Large Dense Numerical Linear Algebra in 1993: The Parallel Computing Influence

Alan Edelman
Department of Mathematics
and Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

December 3, 1994

Abstract

This article surveys the current state of applications of large dense numerical linear algebra and the influence of parallel computing. Furthermore, it attempts to crystallize many important ideas that are sometimes misunderstood in the rush to write fast programs.

1 Introduction

This article represents my continuing efforts to track the status of large dense linear algebra problems. The goal is to shatter the barriers that separate the various interested communities while commenting on the influence of parallel computing. A secondary goal is to crystallize the most important ideas that have all too often been obscured by the details of machines and algorithms.

Parallel supercomputing is in the spotlight. In the race toward the proliferation of papers on person X's experiences with machine Y (and why X's algorithm runs faster than person Z's), sometimes we have lost sight of the applications for which these algorithms are meant to be useful. This article concentrates on large dense numerical linear algebra. I do not tell you how to write an LU decomposition – there are more than enough people doing this. (By all means consult [14, 19, 12, 32].) What I propose is to give a bird's eye perspective on dense numerical linear algebra and how it is currently used on very large systems.

Some caveats are in order. The most obvious is that what is true in 1993 may or may not be true five years from now. Five years ago, only a few of the world's leading numerical linear algebra experts were actively working on distributed-memory parallel supercomputers. (In fact, five years ago few of the world's experts in any field were actively working on distributed-memory supercomputers.) At that time, some linear algebra experts felt that highly parallel machines (such as the Connection Machine) would never be very useful for large dense linear algebra computations. Perceptions changed when Intel and Thinking Machines began a heated race for the top of the Linpack benchmarks. The early mistake serves as an instructive lesson in the sociology of science: experts may not always fully understand the potential of a new technology. This is the past. The pendulum has swung the other way now, and we begin to ask ourselves whether we are expecting too much too soon from this technology. As Cybenko and Kuck [10] point out,

> Scientists and engineers now expect teraflops machines by 1996 – and in a climate of heightened expectations, four years is a long time. ...[If experience with newer machines resembles that of earlier machines] the field of massively parallel computing risks taking a spot on the technical community's back burner, much as artificial intelligence did after overextending itself in the 1980's.

I will discuss all the applications of large dense linear equation solving and eigenvalue computation that I have been able to obtain. It would be folly to conclude that this is exactly the set of applications that will be important in the future. When I discuss large problems, I am only considering a small segment of the user community with access to fast machines. Therefore, I am covering only those applications areas in which some proper mix of necessity, money, or computing culture has allowed for entrance into the world of dense numerical linear algebra.

My data consist of the results of my second annual dense numerical linear algebra survey posed in the electronic newsletter *NA Digest* and a variety of electronic bulletin boards. I do not claim to understand all of the applications, but I hope that my general descriptions are accurate. Indeed, I rely on my colleagues and friends to correct inaccuracies when I am on shaky ground. Of course, I have not reached everybody. My continuing hope is to gather more information in years to come.

2 Approaches towards large problem solving

I would like to examine three points of view that illustrate very different approaches toward the solving of large problems. Each approach serves important, though different purposes. It would be a mistake to construe that any of these activities are somehow a waste of time, but I suspect my biases are clear.

The first approach is the most common, I believe, because it is has been the most exciting and requires a minimum of training. Mainly what is needed is determination and hard labor.

> 1. "How many machoflops can we get?"
> Here the style is to try to get the highest possible performance out of some machine, usually on a matrix multiply problem or on a linear solver.

The race for machoflops has been run by both supercomputing manufacturers and computer science researchers. Those with the best access to the true machine characteristics have often had the advantage. As an exercise for students, getting machoflops may be the best way to really get to know an architecture.

Another tradition represented by LAPACK and its predecessors is the following:

> 2. "PROVIDE A BLACK BOX ROUTINE FOR THE MASSES"
> Here performance is important, but more important is to make sure that state-of-the-art *numerical* techniques are used to insure the best possible accuracy is obtained by users who want to trust the software. Since it is for the masses, the code must be portable.

LAPACK software is like an automobile built for public use that has been designed to meet safety regulations. Such a car should drive efficiently, but certainly would need to be rebuilt for more dangerous racing purposes. LAPACK is written in Fortran 77 plus calls to the basic linear algebra subroutines (BLAS) to perform kernels such as matrix-matrix multiplication in a manner that may be optimized for each machine. If high performance Fortran lives up to its promises, one can imagine successors to LAPACK being written in it. At the moment, it seems unlikely that any such software will be able to support the full functionality of, say, the Connection Machine Scientific Software Library.

There is another viewpoint regarding large dense linear algebra problems popular among specialists, but probably not so widely considered among other circles:

> 3. THE "ALL LARGE DENSE MATRICES ARE STRUCTURED" HYPOTHESIS.
> This point of view states that nature is not so perverse as to throw n^2 numbers at us haphazardly. Therefore when faced with large n by n matrices, we ought to try our hardest to take advantage of the structure that is surely there.

In terms of numerical linear algebra algorithms, what is needed are specialized approaches for individual problems that use more information about the matrix other than the fact that it is an n by n array of numbers. This idea is familiar when the dense matrix has obvious structure such as Vandermonde matrices, Toeplitz matrices, and more simply, orthogonal matrices.

At first, using "sparse matrix" techniques for dense matrices may seem counterintuitive. Nevertheless, we outline a basic recipe that is highly appropriate when considering large problems:

"SPARSE" APPROACHES TO DENSE PROBLEMS
1. Replace a traditional "dense" approach with a method that accesses the matrix only through matrix vector multiply.
2. Look for good preconditioners.
3. Replace the $O(n^2)$ vanilla matrix-vector multiply with an approximation such as multipole or multigrid-style methods or any other cheaper method.

This recipe forces the software developer to access the structure in the matrix, both in the search for a good preconditioner and in the matrix-vector multiply. For sparse matrices this has always been obvious; great efforts have gone into the study of preconditioners for sparse matrices and the computational organization of matrix-vector multiplies. Efforts in the dense case are rarer, I think, mainly because of the division of labor (and understanding) between computing scientists (i.e., physicists, chemists, and engineers) and computer scientists. See [20] for one success story.

Perhaps some tradeoffs between traditional dense approaches and sparse approaches are apparent. Everything about using LU is predictable, from its error analysis to its time of execution. Nearly everything one needs to know can be found in undergraduate textbooks. In contrast, finding the right "sparse" approach can be a tricky business requiring time and expertise. There are many iterative methods to choose from, many of which have not yet found their way into undergraduate (or even graduate) texts [18]. If one stumbles on an approach that seems to work well for one problem, there is little guarantee that it will work for another problem. Also, replacing a matrix-vector multiply with a fancier approach requires a fairly serious software effort, and finding good preconditioners may require considerable expertise. There are many problems for which good preconditioners have yet to be discovered.

Nevertheless, trying a few basic iterative methods is not so difficult. A modest effort is required to merely play with a few ideas, and under the right circumstances the payoff can be enormous.

The idea that general dense methods might be less appropriate than application-specific ideas was eloquently expressed fourteen years ago in a prediction by Parlett

> Rather than solving more and more *general* problems, the development of the field [of numerical analysis] will lie in more and more specialized applications ... chemists will not be well advised to borrow the "best" method for ... general ... problems, but will be well advised to consult with the experts on how to build their own codes. ([28])

There is a subtler reason to shun dense approaches to large dense linear algebra problems, which will be discussed below.

3 Records

Records

Largest Gaussian elimination (LU):	75,264	(complex unsymmetric)
Largest symmetric eigenproblem:	27,000	(all eigenvalues and all eigenvectors)
Largest unsymmetric eigenproblem:	10,000	

The record holder for the dense (unstructured) linear solve problem is $n = 75,264$ performed by Intel Corporation on a 128 processor iPSC/860. The problem ran for 64 hours in double precision. (Perhaps another interesting question might be what is the longest any dense linear problem has ever run?) The real and imaginary parts of the elements of the matrix were generated independently and uniformly on $[-1, 1]$, and then 1,000 was added to the diagonal so that the matrix was virtually diagonally dominant. In some ways this matrix resembles the matrices that arise in radar cross-section calculations. The matrix factorization routine performed partial pivoting. The solution x to $Ax = b$ was found where b is the vector of all ones. The residual vector was found to have elements of order 10^{-13} or smaller.

A back-of-the-envelope random matrix calculation (see [15]), convinces me that such a matrix most certainly has a two norm condition number smaller than 3, and probably it is quite close to 1. In any event, this is a highly well-conditioned problem. If stability and conditioning are not concerns, then the only concern is the possibility of round-off. Notice that every matrix element is touched at most $3n$ times. A simple random-walk model predicts an error of order $\epsilon\sqrt{3n}$, taking $n = 75,264$ and $\epsilon = 2^{-52}$ predicts 10^{-13}.

4 What is LARGE?

As is well known by anyone who discusses "large" sums of money, the word is highly subjective. My approach is to seek out the record holders for linear algebra problems. I am using $n = 10,000$ as an arbitrary cutoff. Therefore, what I consider "large" would be out of the range of most scientists accustomed to desktop workstations. Perhaps another point of view is that "large" is bigger than the range of LAPACK. LAPACK, one goal of which has been to target a wide audience of users on workstations to vector architectures, could not support those who are trying to push machines to their limits. (Currently, the largest problems seem to be out-of-core solvers on distributed-memory machines, while the released version of LAPACK includes in-core routines for shared-memory or moderately parallel machines.) Yet others may define large by what can be handled by Matlab on the most popular workstations. We fear that perhaps some consider what Mathematica can handle in a reasonable amount of time as the limit.

It is curious to consider the differences between the largest linear system that has been solved as compared with the symmetric or unsymmetric eigenvalue problem. The difference, I think, says more about the state of the art of algorithmic development than the perceived need of these routines, though both play a role. LU is simply easier than eigenvalue calculations.

5 The Role of Ill-Conditioning and Condition Estimators

The condition number of a problem is often thought of as a measure of the sensitivity of the underlying problem to perturbations. Consider, however, a condition number paradox:

> A CONDITION NUMBER PARADOX
> Discretizations of Laplace's equation or the biharmonic equation have condition numbers that grow as large as you like with the number of grid points. However, a boundary integral equation approach (see below) to the same problem often leads to matrices whose condition numbers are asymptotically finite. If both approaches are solving the same problem, how can one approach be unboundedly sensitive, while the other approach is not?

For those less familiar with boundary integral equations, consider more simply the question of preconditioning. If a problem is fundamentally sensitive to perturbations, how can it be preconditioned? Is it not true that the rounding errors in the preconditioning perturb the solution greatly?

The resolution of the paradox is really quite simple, but I will delay an explanation by some paragraphs so that the reader may think about the problem. The reader may also wish to consider whether one can precondition an ill-conditioned dense matrix that is stored as an n-by-n array of elements.

To understand the resolution of this paradox, it is necessary to remember that the condition number of a matrix measures the worst case sensitivity of x in the equation $Ax = b$ to perturbations in A or in b, or in both. Traditional condition numbers $\kappa(A) = \|A\|\|A^{-1}\|$ measure the worst-case perturbation of x when $Ax = b$ is replaced by $(A + E)x = b + f$. Usually the spectral norm ("two norm") or the sup norm ("∞ norm") are used.

Is the condition number realistic? It is not unusual that the condition number is pessimistic for perturbations to b ([36, p.190]) but realistic for general perturbations to A. That is part of the paradox described above. Even though the matrix A is ill-conditioned, the right hand side is not sensitive to perturbations in b.

The traditional error analysis of Gaussian elimination gives the false impression that Gaussian elimination is the best one can do. The backward error analysis argument goes as follows: Gaussian elimination solves a nearby problem that is not much worse than the backward error in storing the matrix in finite precision. This is correct, but who ever said that each of the n^2 elements must be stored in finite precision? Let me express this as a corollary to the "all large dense matrices are structured" hypothesis:

Corollary: If all large dense matrices are structured, then representing the matrix with fewer than n^2 parameters may not only provide us with faster algorithms, but they may also be more accurate since the space of perturbations attributable to roundoff is much smaller.

If a matrix is well-conditioned, it hardly matters what computation method is chosen from the point of view of accuracy considerations. If a matrix is ill-conditioned, however, it does not imply that one does not deserve an accurate solution. In practice there are many so-called ill-conditioned problems whose solutions are well determined by the right-hand side. In such a case, specialized algorithms can compute residuals that are small compared with b, giving highly accurate solutions. Ill-conditioned matrix problems are not the same as problems that are not well determined by their data once one includes the structure.

The resolution of the paradox is that the condition number of the Laplace equation matrix has too many parameters allowing unnecessary perturbations in "non-physical" directions.

Condition estimators are heuristic algorithms that attempt to approximate the condition number of a matrix at a small cost in terms of execution time. In 1991, I learned that many people solving large problems using Gaussian elimination were not using condition estimators [17]. This seems to be changing. R. Craig Baucke at GE Aircraft Engines writes that all codes that he develops have condition estimators built in, and he encourages all users to exercise this option. The largest linear systems right now are generated from moment methods. These matrices are quite well-conditioned.

6 How to View a Matrix for Parallel Computation

Let us assume that the decision has been made to work directly with the elements of a matrix. Should this matrix be thought of as a two-dimensional array of numbers? From one point of view, it is natural to express a linear operator in a basis as a two-dimensional array because an operator is the link between two spaces – the domain and the range. The matrix element a_{ij} represents the component of the image of the ith domain basis vector in the direction of the jth range basis vector. Another reason that a matrix is naturally expressed as a two-dimensional array, I believe, is that we are three-dimensional creatures in the habit of communicating on two-dimensional surfaces, most commonly paper.

The following idea is trivial, yet I feel compelled to call it the most important concept in parallel matrix computations.

The lesson of parallel computing: Matrix indices need not be linearly ordered. Indices are better thought of as a set (hence unordered) rather than a sequence.

The nicest illustration of the idea that matrix indices need not be linearly ordered can be found in the Lichtenstein and Johnsson [26] implementation of LU on the CM-2. Probably the most common assumption on distributed memory machines is that the data layout is block consecutive. The matrix is divided into neat rectangles just as land in the "Old West" was parceled out. Traditional algorithms require that elimination begin with column 1 and proceed in consecutive order, leading to a hopelessly unbalanced workload. By a (psychological) relabeling of indices, the problem goes away. This is referred to fancifully as interchanging the space and time dimensions of the algorithm in [26], but the idea is much simpler: do not think of the matrix as linearly ordered; think of the indices as a set, not a sequence.

Another application of this lesson is the high-bandwidth hypercube matrix multiply discovered by Johnson and Ho and later rediscovered independently by myself after working on direct N-body solvers (Ho, Johnsson and Edelman [24]).

I digress to mention that the hypercube with fully progrmmable concurrent communication to every node's nearest neighbors is the only mathematically elegant ouse of the hypercube topology. Roughly speaking, every time you snap your fingers you may send words that you choose simultaneously to all of your neighbors. The CM-2 had this feature, though it was not widely publicized. I consider it a minor tragedy to mankind that the architecture was rarely properly appreciated [16]. Many researchers purporting to use the hypercube architecture really did not. The hypercube architecture, at least for now, has been set aside by most hardware designers. The ability to self-direct the communications in a deterministic manner has also been set aside by hardware designers. Despite asking for three years now, I have yet to hear a convincing argument of why the fearless individuals who wish to program at the assembly level are denied full access to the communications system as a matter of hardware policy, even though communication is a major bottleneck on modern distributed-memory machines.

The full-bandwidth algorithm rejects the notion that any block local to a processor would be unhappy if it were ripped to shreds and dealt in pieces to nearest neighbors. At the same time, the processor's neighbors send pieces of blocks to be glued together locally. A local, highly vectorized matrix multiply runs at high speed, and the operation continues in systolic fashion. Further details may be found in [24] or [12].

A popular misconception is that the following is the most important lesson of parallel computing on dense matrices:

A NON-LESSON OF PARALLEL COMPUTING: Operations must be blocked and data reused so as to maximize bandwidth to memory.

I refer to this as a nonlesson of parallel computing because it is not new, it is already the lesson of vector (and earlier forms of) computing. Some of these ideas date back to the earliest out-of-core solvers. The LAPACK BLAS neatly incorporate this lesson into high-quality software. If computer science really is a science rather than a sequence of case histories and experiments, we should not think of this observation as new and special for parallel computing, but merely refer readers to the previous literature.

The critical defect of the early CM-2 compilers and the Paris "assembly language" was the inability to perform multiple arithmetic operations on data for each memory reference. This defect was mistakenly considered a problem with the machine architecture when indeed it was a problem with the software model and underlying system software that was later corrected in newer CM-Fortran compilers.

7 Applications of Large Dense Linear Systems Solvers

In 1991 I pointed out that all large dense linear systems ($n > 10,000$) arose from the solution of *boundary integral equations* [17]. This continues to be the major source of large dense linear systems, but another important case arises in quantum scattering.

7.1 Boundary Integral Equations

Boundary integral equations are mostly coming from two distinct communities. Nevertheless, the techniques these communities are using come from the same basic framework.

There are a variety of ways to enter the enormous literature on boundary integral equations. There are the excellent recent surveys by Atkinson [2] and Sloan [34]. I have used Chapter 10 of ([25]) in a graduate course for the numerical solutions of PDE. A simple exposition with example can be found on pages 321–327 of ([1]). Other examples can be found in Bendali, 1984; Brebbia, 1984; Canning, 1990a, 1990b 1993; Harrington, 1990; Hess, 1990; and Wang, 1991.

The chart below indicates the major two communities involved with large boundary integral equation applications

Community	Electromagnetics	Fluid Mechanics
Nomenclature	Moment methods	Panel methods
Equation solved	Helmholtz	Laplace
	$\nabla^2 u_i + k^2 u_i = 0$	$\nabla^2 u = 0$
First kind	Charge distribution ρ	Single layer potential σ
Second kind	Current distribution j	Double layer potential μ

Mathematically, a differential equation in three-dimensional space is replaced with one of a variety of equivalent equations on a two-dimensional boundary surface. One can proceed to recover the solution in space by an appropriate integration of the solution on the surface. The price of going from three dimensions to two is the creation of a dense linear system that needs to be solved.

The electromagnetics community is by far the leader for large dense linear systems solving. Their goal is to solve the so-called radar cross section problem: a signal of fixed frequency bounces off an object, and it is of interest to know the intensity of the reflected signal in all possible directions.

They call their methods "moment methods" because of the inner products characteristic of Galerkin techniques.

Fluid mechanics, unlike electromagnetics, uses boundary integral methods only as a simplifying approximation for the equations they wish to solve. The goal is to understand the flow of air (for example) past an object using an idealized model of the flow. They call their methods "panel methods" because the most vanilla approach is to discretize the boundary into quadrilaterals.

In electromagnetics, we are interested in constant frequency solutions to the wave equation

$$\mu\epsilon \frac{\partial^2}{\partial t^2}\Phi = \nabla^2\Phi \, ,$$

where μ and ϵ are material constants of the medium in which the wave travels (the dielectric constant and the permeability constant). If we assume solutions Φ of the form $\phi e^{i\omega t}$, we obtain Helmholtz's equation.

The electromagnetics community often produces complex matrices that are symmetric because they arise from a Galerkin method in which the so-called basis functions and testing functions are complex conjugates. However, as was properly pointed out to me by R. Craig Baucke of GE Aircraft Engines, the matrices need not be symmetric, as arbitrary functions can be used.

In fluid mechanics, one ideally wishes to solve the Navier-Stokes equations of fluid flow. However, in certain situations one can greatly simplify the equations by assuming the flow of an incompressible, inviscid, irrotational fluid. In this case, the velocity at any point in space is obtained as the gradient of a potential: $v = \nabla\Phi$, where Φ is a solution of Laplace's equation.

Numerical linear algebraists sometimes like to ask whether a solution to a linear system is computed to high relative accuracy. However, this may be of less interest to some engineers:

> Prospective users of a flow–calculation method are rarely interested in whether or not an accurate solution of an idealized problem can be obtained, but are concerned with how well the calculated flow agrees with the real flow. ([23])

Some version of Green's identity is used to recast the problem as a boundary integral equation. In electromagnetics, one solves for either a "charge distribution" or a "current distribution" (or some mix) on the boundary which can then be used to obtain the desired solution at any location in space. Analogously, in fluids one solves for either a "single-layer potential" or a "double-layer potential" (or some mix) on a boundary.

Mathematically, one has an integral equation of either the "first kind" or of the "second kind." Characteristic of integrals of the first kind is the appearance of a Green's function in the integral, while in the second kind one notices normal derivatives of the Green's function.

7.2 Quantum Scattering

In quantum mechanical scattering, the goal is to predict the probability that a projectile will scatter off a target with some specific momentum and energy. Typical examples are electron/proton scattering, scattering of elementary particles from atomic nuclei, atom/atom scattering, and the scattering of electrons, atoms, and molecules from crystal surfaces. Experimentally, projectiles can be collided with each other (as, for example, in the proposed Superconducting Super Collider), or a projectile can be scattered from a fixed target (as, for example, in atom/crystal scattering, where an atom is scattered from a crystal surface). Both classical and quantum scattering of atomic scale particles are inherently probabilistic and the experimentally measured quantity is the scattering cross section. The differential scattering cross section gives the probability that a particle with an initially well-defined momentum will scatter into a particular direction with a particular momentum. The scattering cross section can be defined using classical physics, but for energy regimes in which quantum mechanics is applicable, scattering theory based on classical physics leads to physically incorrect results. In such cases, quantum mechanics is required to correctly determine the scattering cross section.

In quantum scattering theory, the projectile and target are assumed to initially be so far apart that they feel no mutual interaction. This corresponds to a time infinitely long before the collision. This initial state of the projectile/target system is given by its Schrödinger wavefunction. Long after the collision has occurred and the projectile has moved infinitely far away from the target and is again no longer interacting with it, the state of the system will be given by its final Schrödinger wave function. This corresponds to a time infinitely long after the collision. The scattering cross-section is obtained from the final Schrödinger wave function or from the scattering operator S, which maps the initial wave function into the final wave function.

The final Schrödinger wave function can be calculated by solving either the time-dependent or the time-independent Schrödinger equations. The time-dependent Schrödinger equation is a partial differential equation that is first order in time and second order in position. The time-independent Schrödinger equation is a partial differential equation that is second order in position. However, modern scattering theories usually determine the cross section by calculating the scattering operator S, instead of the Schrödinger wave function. The S operator is related to the transition operator T, which is given by the so-called Lippmann-Schwinger equation. This equation is a Cauchy singular Fredholm integral equation of the second kind. Numerical solution of either the time-dependent or time-independent Schrödinger equations, or of the Lippmann-Schwinger equation, nearly always requires solving large linear systems.

John Prentice [31] of Quetzal Computational Associates and Sandia National Laboratories has developed a theory for calculating the transition operator T (and hence the S operator and the scattering cross section) for inert atoms scattering from crystal surfaces at low energy. Assuming certain crystal lattice symmetry, the Lippmann-Schwinger equation in this case leads to a system of one-dimensional Cauchy singular Fredholm equations of the second kind. Prentice uses a Nystrom method to numerically solve this system and has so far discretized the equation into a dense linear system with $n = 6500$ unknowns on a Cray-2, though he would ideally like to solve the system for $n = 10,000$. He can foresee the need to perform $n = 100,000$ as well. Initial attempts to solve his system on the CM-2 have not yet been successful. The author considered some iterative methods for his system, but has not yet reported any success in this area.

Another very complicated example of quantum scattering, described to me by Don Truhlar from the University of Minnesota [33], includes chemical reactions that are assumed to occur when the projectile hits the target. In the language of quantum mechanics, these are known as "multichannel collisions." Here each "channel" refers to a possible set of quantum numbers for the reactants or products. Donald Truhlar reports solving dense linear systems of size $n = 11,678$ as one component of the calculation. The size of the matrix is the number of channels times the average number of basis functions per channel. The authors describe some preliminary results using GMRES.

7.3 Economic Models

Ted Frech, at University of California–Santa Barbara, informed me that large matrices have been used in input-output models of a central planning economy such as the former Soviet Union. The input-output model that has found many applications also for market economies is the Nobel Prize-winning invention of Leontief of New York University. The size of the matrix is related to the number of goods that are modeled. Frech surmises that the constraint for modeling many goods is the accumulation of quality, timely data rather than computing power. Neither he nor other experts in economics that I have contacted are aware of any current applications requiring dense matrices of order greater than 10,000.

7.4 Large Least-Squares Problems

Linda Kaufman of Bell Lababoratories reported a 2.7 million by 1300 dense least squares problem. She did not mention the application.

Srinivas Bettadpur described a large least squares problem in which the data were variations in the Earth's gravitational field. Techniques known as differential accelerometry can measure the gradient of the constant g to within 10^{-9} m/sec^2 for each meter. These measurements are then fit using least-squares to an expansion in spherical harmonics. In their problem they used the normal equations to build a 11,000-by-11,000 matrix, which then solved by the Cholesky factorization. They very carefully estimated the condition number of their matrix and found it to be acceptable [4].

8 Jacobi or Not Jacobi for Large Dense Symmetric Eigenvalue Problems?

In the early days of computing, the Jacobi algorithm was the method of choice for solving the symmetric eigenvalue problem. During the 1960's, the advantages of the QR algorithm became increasingly clear to the point that the Jacobi algorithm was discarded.

During the 1980's, the Jacobi eigenvalue method for symmetric matrices was resurrected because of its perceived utility on parallel machines. Many authors considered variations on the the "Kirkman Schoolgirl Problem" (from classical combinatorics) of n schoolgirls paring off on each of $n-1$ days so that everyone gets to be in a pair with everyone else exactly once. Such orderings have become parallel Jacobi orderings. One variation that I entitled the Paradiso Cafe Problem (named for a Harvard Square cafe that has four triangular tables outside) remains unsolved. Its solution seems to have little current relevance for parallel computing, but the problem has great sentimental value:

> THE PARADISO CAFE PROBLEM: If twelve computer scientists drink espressos at four different triangular tables, each night for six nights, is it possible to create a seating arrangement so that each scientist is sitting at a table with each other scientist at least once? What if the tables are arranged in a square and two of the three people at each table are found the next night at the two neighboring tables? (This corresponds to adding wires on an n-dimensional hypercube.) Generalize to $(n+1)2^n$ scientists at 2^n tables arranged in a hypercube.

The Jacobi algorithm received further attention in 1990 when Demmel and Veselić [13] announced that if a symmetric positive definite matrix happens to well determine its small eigenvalues, the Jacobi algorithm with a minor modification will compute these eigenvalues accurately, while the QR algorithm can destroy the eigenvalues. A matrix well determines its eigenvalues if small relative perturbations to the entries lead to small relative perturbations in the eigenvalues.

The Jacobi algorithm is once again falling out of favor in the 1990s, probably because the very fine grained model of computing that made Jacobi seem so attractive, turned out to be more abstraction than reality.

Though the Jacobi algorithm may still have uses in certain special cases, its chief attraction both on serial and parallel machines has always been its simplicity more than anything else. This idea has been espoused by Parlett ([30, p.184]).

9 Applications of Large Dense Eigenvalue Problems

It is clear that the electromagnetics community (rightly or wrongly) has been pushing the state of the art in conventional dense linear systems solvers. The large dense eigenvalue problem does not seem to have quite so strong a champion. The principle customers for large dense eigenvalue problems are computational chemists solving some form of Schrodinger's equation.

Rik Littlefield of the Pacific Northwest Lab sums up the situation as follows (personal communication):

> Eigenproblems in computational chemistry come in a wide variety of sizes and types. Generally the matrices are real symmetric, occasionally complex Hermitian. Roughly speaking, there are three main regimes:
>
> 1. Small (N = a few hundred) and dense, with complete solution required (all eigenvalues and vectors).
> 2. Moderate (N = a few hundred to a few thousand) and dense, requiring partial solution (20%-50% of the eigenvectors, corresponding to the smallest eigenvalues).
> 3. Large to very large ($N = 10^4$ to 10^9) but sparse (1% dense), and requiring only a few (one to ten) of the eigenvectors, corresponding to the smallest eigenvalues.

9.1 Schrödinger's Equation via Configuration Interactions

Probably the most studied eigenvalue problem in the computational quantum chemistry is the approximate solution of Schrödinger's wave equation,

$$H\psi = E\psi,$$

for a molecular system with many electrons. Here H is a linear differential operator known as the Hamiltonian. In this context it has four terms (not shown here): a Laplacian term, representing the electron kinetic energy, and three terms representing potential energy: electron-nucleus interaction, electron-electron repulsion, and nucleus-nucleus repulsion. The nuclei are considered fixed; hence the last term is constant.

The Schrödinger equation for a one-electron system (hydrogen) has a well-known analytic solution that is widely discussed in undergraduate courses in physical chemistry, general chemistry, general physics, or quantum mechanics. When feasible the electron problem is discretized and solved numerically as an eigenvalue problem. Alternatively, large dense random matrix models described in Mehta ([27, p.3]) have been used to model such systems.

Discretizing the problem requires a choice of a finite dimensional basis, but what constitutes a good basis? Bases are typically built up from sums and products of eigenfunctions for the hydrogen atom, with symmetries imposed from physical motivation.

Perhaps in the distant past, EISPACK style algorithms were used for such problems, but now the most common method of solution is Davidson's method ([11, 9, 28]) requiring accessing the matrix only through matrix-vector multiplies. Davidson's algorithm can be thought of as a hybrid algorithm between Lanczos and inverse iteration. The algorithm is similar to Lanczos in that it computes an orthogonal basis for a subspace one column at a time. It differs from Lanczos in that one does not compute a subspace on which the original operator is tridiagonal. The trade-off is that tridiagonality is sacrificed in the hope of speedier convergence. In Lanczos, the next vector to be computed has to be an orthogonal basis vector for the next bigger Krylov space. In Davidson, one performs an approximate inverse iteration to get a good eigenvector and insists that it be in the span of one's orthogonal space at the next step.

This method, though familiar for many years to researchers in computational chemistry, remains somewhat unknown to the numerical linear algebra community. For this reason, we discuss it briefly.

Davidson's Method for Eigenvalue Computation

Choose an initial $V_1 = [v_1]$
for k=1,...
  $H_k := V_k^T A V_k$
  Compute the smallest eigenpair (λ_k, y_k) of H_k
  $x_k := V_k y_k$
  $r_k := (\lambda_k I - A)x_k$
  $t_{k+1} := (\lambda_k I - D)^{-1} r_k$, where D =diag(A)
  Orthogonalize $[V_k; t_{k+1}]$ into V_{k+1}.

Notice that normally the computation of r_k would put one into a next higher Krylov space, but the computation of t_{k+1} is an approximate inverse iteration step. Notice that this step would be of most use when D approximates A very well, that is, the matrix is highly diagonally dominant. Sadkane [9] generalize this idea by allowing any matrix to be used here.

With such methods, the smallest eigenvectors of matrices of sizes into the millions have been calculated.

9.2 Nonlinear eigenvalue problems and molecular dynamics

Another technique for solving molecular dynamics problems uses the so-called local density approximation. John Weare at the University of California in San Diego is leading a large team with high ambitions of performing such calculations on parallel supercomputers. The approximation leads to nonlinear eigenvalue problems of the form

$$A(y(X))X = X\Lambda\,,$$

where A is an n-by-n matrix function of a vector y, X is an n-by-k matrix of selected eigenvectors, and the ith component of $y(X)$ is $\sum_{j=1}^{k} |X_{ij}|^2$.

9.3 The nonsymmetric eigenvalue problem

There appear to be few applications currently solving large dense nonsymmetric eigenvalues problems for $n > 10,0000$. One of the larger problems that has been solved was a material science application described in Nicholson and Faultner ([29]), but the respondent gave me very little information about this application.

10 Are algorithms $O(n^3)$, $O(n^2)$, or $O(\log n)$ anymore?

Numerical linear algebraists tend to know the approximate number of floating-point operations for the key dense algorithms. Often the exact constants are known, but here we consider the asymptotic order. Thus, the dense symmetric eigenvalue problem requires $O(n^3)$ operations to tridiagonalize a matrix and $O(n^2)$ operations to then diagonalize. Traditionally, the $O(n^2)$ algorithms are considered negligible compared with the $O(n^3)$ algorithms. This, of course, assumes both algorithms can run at the same rate. This assumption is far from being a given in modern computation. Vector and parallel architectures require that we no longer so quickly dismiss the $O(n^2)$ part of an algorithm as irrelevant.

A complexity term that I find particularly insidious is the description of certain algorithms as having parallel complexity $O(\log n)$. The simplest example is the summing of n numbers on a binary tree or hypercube with $n = 2^k$ leaves/nodes. Strictly speaking such an algorithm requires $\log n$ parallel operations, but referring to the algorithm as an $O(\log n)$ algorithm ignores the modern realities of parallel computing. Big parallel machines are expensive, and it is a terrible waste of resources to run an algorithm with one element per processor:

THE MYTH OF FINE GRAINED PARALLELISM: Nobody computes with one element per processor. If you only have one element per processor, why are you using such a big machine?

More realistically, the ability to vectorize on a smaller machine with more data per processor far outweighs the ability to parallelize. If there are m summands per processor, the number of parallel operations becomes $(m - 1) + \log n$. The more elements per processor, the more likely the $O(m)$ part of the algorithm will overtake the $\log n$ portion.

Other examples of this phenomenon are parallel prefix, reduction, and spreads on trees or hypercubes. The parallel prefix operation may be defined on a sequence $\{x_i\}$ as

```
PARALLEL PREFIX
 function x :=scan(x):
 if length(x)> 1,
     x_{2i} := x_{2i} + x_{2i-1} (all i)
     {x_{2i}} :=scan({x_{2i}}) (recursive step on the even components)
     x_{2i+1} := x_{2i} + x_{2i+1} (all i)
 endif
```

There are further complications if the information is to be available in the right place on a tree or hypercube, but the important idea in parallel prefix is readily seen above. (By describing the algorithm on a tree, I suspect that many authors "miss the forest for the tree.") When the algorithm completes, x_i will contain the partial sum of the original x up to x_i. On a real machine, if there are m numbers per processor, the local sum of elements is computed serially in each processor, the parallel prefix is performed, then the result is added to form all the partial sums per processor. If there are m elements per processor, the number of parallel operations is roughly $2m + 2\log n$. Indeed, if the $\log n$ is going to be felt, it will be because this part of the operation does not vectorize and the communication overhead is expensive. In conclusion, the $\log n$ part is the bad part of the algorithm, not the good part.

11 Conclusion

This article contains a snapshot of large dense linear algebra as it appears in 1993. I received more than 100 messages in direct response to my survey questions. I wish to gratefully acknowledge and thank everyone for their information. I look forward to observing the future of supercomputing and linear algebra.

12 Acknowledgement

This work was supported in part by the Applied Mathematical Sciences Subprogram of the Office of Energy Research, U.S. Department of Energy, under contract DE-AC03-76SF00098.

References

[1] D.A. Anderson, J.C. Tannehill, and R.H. Pletcher, *Computational Fluid Mechanics and Heat Transfer,* McGraw-Hill, New York, 1984.

[2] K. Atkinson, A survey of boundary integral equation methods for the numerical solution of Laplace's equation in three dimensions, in *Numerical Solutions of Integral Equations,* ed. by M. Golberg, Plenum Press, New York, 1990.

[3] A. Bendali, Numerical analysis of the exterior boundary value problem for the time-harmonic Maxwell equations by a boundary finite element method. *Math. of Comp. 43* (1984), 29–68.

[4] S. Bettadpur, V. Parr, B. Schutz, C. Hempel, Large least squares problems and geopotential estimation from satellite gravity gradiometry. *Supercomputing '92.* Los Alamitos, Calif.: IEEE Computer Society Press.

[5] C.A. Brebbia, ed., *Topics in Boundary Element Research, Volume 1: Basic Principles and Applications.* Springer-Verlag, New York, 1984.

[6] F.X. Canning, Sparse approximation for solving integral equations with oscillatory kernels, to appear. *SIAM J. Sci. Stat. Comp.,* in press.

[7] F.X. Canning, The impedance matrix localization (IML) method for moment-method calculations, *IEEE Antennas and Propagation Magazine* (October, 1990).

[8] F.X. Canning, Sparse matrix approximations to an integral equation of scattering, *Communications in applied numerical methods, 6* (1990b), 543–548.

[9] M. Crouzeix, B. Philippe, and M. Sadkane, The Davidson Method. CERFACS Technical Report TR/PA/90/45, December, 1990.

[10] G. Cybenko and D.J. Kuck, Revolution or evolution? *IEEE Spectrum,* 29:39-41 (September 1992).

[11] E.R. Davidson, The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. *J. Computational Physics,* 17:817–825 (1975).

[12] J.W. Demmel, M.T. Heath, and H.A. van der Vorst, Parallel numerical linear algebra, *Acta Numerica 1993,* A. Iserles, ed., Cambridge University Press, 1993, to appear.

[13] J.W. Demmel and K. Veselić, Jacobi's method is more accurate than QR. *SIAM J. Mat. Anal. Appl.* 13:1204–1246. (Also LAPACK Working Note No. 15.)

[14] J. Dongarra, I. Duff, D. Sorensen, H. van der Vorst, *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM, Philadelphia, PA, 1991.

[15] A. Edelman, *Eigenvalues and Condition Numbers of Random Matrices*, PhD thesis, Department of Mathematics, Massachusetts Institute of Technology, 1989.

[16] A. Edelman, Guess what? We have a hypercube. Internal Report. Thinking Machines Corporation, Cambridge, 1989.

[17] A. Edelman, The first annual large dense linear system survey, *SIGNUM Newsletter,* 26:6–12 (October 1991).

[18] R.W. Freund, G.H. Golub, and N.M. Nachtigal, Iterative solutions of linear systems. In, A. Iserles (ed.), *Acta Numerica 1992.* Cambridge University Press, 1992.

[19] K. Gallivan, M. Heath, E. Ng, J. Ortega, B. Peyton, R. Plemmons, C. Romine, A. Sameh, and R. Voigt, *Parallel Algorithms for Matrix Computations*, SIAM, Philadelphia, PA., 1990.

[20] A. Greenbaum, L. Greengard, and G.B. McFadden, Laplace's equation and the Dirichlet-Neumann map in multiply connected domains, *Journal of Comp. Phys.,* 105(2): 267–278 (1993).

[21] R. Harrington, Origin and development of the method of moments for field computation, *IEEE Antennas and Propagation Magazine*, June 1990.

[22] J.L. Hess, Panel methods in computational fluid dynamics, *Annual Rev. Fluid Mechanics,* 22:255–274 (1990).

[23] J.L. Hess and M.O. Smith, Calculation of potential flows about arbitrary bodies, in *Progress in Aeronautical Sciences, Volume 8*, ed. by D. Küchemann, Pergamon Press, London, 1967.

[24] C.T. Ho, S.L. Johnsson, A. Edelman, Matrix multiplication on hypercubes using full bandwidth and constant storage, *The Sixth Distributed Memory Computing Conference Proceedings,* 447–451. IEEE Computer Society Press (1991),

[25] C. Johnson, *Numerical solution of partial differential equations by the finite element method,* Cambridge University Press, Cambridge, 1987.

[26] W. Lichtenstein and S.L. Johnsson, Block-Cyclic dense linear algebra. *Siam J. Sci. Stat. Comp.,* in press.

[27] M.L. Mehta, *Random Matrices,* Academic Press, New York, 1991.

[28] C. Moler and I. Shavitt, Numerical Algorithms in Chemistry: Algebraic Methods, Report on the Workshop. Lawrence Berkeley Laboratory Technical Report 8158 (1978).

[29] D.M. Nicholson and J.S. Faultner, Applications of the quadratic Korringer-Kohn-Rostoker band-theory method. *Physical Review B II, 39*, 8187–8192 (1989).

[30] B.N. Parlett, *The Symmetric Eigenvalue Problem.* Prentice-Hall, Englewood Cliffs, NJ, 1980.

[31] J.K. Prentice, *A Quantum Mechanical Theory for the Scattering of Low Energy Atoms from Incommensurate Crystal Surface Layers*, PhD thesis, Department of Physics, The University of New Mexico, 1992.

[32] Y. Robert, *The impact of vector and parallel architectures on the Gaussian elimination algorithm.* Wiley, New York, 1990.

[33] D.W. Schwenke and D.G. Truhlar, Localized basis functions and other computational improvements in variational nonorthogonal basis function methods for quantum mechanical scattering problems involving chemical reactions. In, *Computing Methods in Applied Sciences and Engineering*, R. Glowinski and A. Lichnewsky (eds.), 291–307. SIAM, 1990.

[34] I.H. Sloan, Error analysis of boundary integral methods, in A. Iserles, ed., *Acta Numerica,* pp. 287–340. Cambridge University Press, 1992.

[35] J.J.H. Wang, *Generalized Moment Methods in Electromagnetics.* John Wiley & Sons, New York, 1991.

[36] J.H. Wilkinson, *The Algebraic Eigenvalue Problem.* Clarendon Press, Oxford, 1965.