solve ill-conditioned problems accurately without recourse to higher precision arithmetic.

Algorithms that compute a matrix inverse, whether iteratively or directly, are unsatisfactory for two reasons. First, they are a step backward from Gaussian elimination. Not only are they less efficient, but solutions obtained by inverting and multiplying may fail to satisfy the original equations. Second, the inverses of large sparse matrices are not usually sparse, and their computation increases storage costs dramatically. For iterative inversion methods the problem of obtaining a starting approximation is only an additional complication to an otherwise unpromising approach.

Given the drawbacks of iterative matrix inversion, it is not surprising that Schultz's method has languished (*2*). Nor has the problem of finding a starting approximation stumped researchers; it was solved 20 years ago by Ben-Israel (*3*). The fact that it made no splash suggests the status of the problem in the numerical community.

The parallel implementation of Schultz's algorithm proposed by Pan and Reif requires about n^3 processors. The article does not alert the reader to the consequences of this fact. Let us suppose, very generously, that we can place 100 processors with the capability of doing floating-point arithmetic on a board 8 inches square and that we can place the boards an inch apart, so that 100 processors occupy a volume of 1/27 cubic feet. Then the billion processors required to solve a system of order 1000 (a moderate system by today's standards) would occupy a 72-foot cube, the size of a-not-inconsiderable mansion. When the theoretical number $n^{2.5}$ is used in place of n^3 the cube shrinks to 23 feet, still a monstrously large computer. None of this makes any concession to practicalities like heat dissipation and communication costs, which will increase the size of the computer and decrease its efficiency.

Given the unsuitability of the parallel version of Schultz's method for solving problems of moderate size, how can it be said that "Pan and Reif made their method much more efficient for certain commonly occurring problems in which the matrices have lots of zeros. . . "? The answer is that Pan and Reif actually propose two algorithms: the iterative method described in the article and a direct method, based on a technique known as nested dissection, which is closely related to Gaussian elimination. The two algorithms are connected by the fact that the first is used to speed up the second. Nowhere in the article is this made clear.

Does the combined method promise to have a real impact on how linear equations are solved? I think not—for the following reasons. Typically the method of nested dissection might be applied to a system of linear equations associated with a two- or three-dimensional mesh. The order of the system is equal to the number of points in the mesh. The first step in the method is to find a set of points, called a separator, whose removal will divide the mesh into two pieces. The equations corresponding to these points are eliminated from the system and the process is applied recursively to the two submeshes. The elimination of the equations can be accomplished in various ways. Pan and Reif propose to do it by using their first algorithm to solve systems of the same order as the number of points in the separator. There is more to the algorithm than this (for example, the eliminations are done in the order opposite that described above), but the description will suffice for what follows.

Now consider a 32×32 mesh, which corresponds to a system of order about 1000. The separators have 32 points, and accordingly Pan and Reif will require about 32,000 processors to solve a system of order 32. This is overkill.

However, by today's standards a 32×32 mesh is a toy problem. A more realistic problem might come from a three-dimensional mesh of dimensions $10 \times 100 \times 100$. Here the separators are of order 1000, and we are back to the billion processors mentioned above. The difficulty is that n^3 is such a swiftly growing function of n that by the time we reach realistic problems the number of processors has become unrealistic.

It is important to keep things in perspective. By itself, the method of nested dissection is a prime candidate for parallel implementation, and in fact a number of people besides Pan and Reif are working along these lines. Moreover, by coupling nested dissection with Schultz's method, Pan and Reif have obtained some important theoretical results on the complexity of the solution of linear systems. But for the reasons outlined above, the combined algorithm is unlikely to have any impact on the way linear equations are solved.

G. W. STEWART
Department of Computer Science, Institute for Physical Science and Technology, University of Maryland, College Park, MD 20742

REFERENCES AND NOTES

1. G. Kolata, *Science* **228**, 1297 (1985).
2. For a complete analysis of the algorithm, its convergence, and its behavior in the presence of rounding error, see T. Sodorstrom and G. W. Stewart, *SIAM J. Numer. Anal.* **11**, 61 (1974).
3. A. Ben-Israel, *Math. Comput.* **20**, 439 (1966).

8 August 1985; accepted 20 March 1986

Response: Stewart makes three main points.

1) He criticizes what he feels is a general lack of accuracy in Kolata's science reporting.

2) He (correctly) states that there was a prior paper of Ben-Israel that had previously given a method for initializing the Newton iteration method for matrix inversion.

3) He argues (we believe incorrectly) that the parallel algorithms given in our paper for solving linear systems cannot be practical because of large processor bounds.

Our reply to point 1 is that, indeed, Kolata made some misleading statements in her article, particularly in not differentiating between the parallel Newton iteration algorithm for dense matrices (which we view as being only of theoretical interest for large linear systems) and our parallel nested dissection algorithm for sparse matrices (which we have found to be practical for large sparse linear systems).

Also, Kolata gave weather prediction as an example of the application of large linear systems, when, as Stewart correctly points out, weather prediction involves the solution of nonlinear systems. But these nonlinear systems can be linearized by stepping in time. Thus the solution of the resulting large linear systems is a significant component of weather prediction, but not the only one. However, the point Kolata was making was that the solution of large linear systems is an important problem. This is not a controversial statement—it is widely accepted by the scientific community. The solution of large linear systems is central in the engineering sciences (for example, in structural analysis and circuit analysis), in applied physics (for example, in the solution of partial differential equations), and in many optimization problems that occur in economics.

Stewart is correct about Ben-Israel's prior work, but the fault is clearly ours, not Kolata's. Although we were not aware of his work when we presented our paper in 1985, we have long since revised the paper to give Ben-Israel credit for this contribution.

Kolata's reporting, particularly in the areas of mathematics and computer science, has been credible. She has generally succeeded in the difficult task of describing complex technical ideas in simple terms that can be understood and appreciated by the general readership of *Science*. Technical errors, no matter how limited, detract from the general goals of science reporting. Unfortunately, Stewart's comment, which lists what he regards as Kolata's errors, actually contains a number of what we view as errors.

Many in the scientific computing community might disagree with Stewart's statement that iterative methods are not commonly used to improve the accuracy of the solu-

tions provided by Gaussian elimination. Most, if not all, linear system software packages provide routines for exactly such iterative improvement.

Finally, we address Stewart's point 3. We summarize our response as follows. We will show there is a fallacy in his argument that the parallel nested dissection algorithm is not practical. In particular, Stewart selectively ignores the use of the slowdown principle, which allows the algorithm to be used in practice. (In fact, our parallel nested dissection algorithm has been implemented on two massively parallel machines.)

Before giving a detailed response to Stewart's point 3, we give some background information. Our paper "Efficient parallel solution of linear systems" (*1*) was first presented at the 17th Annual Association for Computing Machinery Symposium on Theory of Computing in May 1985. The goal of our work had been to develop parallel algorithms to solve linear systems by using some newly available massively parallel machines, including the 16,384-processor Massive Parallel Processor (MPP) built by Goodyear, Inc. for the Goddard NASA Space Flight Center at Greenbelt, Maryland, and the 65,536-processor Connection Machine constructed by Thinking Machines, Inc., in Cambridge, Massachusetts. Although these machines had been built in 1985, it was not clear at the time which parallel algorithms could be used to exploit their massive parallelism.

How can we compare the performance of parallel algorithms? The critical resources used by a parallel algorithm are the number of processors, P, and the time, T. The ultimate goal of such massive parallelism is to minimize the time bound, T, as much as possible. A parallel algorithm is defined to be fully parallel if it has a time bound T that is polylog [that is, T grows as a constant times $(\log n)^c$, where c is a constant], which increases only very slowly with the size, n, of the problem.

The total work done by a parallel algorithm is defined to be PT, the product of the processor bound by the time bound. A parallel algorithm is defined to be efficient if PT is no more than a constant factor more than the best known time bound for a sequential algorithm to solve the problem. The goal of our paper was to develop parallel algorithms that were both efficient and fully parallel. We succeeded in both goals. For a dense linear system with n variables and n unknowns, the sequential number of steps to solve this problem is approximately n^3 by Gaussian elimination, which can be efficiently parallelized to a minimum of n time using n^2 processors. Gaussian elimination cannot be made fully parallel, that is, it cannot be made to run with polylog-time bounds. In contrast the Newton iteration method described in our paper is both efficiently and fully parallel, with a constant times $(\log n)^2$ time using less than $n^3/(\log n)^2$ processors (in fact, our processor bounds are a power of n somewhat less than 3).

It is important to define a slowdown principle well known to parallel algorithm designers. Suppose that an efficient parallel algorithm exists with time bound T and processor bound P. Suppose also that we wish to implement the parallel algorithm on a parallel machine with smaller processor bound $P' < P$. Then we let each of the P' processors of the machine sequentially simulate a group of P/P' processors of the algorithm, increasing the time bound by a factor of P/P' to $T' = T\ (P/P')$. The resulting slowed-down algorithm is still efficient, since $P'T' \leq 2\ PT$. Stewart uses this principle to slow down parallel Gaussian elimination to approximately time n^2, using n processors. In his comparison of the Newton iteration and the Gaussian elimination methods, Stewart does not mention that the Newton iteration method can also be arbitrarily slowed down, resulting in an efficient parallel algorithm with a time bound to approximately n^3/P for any given processor bound $P \leq n^3/(\log n)^2$. In particular, the slowed-down Newton method also takes time approximately n^2, using n processors. Stewart applies the slowdown principle selectively only to the Gaussian elimination method; that method actually is more restrictive and is known to be less numerically stable than the Newton iteration method. Stewart correctly states that the Newton iteration method uses work PT, which is approximately n^3, but does not mention that the widely used Gaussian elimination method has precisely the same asymptotic bound, $PT = n^3$. Both methods are practical for n ranging up to a few hundred, but grow in cost rapidly as n grows above 1000.

At any rate, neither of these methods are of practical interest for the solution of large linear systems where $n \geq 1000$ variables and equations, since generally in these cases the associated matrices are sparse, that is, have mostly zeros. In many applications these sparse matrices have a special structure of zeros and nonzeros that can be parameterized by a variable, S, known as the separator bound. In many cases such as in the solution of two-dimensional partial differential equations, S is approximately $n^{0.5}$.

The parallel nested dissection algorithm described in our paper had a time bound of a constant times $(\log n)^3$ and a processor bound of less than $S^3/(\log n)^3$ (in fact, our processor bounds are a power of S somewhat less than 3). If one assumes that S is approximately $n^{.5}$ the total work of our parallel algorithm is $PT = n^{1.5}$, matching the bounds on work done by a previously known sequential method given by Lipton, Rose, and Tarjan in 1979 (*2*). This work bound ($n^{1.5}$) is much less than n^3 for even a moderate n and so is a substantial improvement over both the Newton iteration method and the Gaussian elimination method, which have a work bound of n^3. In his argument that parallel nested dissection is not practical because of large processor bound, Stewart again does not mention that the parallel nested dissection can be slowed down, to a constant times $n^{0.5}$ time, if we use only n processors. In comparison, the parallel Gaussian elimination method takes n^2 time using n processors, which is orders of magnitude more time-consuming. This is the crucial fallacy in Stewart's argument that the parallel nested dissection algorithm cannot be practical because of too large a processor bound.

Also Stewart does not mention (although it is prominently mentioned in our paper) that after computing a sparse factorization of a sparse input matrix, our parallel nested dissection then requires only a constant times $(\log n)^2$ and n processors to compute any further solutions. Thus the algorithm runs fully parallel (without slowdown), in frequently occurring cases where the coefficients of a linear system stay invariant (for example, in solution of a fixed stress analysis problem given many distinct loadings or in a fixed partial differential equation with varying boundary conditions).

The practicality of our parallel nested dissection algorithm was demonstrated by its implementations (*3*) on two massive parallel machines—the previously mentioned MPP and the Connection Machine.

VICTOR PAN
State University of New York,
Albany, NY 12203
JOHN H. REIF
Department of Computer Science,
Duke University,
Durham, NC 27706

REFERENCES

1. V. Pan and J. H. Reif, *Proceedings of 17th Annual ACM Symposium on Theory of Computing* (Association for Computing Machinery, New York, 1985), pp. 143–152.
2. R. Lipton, D. Rose, R. E. Tarjan, *SIAM J. Numer. Anal.* **16**, 346 (1979).
3. C. E. Leiserson *et al.*, in *Annual SIAM Conference* (Society for Industrial and Applied Mathematics, Boston, MA, July 1986), p. A51; T. Opsahl and J. H. Reif, in *First Symposium on Frontiers of Scientific Computing* (NASA Goddard Space Flight Center, Greenbelt, MD, September 1986), pp. 241–248.

11 February 1987; accepted 3 March 1987