields the equations (3) and (4).

$$D_i L_i^T x_i = L_i^{-1} (b_i - U_i y_i) \tag{3}$$

$$(B_i - U_i L_i^{-T} D_i^{-1} L_i^{-1} U_i) y_i = (c_i - U_i L_i^{-T} D_i^{-1} L_i^{-1} b_i) \tag{4}$$

The interpretation of equation (4) is that a factorization of the leading submatrix can be used to update the rest of the matrix so that the unknowns not involved in the leading submatrix can be solved without reference to the unknowns involved in the leading submatrix. In this sense, the unknowns involved in the leading submatrix have been "eliminated". The factorization is the familiar square root free variant of Cholesky factorization, into a lower unit triangular matrix times a diagonal matrix times the transpose of the triangular matrix. Of course once the other unknowns have been found, the factorization can readily be used, as in equation (3), to find the unknowns involved in the leading submatrix. Usually the factors are saved until the end of the computation in order to do this, but Sherman (4) has observed that in sparse matrix computations it is often practical merely to rederive them when needed. The index i appears on all the symbols in equations (1) to (4) to remind us that the process is iterated, for the new equation (4) often has a submatrix whose factorization can be similarly exploited. Care must be taken in forming equations (3) and (4), that the matrix products and inverses be formed in the most economical way (5).

In our context, the interpretation lets us look quite

directly at the arithmetic which must be performed, the fill-in which occurs, and storage representations which are desirable at each stage in the elimination. By definition, each node on the original grid had an equation which had nonzero coefficients only for unknowns corresponding to other nodes on the same elements that this node is on. Consideration of Figures 3 and 4 show this remains true at each stage s=1,2,...,m where at each stage we are interested only in nodes (and their corresponding unknowns and equations) on the generalized elements relevant to this stage. The equations corresponding to nodes on a separator to be eliminated at this stage contain nonzero coefficients only for unknowns corresponding to nodes on the two generalized elements being merged. Neglecting for the moment the nodes just beyond either end of the separator, the remaining nodes have equations with nonzero coefficients corresponding to nodes on the generalized element they currently are on, but zero coefficients corresponding to the other nodes on the generalized element their generalized element is merging with. The elimination process adds a linear combination of the equations corresponding to each node in the separator to the equations corresponding to each node in the resulting generalized element, with the result that the zero coefficients referred to above become nonzero. Of course the equations at nodes on the perimeter of the new generalized element may have had other nonzero coefficients, cor-

responding to other generalized elements which these nodes are in, but these nonzeros are not affected by the eliminations for this separator.

Unfortunately, when we try to do all replications of stage s in parallel, we discover that adjacent generalized elements which are not being merged at this stage both need to update the coefficients in equations at nodes along their common edge corresponding to other nodes on their common edge. Most parallel machines cannot apply two or more updates simultaneously to a single stored quantity. A related problem occurs when we try to find a compact and regular storage structure, without pointers, for the equations: what is right for one generalized element is awkward for its adjacent neighbour. To address these problems, we consider equation (5) which is an elaborated version of equation (1).

$$\begin{bmatrix} L_1 D_1 L_1^T & 0 & U_1 \\ 0 & L_2 D_2 L_2^T & U_2 \\ U_1^T & U_2^T & B \end{bmatrix} =$$

$$\begin{bmatrix} L_1 & & \\ 0 & L_2 & \\ U_1^T L_1^{-T} D_1^{-1} & U_2^T L_2^{-T} D_2^{-1} & I \end{bmatrix}$$

$$\begin{bmatrix} D_1 & & \\ & D_2 & \\ & & B - U_1^T L_1^{-T} D_1^{-1} L_1^{-1} U_1 - U_2^T L_2^{-T} D_2^{-1} L_2^{-1} U_2 \end{bmatrix}$$

$$\begin{bmatrix} L_1^T & 0 & D_1^{-1} L_1^{-1} U_1 \\ & L_2^T & D_2^{-1} L_2^{-1} U_2 \\ & & I \end{bmatrix} \qquad (5)$$

The interpretation of this identity is that when two sets of unknowns are being eliminated, and they are coupled only in that each set is coupled to the unknowns not being eliminated, then the updates $U_1^T L_1^{-T} D_1^{-1} L_1^{-1} U_1$ and $U_2^T L_2^{-T} D_2^{-1} L_2^{-1} U_2$ can be computed simultaneously, and the actual serial updating of B can be done subsequently. If the computation of

the updates is sufficiently expensive, this serial step will be insignificant. During the computation of the updates, the elements of B effectively have a multiple representation.

In our context, we use this observation by storing for each generalized element the coefficients of the equation at each node in the generalized element that relate to unknowns at nodes in that generalized element. This means that the complete equation for any node will be stored in pieces, one piece with each generalized element which this node is in. It also means that the coefficients relating this node to other nodes on a common edge between two generalized elements will be the sum of two parts, one stored with each generalized element. These parts must be summed at the stage s when the node becomes part of the current separator and must be eliminated.

Another thing which must be done at each stage is reordering. As mentioned previously, the regularity of the nonzeros in the partial equations associated with each generalized element makes it possible, indeed advantageous, to store these coefficients compactly, without pointers or row or column indices. For example, Figure 5 is the same as Figure 3, but the arrow indicates an imposed ordering on the nodes, and the numbers indicate segments of edges of generalized elements. Figure 6 shows the nonzero elements at the start of the H separator elimination phase of that

stage. Figures 7 and 8 show the same for the V separator elimination phase of that stage.

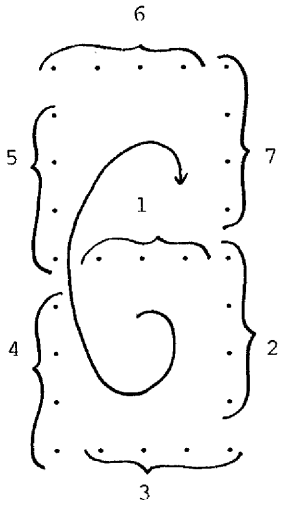

Figure 5 An ordering of the nodes at stage s=3, H separator elimination phase

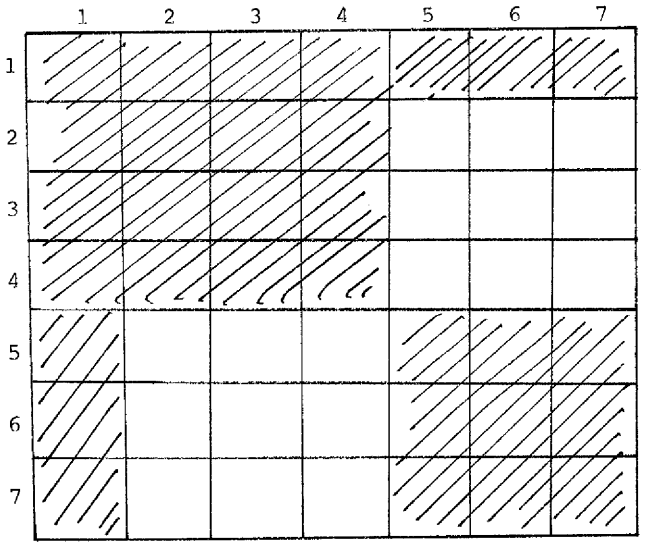

Figure 6 The 27 by 27 array of coefficients corresponding to Figure 5

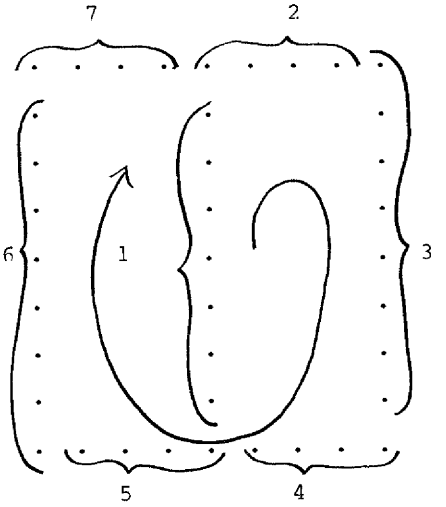

Figure 7 An ordering of the nodes at stage s=3, V separation elimination phase

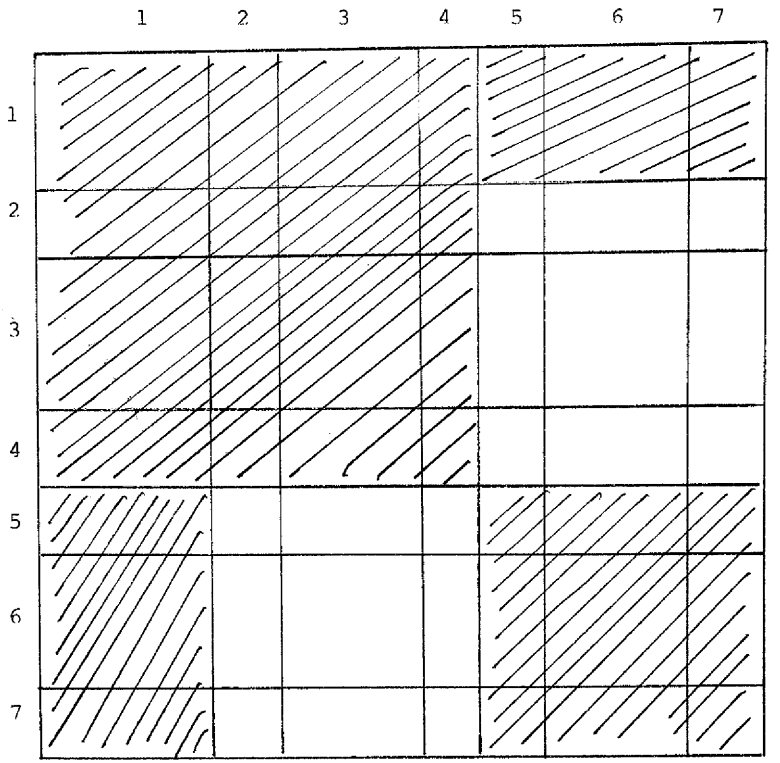

Figure 8 The 39 by 39 array of coefficients corresponding to Figure 7

Only the hatched regions are nonzero at the beginning of an elimination phase, except that the first row and column of the segments numbered 2 and 5 are nonzero across the entire array. The result of the elimination phase is to zero the segments numbered 1, which corresponds to the separator, and to fill in the rest of the array.

The implication of Figures 5 and 7, however, is that adjacent generalized elements number the nodes on their common

edge in opposite orders. Thus at the start of a phase, when summing the two partial representations of the equations at the nodes in the separator, the coefficients in each equation must be rearranged in order, and the equations themselves may need to be rearranged in order. Beyond this, it may be convenient to shift segments of coefficients or segments of equations to form coefficient arrays as in Figure 6 and 8, even though the order within the segment is correct.

It is possible that instead of the ordering of Figures 5 and 7, there might exist an ordering such that no rearrangement is required between phases. Such an ordering would have to be different for each generalized element, depending on the coordinates of the nodes in the original grid in something like the "bit reversed ordering" that is used with the Fast Fourier Transform. No such ordering has yet been found though, so this reordering is still necessary.

III. EXPLOITING PARALLELISM

We have already observed that one source of parallelism available to be exploited is that at any stage s and phase H or V all replications can be done simultaneously. The other source of parallelism available is the elimination procedure itself. For each generalized element, the elimination procedure is effectively a sequence of outer product updates to the arrays of Figure 6 or 8, one update for each node in the separator. Such outer product updates can readily be done in parallel, simultaneously updating each element of

the array. In general this requires one processing element per element of the array (although we shall show how this may be reduced) so our crudest estimate of the parallelism available at stage s is simply the number of replications times the number of elements in the array which is

for the H separator elimination phase

$$P_{H\ crude} = 2^{2(m-s)+1}\{7x2^{s-1} - 1\}^2$$
$$= 24.5x2^{2m} - 14x2^{2m-s} + 2x2^{2m-2s} \qquad (6)$$

for the V separator elimination phase

$$P_{v\ crude} = 2^{2(m-s)}\{10x2^{s-1} - 1\}^2$$
$$= 25x2^{2m} - 10x2^{2m-s} + 2^{2m-2s} \qquad (7)$$

for the special case s=1

$$P_{1\ crude} = 2^{2(m-1)}\{9\}^2$$
$$= 20.25x2^{2m} \qquad (8)$$

The first thing we observe from equations (6), (7), and (8) is that the two sources of parallelism play off against each other so that, to leading order, the total amount of parallelism available is about the same at all stages of the computation. This is even true for solving the system corresponding to the final generalized element, with nodes on the perimeter of the original region, as that system is of order $4x2^m$ and so has potential parallelism of $16x2^{2m}$. One

consequence of this is that if a parallel computer was able to take advantage of both kinds of parallelism, there would be little advantage in considering incomplete nested dissection (4), where the dissection process is stopped before the separators becomes too small, band elimination being used below that level. We shall come back later to the problem of how to reallocate processing elements between stages.

A related observation is that the active storage requirement, that is all the replications of the arrays of Figure 6 or 8, is roughly constant over the computation, and is less than 3 times the storage required for the original system of equations, 9 coefficients at each of 2^m+1 by 2^m+1 nodes. (These storage requirements do not assume symmetry, which will be discussed later). The partial factorization up to the end of stage s is inactive after stage s except for the backsolve. The number of coefficients in the partial factor up to the end of stage s, exploiting symmetry fully, is:

Size of partial factor

$$= 2^{2(m-1)} \times 9$$

$$+ \sum_{2}^{s} 2^{2m-2t+1} \times 2^{-1} \times 2^{t-1} \times (2^{t-1}-1)(7 \times 2^{t-1} - 1 + 6 \times 2^{t-1} + 1)$$

$$+ \sum_{2}^{s} 2^{2m-2t} \times 2^{-1} \times 2^{t} \times (2^{t-1}-1)(10 \times 2^{t-1} - 1 + 8 \times 2^{t-1} + 1)$$

$$= 7.75xsx2^{2m} - 11x2^{2m} + 11x2^{2m-s} \qquad (9)$$

For typical values of m, this is sufficiently large that it would probably have to be written to backing store, or as mentioned earlier it might be discarded and recalculated as needed, depending primarily on the relative cost of recomputation versus transfer to and from backing store.

Another thing we can observe from equations (6), (7), and (8) is what the cost of the multiple representation has been. For each adjacent pair of generalized elements, the equation at each node on their common edge doubly represent the coefficients corresponding to nodes on the common edge.

Excess store at start of H separator elimination

$$=2^{m-s+1}(2^{m-s} - 1)(2^{s-1} + 1)$$

$$+2^{m-s}(2^{m-s+1} - 1)(2^{s} + 1)$$

$$=2.5x2^{2m} + 6x2^{2m-s} + 4x2^{2m-2s}$$

$$-1.5x2^{m+s} - 2.5x2^{m} - 3x2^{m-s} \qquad (10)$$

Excess store at start of V separator elimination

$$=2x2^{m-s}(2^{m-s} - 1)(2^{s} + 1)$$

$$=2x2^{2m} + 4x2^{2m-s} + 2x2^{2m-2s}$$

$$-2x2^{m+s} - 4x2^{m} - 2x2^{m-s} \qquad (11)$$

Excess store at start of special case s=1

$$= 2x2^{m-1}(2^{m-1}-1)(3)^2$$

$$= 4.5x2^{2m} - 9x2^{m} \qquad (12)$$

So although the multiple representation initially adds about 23% to the storage requirements, in later stages this decreases to about 10% of the storage requirement. It is a cheap investment for the increased regularity and improved parallelism.

Returning to issue of exploiting symmetry, this has no effect on the number of sequential steps in the parallel solution, but does affect the amount of parallelism, or equivalently, the size of the active store. The arrays of Figure 6 and 8 are positive definite and symmetric, and so only the lower triangle, say, need be stored and operated on. The requirements of equations (6), (7) and (8) are readily recalculated, and are, of course, roughly halved. Unfortunately, this is not quite the appropriate story. For most SIMD parallel computers, even when an outer product update can be performed in a single operation, it is not convenient to operate on symmetric arrays where only the lower triangle is represented, the other elements being obtained by symmetry. Broadcasts along row and column highways, or systolic data movements, require a true rectangular representation of the outer product. The only general saving that seems possible to exploit is to note, both in phase H and phase V, that there is no point in representing coefficients corresponding to nodes in the separator in

those equations at nodes not being eliminated. Such components are only used in determining what multiple of the equation being eliminated to subtract, and from the symmetry of the system this information can be obtained from the coefficient in the equation at the node in the separator corresponding to the node not being eliminated. In other words, in Figures 6 and 8, it may be feasible to dispense with the portion of those columns in the segment labeled 1 below the rows labeled 1. From Figures 6 and 8, it is readily seen that this saving is between 10% and 16%. It hardly seems worth the trouble to exploit.

Let us turn to the number of sequential steps taken to solve the system. Each stage s requires $2^{s-1}-1$ outer product updates for eliminating the nodes of the H separator, and $2^{s}-1$ outer product updates for eliminating the nodes of the V separator. The special stage s=1 requires 1 outer product update. Thus the number of sequential outer product update steps to the end of stage s is

Sequential outer product update steps to end of stage s

$$=1+\sum_{2}^{s}(2^{t-1}-1)+\sum_{2}^{s}(2^{t}-1)$$

$$=3\times 2^{s}-2s-3 \qquad (13)$$

Notice that the total number of sequential outer product update steps to reach the final generalized element is only 3 times the length of a side of the original grid, that is, 3/2 the number of steps it would take to propagate a signal

from one corner of the original grid to the diagonally opposite corner by nearest neighbour connections. Notice also that half the outer product update steps to the end of stage s are performed during stage s itself. Sherman's idea of recalculating the partial factorization instead of saving it means the total number of sequential outer product steps is equation (13) summed from s=1 to m.

Total sequential outer product update steps if partial factors recalculated

$$= \sum_{1}^{m} (3x2^{s} - 2s - 3)$$

$$= 6x2^{m} - m(m+4) - 6 \qquad (14)$$

That is, it doubles the computation time. The ratio changes when the $4x2^{m}$ outer product update steps required to solve the final generalized element are considered, but the result remains roughly true. Thus from equations (9) and (14), the tradeoff of whether it is better to save the partial factor on backing store or to recompute it comes down to whether transferring the original system to and from backing store (which is order 2^{2m}) is m times cheaper than all the arithmetic to solve the system (which is order 2^{m}), and it seems certain that for large enough problems, recomputation is better.

IV. FITTING THE ALGORITHM TO MACHINES

Although the algorithm we have constructed solves an important class of problems, and has enormous parallelism

available to exploit, it does not fit well with current parallel machines. There are three reasons for this. The first is that the basic parallel operation of this algorithm, the ability to do many disjoint outer product updates as a single operation, is not available. The second is that the data routing and processor reallocation between stages cannot be done easily. The third is that typical problems have more parallelism, or a larger active storage requirement, than current machines can provide. We will consider these in turn.

The outer product of two vectors is an operation fundamental to many linear algebra computations. Even when other formulations are possible, the outer product formulation is often superior. It is therefore surprising that none of the vector computers have provided an outer product operation (much less the ability to do many outer products simultaneously in a single operation), even though it would seem straightforward to implement and would help keep "vectors" long, i.e. pipelines full. The operation is readily provided on more flexible machines, such as the Floating Point Systems "add-on" array processors, but these rarely have enough memory to directly exploit the available parallelism. True parallel machines, with a large number of processing elements, can simultaneously do the multiplications and then simultaneously do all the additions if each combination of one element from either vector in the outer

product is routed to a different processing element. A two dimensional array of processing elements, such as the ICL Distributed Array Processor, can do this routing for a single outer product effectively as a single operation but the obvious way to do more than one outer product at a time by concatenating the sets of vectors is extremely wasteful of processing elements. More general data routing internal to the implementation of the outer product seems to be required. This is particularly true since the size of the outer products and the number of replications change over the course of our algorithm. It is not obvious that a parallel machine with any fixed set of data paths can cope with this variation, unless the data routing is treated as a multistep general permutation through the connection network.

The second reason why the algorithm does not fit well is a more obvious data routing problem. If the data structure and processor allocation at the beginning of a stage and phase is a set of replications of arrays as in Figures 6 and 8, it is not at all obvious how, at the end of the stage and phase, to rearrange processors and data so the next stage and phase can begin. The results of equations (6), (7), and (8) show that given, say, 25×2^{2m} processors (or storage cells) there will be enough at each stage. However, not only must processors or storage associated with equations or variables eliminated in the stage or phase be reallocated to

accept the fillin of the next stage or phase, but a rearrangement may be required as the arrays get bigger and there are fewer of them. This is in addition to any rearrangement required in how the blocks of the arrays at one phase fit into the blocks of the array at the next phase and, of course, to the rearrangement required for the reversal of order and summing to unify the multiple representation of the separator. It is also in addition to any rearrangement required to conform with physical layout required by the machine to perform the outer products of the next phase simultaneously, as discussed above. Mapping the arrays of Figures 6 and 8 onto processors which form a one or two dimensional address space, it appears that again the data routing will have to be treated as a multistep general permutation through the connection network.

The third reason why the algorithm does not fit current machines well seems obvious to remedy: if the machine does not provide enough parallelism for the algorithm, the computation could be done serially in chunks, where a chunk is as large as the available parallelism will permit. For many algorithms, this is inordinately expensive, due to difficulty patching the chunks together. For nested dissection this is not a problem, if chunks are chosen appropriately, as part of an outer product can still be an outer product, and replications can obviously be done serially. However severe performance loss can still occur because of bandwidth

limitations, moving data between processors and backing store. We have organized the algorithm by stages to maximize parallelism, but if the cost of transfers to and from backing store is sufficiently high, it may be desirable to reorganize the algorithm so once part of the data is in the processing elements, several stages are done with this part of the data before going back to do those stages again with the next part of the data. Particularly for the early stages of the algorithm, this will improve the ratio of computation to transfer cost.

Despite the aforementioned difficulties, nested dissection must be fitted to today's machines, especially serial or vector machines. Although other implementations are possible, one way to do this is by simulating the parallelism in the algorithm as described above.

For a serial machine, the outer products pose no problem and the rearrangements, since they do not need to be done in place nor with vector or array operations, amount simply to a pass over the data, moving each value to its new location. For a serial machine, there is no need to use the multiple representation technique to avoid the multiple simultaneous update problem, of course. However since it is so cheap, and makes the data structures more regular, it is worth keeping anyway. For a serial machine there is no reason not to exploit symmetry fully in the arrays of Figures 6 and 8, both in storage representation and in the arithmetic of

outer product updates. This roughly halves the storage requirements expressed by equations (6), (7) and (8) to give instead

H separator elimination phase symmetric storage

$$=2^{(2m-2s+1)} \times 2^{-1} \times 2^{s-1} \times (7\times 2^{s-1}-1) \times (7\times 2^{s-1}-1+1)$$

$$=12.25\times 2^{2m} - 3.5\times 2^{2m-s} \qquad (15)$$

V separator elimination phase symmetric storage

$$= 2^{(2m-2s)} \times 2^{-1} \times 2^{s-1} \times (10\times 2^{s-1}-1) \times (10\times 2^{s-1}-1+1)$$

$$= 12.5\times 2^{2m} - 2.5\times 2^{2m-s} \qquad (16)$$

Special stage s=1 symmetric storage

$$=2^{2m-2} \times 2^{-1} \times (9) \times (9+1)$$

$$=11.25\times 2^{2m} \qquad (17)$$

The total arithmetic (multiply and add operation) count to the end of stage m is readily calculated

Serial arithmetic count to end of stage m

$$=371/12\times 2^{3m} - 17\times m\times 2^{2m} - 145/4\times 2^{2m} + 16/3\times 2^{m} \qquad (18)$$

To this must be added the cost of solving the final generalized element corresponding to the perimeter. The complete cost (neglecting the backsolve and the possibility of Sherman's technique) is thus

Serial arithmetic count for complete factorization

$$=499/12\times 2^{3m} - 17\times m\times 2^{2m} - 145/4\times 2^{2m} + 14/3\times 2^{m} \qquad (19)$$

Comparison with the arithmetic operation count in (3) shows

this implementation is just over four times as expensive as the one given there. The principal difference is that the separators here go up to, but do not include, the perimeter as they do there, and consequently the final clique is larger and there are no special case boundary elements as described there. We will return to this point. Notwithstanding, if we compare this implemention with band methods, which is appropriate as both use dense matrix representations without pointers, the band methods take about $1/2x2^{4m}$ operations, so the crossover should occur before m=7, i.e. $2^{m}=128$.

For a vector machine, this implementation could again be used and all the major operations are vectorizable, provided no constraint in what the machine can do interferes. Even the rearrangements can be done by block moves, although the reversal and summing of the separator requires the ability to step backward through a vector. Again, as with serial machines, multiple representation is not essential but is cheap for the regularity it provides (keeping average vector lengths up), and symmetry can be fully exploited. The space requirement and per-result operation counts for vector machines are thus as for serial machines. The crucial question for vector machines, however, is the number of vector operation startups.

The basic operation of this implementation, multiple outer product updates, can be expressed serially as three

nested loops: across replications, and across the components of each of the vectors whose outer product is being formed. The order in which these loops are nested is immaterial, and for any order of loops the representation of the multiple arrays like Figure 6 and 8 can readily have the innermost stride unity; but current vector computers can only vectorize the innermost loop. Because most of the computation occurs in the later stages of the algorithm, it is readily shown that the loop across replications should be outermost, i.e. not vectorized. (This would not be true if we were willing to use different code and data structures for the early stages, where replications obviously provide the longest vectors.) The number of startups for arithmetic operations through to the end of stage m, assuming a single startup to subtract a scaled vector from another, is

Startups to end of stage m

$$= 31/4 \times m \times 2^{2m} - 11/3 \times 2^{2m} + 11/3 \qquad (20)$$

To this must be added the startups for arithmetic operations in solving the final generalized element corresponding to the perimeter. The complete cost is thus

Startups for complete factorization

$$= 31/4 \times m \times 2^{2m} + 13/3 \times 2^{2m} + 2 \times 2^{m} + 11/3 \qquad (21)$$

From equations (19) and (21), the average vector length for arithmetic operations, to leading order, is

$$\text{ave vector length} = 499/93 \times 2^{m}/m \qquad (22)$$

for $m=7$, i.e. $2^m=128$, this average length is 98, so on a 100-1 vector machine approximately as much time would be spent in startups as in per-result arithmetic.

V. CONCLUSIONS AND GENERALIZATIONS

We have demonstrated that nested dissection is an algorithm with considerable parallelism to be exploited, and though deficiencies in current parallel machines make it difficult to take advantage of the parallelism, an implementation designed to exploit the parallelism is still quite effective for serial or vector machines. However in deriving our implementation we have limited ourselves to square grids with 2^m+1 nodes on each side, and to separators which stop short of the perimeter. Neither of these are intrinsic to nested dissection, which works even with irregular regions. Instead, they are restrictions we imposed in order that all replications at each stage be identical, so SIMD parallelism could be used. If, as in the serial or vector machine realization of our implementation, no such parallelism across replications is being used, then special cases for boundary elements are easily introduced. Serial arithmetic, vector startups, and storage requirements are all improved by using a 2^m-1 by 2^m-1 grid of unknowns with the separators as before, except that now the separators go right onto the boundary. The special boundary elements required are discussed in (1), and, of course, exploiting them gives space requirement and operation count quoted there.

For a fully parallel implementation, however, it probably is most effective to define the initial equations so the desired problem is simply embedded within the grid that this implementation solves.

REFERENCES

1. George, J.A., "Nested Dissection of A Regular Finite Element Mesh", SINUM, Vol. 10, No. 2, pp. 345-363, 1973.

2. Lambiotte, J.J., The Solution of Linear Systems of Equations on a Vector Computer, Ph.D. dissertation, Department of Applied Mathematics and Computer Science, University of Virginia, 1975.

3. Calahan, D.A., "Complexity of vectorized solution of two-dimensional finite element grids", Technical Report No. 91, Systems Engineering Laboratory, The University of Michigan, 1975.

4. Sherman, A.H., On the Efficient Solution of Sparse Systems of Linear and Nonlinear Equations, Ph.D. thesis, Computer Science Department, Yale University, New Haven, Connecticut, 1975.

5. George, J.A.; Poole, W.G.; and Voigt, R.G., "Analysis of Dissection Algorithms for Vector Computers", Journal of Computers and Mathematics, Vol. 4, pp. 287-304, 1978.

6. George, J.A.; Poole, W.G.; and Voigt, R.G., "Incomplete Nested Dissection for Solving n by n Grid Problems, SINUM, Vol. 15, No. 4, pp. 662-673, 1978.

7. Lipton, R.J.; Rose, D.J.; and Tarjan, R.E., "Generalized

Nested Dissection", SINUM, Vol. 16, No. 2, pp. 346-358, 1979.

8. George, J.A. and Liu, J.W., Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.