An Order N Algorithm for the Solution of the Boundary Integral Equations of Potential Flow

F. T. Korsmeyer
Department of Ocean Engineering
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139
U.S.A.

INTRODUCTION

The proliferation of supercomputing resources is causing many researchers in ocean engineering to contemplate the routine computation of fully nonlinear, three-dimensional, free-surface-wave/body interactions. The achievement of this particular goal may be some years away, but many of the theoretical and numerical issues have been successfully addressed in two-dimensional and axisymmetric model problems. In spite of the available supercomputing power, a key difficulty of the problem which remains is the substantial computational effort required for even a modest problem. The goal of the research described here is to develop a fast algorithm for the computationally "hottest" part of the problem. A constraint on the design of this algorithm is that it must be suited to the fastest architectures likely to be available in the near future.

N-BODY PROBLEMS

The algorithm we are developing for these free-surface problems is a derivative of those developed for the simulation of the interactions of many bodies common in celestial mechanics, molecular dynamics, and plasma physics. Consider an "N-body" problem for point charges:

-$O(10^4)$ particles of known charge are distributed in some fashion in a two- or three-dimensional space;

-their pair-wise interactions are governed by Coulomb's law (the potential is proportional to $\log r$ in two dimensions or $1/r$ in three dimensions);

-at some reference time, the particle positions and velocities are known, and we wish to know the positions and velocities for subsequent times.

If this problem is solved in the obvious $O(N^2)$ manner, then at each time-step in the simulation we would construct an $N \times N$ matrix of influence coefficients, apply this matrix to the vector of known charges to get the potential at each particle, and use this to update the particle positions and velocities. Note that there are two processes which require $O(N^2)$ effort: construction of the matrix, and the application of this matrix to the vector.

It is easy to see that the computational effort required by this simulation may be reduced to $O(N \log N)$ by sorting the particles in such a way that the influence of a group of particles, all of which are in some sense far from a particular particle, may be approximated as the influence a single particle located at the centroid of the group and having a charge equal to the sum of the charges of the particles in the group. Consistent approximation allows distant groups to include larger regions of the space than ones comparatively closer to the particular particle. Programs which make use of this idea bear the generic name of "tree codes" because of the nature of the sorting inherent in their data structure.

A series of papers published by L. Greengard and V. Rokhlin [(1987), for example] demonstrated that the computational effort of two-dimensional N-body problems could be reduced to $O(N)$. Besides the reduction in computational effort for large N, their method has the additional benefit that it is exact. The method makes use of a collection of theorems and lemmas which establish the validity of representing the potential due to groups of particles by multipole expansions, of shifting the centers of these expansions and of rewriting these expansions as power series.

The following is an outline of the method:

–the space occupied by the particles is mapped to a unit computational square;

–the square is subdivided into four "child" squares, each of these is subdivided, and so on, building a tree-like hierarchy of meshes;

–in each "leaf" (finest grain square), the influence due to the particles therein is represented by a multipole expansion about the center of the square which is valid *outside* the smallest circle which contains the square;

–working toward the root, in every square at every level above the leaves, the multipole expansions from the child squares are shifted to the center of the "parent" squares and summed;

–at a coarse level, the multipole expansions are rewritten in their valid regions as power series about the centers of squares in those regions, and these series are valid *inside* the smallest circle which contains the square;

–working toward the leaves, these power series are shifted from the centers of the parent squares to the centers of their children;

–finally, at the leaves, the power series may be evaluated to find the total potential at each particle.

Note that in this method, neither of the $O(N^2)$ operations ever take place: we do not form the matrix of influence coefficients and we do not directly apply any matrix to the vector of charges.

For a specified tolerance in the accuracy of the computation, the highest level of refinement of the hierarchy of squares and the number of terms to retain in the expansions may be determined *a priori*. Neither of these parameters depends on N, a fact which guarantees that this is an $O(N)$ not $O(N \log N)$ method. The extension of this method to three-dimensions is demonstrated in Zhao (1987).

AN $O(N)$ ALGORITHM FOR LAPLACE'S EQUATION

Consider the water-wave/floating-body problem we would like to be able to solve: the domain is unbounded, the free-surface is nonlinear (with ambient waves), the body geometry and motions are unrestricted and a potential flow is assumed . Our approach to a computational solution is to restrict the nonlinear domain to a region close to the body with the remainder treated to some appropriate order by a perturbation scheme. In the nonlinear region, the flow is governed by Laplace's equation, a nonlinear free-surface condition, a Neumann condition on the body, and a matching condition on the boundary enclosing the nonlinear domain. The technique of solution then consists of numerically integrating the evolution equations on the free surface and using the updated free-surface quantities, and body and matching boundary conditions, to solve Laplace's equation at each time step. Almost all of the computational effort will be expended in the solution of Laplace's equation.

We are interested, therefore, in developing a rapid algorithm for the solution of Laplace's equation. We can use the techniques derived for the N-body problem if we reformulate the elliptic boundary value problem as a boundary integral equation (BIEM). The boundary value problem is recast (via Green's theorem and the Green function $G = 1/r$) as coupled Fredholm equations, each on one of the boundaries of the nonlinear region. Unlike the linearized problem, here the resulting equations are not entirely second kind, because the free-surface condition is of Dirichlet-type. Now we invoke some cubature scheme, transforming the integral equations into a system of linear equations which may be solved numerically. Without regard to detail, we are confronted with the basic problem of solving the system of equations: $A\vec{x} = \vec{b}$, where A is an $N \times N$ matrix. Let us assume that this system may be solved in considerably less than N steps by a suitable iterative method. This is not a trivial assumption, but is a separate topic of our research about which we have reason to be optimistic. At this stage we have an $O(N^2)$ method because, in general, each step of an iterative method consists of applying the matrix A to one of a recursively generated set of vectors, say $\vec{y}$. But matrix A is very much like the matrix of the N-body problem. It consists of influence coefficients calculated from harmonic functions, regardless of the details of how we discretize the problem, and consequently the underlying ideas of the $O(N)$ algorithm may be applied to BIEM. Rokhlin (1985) illustrates this idea; however, this paper is misleading on critical aspects of the implementation.

There are important differences between a BIEM solution of Laplace's equation and an N-body problem. One of these is that in their exact forms, the BIEM is an integral equation with a singular kernel, but the N-body problem is merely a discrete sum. Another is that BIEM may reduce the dimensionality of the problem by one, but the theorems and lemmas of the $O(N)$ N-body algorithm apply in the original space of the problem.

Consider the model problem for which we have written an $O(N)$ code. We intend to solve the two-dimensional "wave-maker" problem, solved successfully by Vinje and Brevig (1980). In this case, the problem is formulated as a Cauchy integral equation for the *complex* potential, with the real part (the potential) unknown on the solid boundaries and the imaginary part (the stream function) unknown on the free surface. If we choose an N-segment trapezoid rule for the quadrature, the discrete problem appears to be exactly a two-dimensional N-body problem with pair-wise interactions governed by $1/(z_i - z_j)$. This quadrature is an ideal choice for the implementation of the $O(N)$ algorithm, but it is poorly suited to integrals with singular kernels. A more suitable, albeit more complex, procedure is to discretize the problem as is commonly done in the aero/ocean engineering community: with a panel method. Exact integrations of the kernel over panels, and special cases such as self influences and adjacent influences, increase the effort required in computing the multipole expansions for the panels in the leaves, but the efficacy of the panel method in treating these integral equations is well established.

Another aspect of tailoring the $O(N)$ idea to BIEM is to employ the "adaptive" algorithm used for sparse configurations of particles. Once we have discretized the boundary of the wave-maker problem, perhaps with linear segments, the segment ends (nodes) correspond to an extremely sparse array of particles. If we were to build a hierarchy of squares without regard for this sparsity, most of the squares would contain no nodes. Hence the code we have written is adaptive, it produces a tree of squares in which none are empty. This is controlled during the sorting of nodes by never including empty squares and setting a maximum for the number of nodes in any unsubdivided (childless) square.

The fact that the influence coefficients are not computed from simple functions such as $1/(z_i - z_j)$, but rather the integrals of such functions over panels, and the fact that the data structure is not a complete tree increase the computational effort of the $O(N)$ algorithm, on a per-node basis, in the BIEM case over that of the N-body case. This does not, however, negate its efficiency relative to an $O(N^2)$ method for the BIEM.

CONCLUSION

In chosing a two-dimensional Cauchy formulation as a model problem we have eliminated theoretical and numerical uncertainties not related to the $O(N)$ method. For example the treatment of the solid-boundary/free-surface intersection is well established; and the Cauchy formulation is second-kind thereby guaranteeing an iterative solution. We are using this problem to learn how fast the $O(N)$ algorithm is, and how the desired accuracy of the solution affects the time required for its calculation. Preliminary results indicate that for this problem it is faster to compute the application of matrix A to vector $\vec{y}$ by the $O(N)$ algorithm if $N > 10^2$.

We are intending to write an adaptive $O(N)$ algorithm for the Green's theorem formulation of the three-dimensional problem. Along the way, we will write a code for the two-dimensional Green's theorem formulation, and we will investigate the implementation of these schemes on parallel architectures.

REFERENCES

L. Greengard and V. Rokhlin, (1987) A fast algorithm for particle simulations, *J. Comp. Physics*, **73**, pp. 325-348.

V. Rokhlin, (1985) Rapid solution of integral equations of classical potential theory, *J. Comp. Physics*, **60**, pp. 187-207.

T. Vinje and P. Brevig, (1980) Breaking waves on finite water depths. The Norwegian Institute of Technology.

F. Zhao, (1987) An $O(N)$ algorithm for three-dimensional N-body simulations, Research Report AI-TR-995, Massachusetts Institute of Technology, Cambridge.

DISCUSSION

King: In the discretization of the principal value integral you appear to expand the singular kernel about a collocation point. As the mesh grows finer, the nodal points move closer to this collocation point causing you to take more terms in the expansion of the singular integral kernel in order to retain accuracy. What implications does this have for $O(N)$? Does it imply an accuracy of $O(1/N)$?

Korsmeyer: The philosophy underlying our $O(N)$ work is that the iterative method and the $O(N)$ technique for applying a matrix to a vector will not require a modification of the mathematical approach to solving the potential problem. The water-wave community has established a preference for solving Laplace problems by panel methods for a number of well-founded reasons. In the context of that general type of solution method, $O(N)$ will not further restrict your attack.

In reference to your specific point, the $O(N)$ scheme should retain the underlying convergence rate of your method had you simply built a matrix and solved the system by LU decomposition. Take our Cauchy integral formulation for example. For panels 'close' to a field point, we use the exact formula for the integration of a linear variation of the unknown over a straight boundary segment *à la* Vinje and Brevig (1980), where 'close' means panels and field points in the same or neighbouring childless boxes (leaves of the tree). Otherwise, that exact integration formula *itself* is represented by a multipole expansion, which is accurate to as many decimals as specified.

The parameters which will affect the convergence rate (number of terms in the expansion, tolerance for the iterative method, and what is 'close') must be specified in a consistent manner to achieve the desired accuracy with efficiency. We have much to learn on this subject.

Tuck: Regarding parallel architecture, surely benefits are available without going to as many as N processors? You might get benefits from as few as M, where M = number of tree levels.

Korsmeyer: I do not mean to imply that we need $O(N)$ processors. However, I think that the mapping of this algorithm to a parallel architecture is on a box-wise basis, not a level-wise basis. This is because the algorithm moves up and down the tree-like data structure unable to process a level until it has processed the one below or the one above.

Cao: To sort particles, you need many IF statements in the computer code. This prevents that part of the code from vectorization, and can slow down the program significantly. Does it pay off?

Korsmeyer: First, a key point: I do not think vectorization is an important issue. *Parallel* architectures are of interest here. If someday I can work on a massively parallel machine with each processor having a vector architecture, then I will be interested in vectorization of modules of this code.

For the $O(N)$ code, building the data structure should work on a parallel architecture just as well as performing calculations on data in that structure once it is built. In both cases, for any given box, a processor need only know about the one parent box and the up-to-four child boxes. With the 'adaptive' algorithm, there are exceptions to this; a fact which poses interesting challenges.

Schultz: You stated that there is a significant amount of 'accounting' in the development of the algorithm. Does this housekeeping requirement challenge the programmer or the end user? In other words, how user friendly is this code you are going to give us?

Korsmeyer: Like anyone writing code, I have to answer: 'very friendly'. If you are using an iterative method, and you call a routine, say 'MULT', which applies your matrix to a vector, you simply call the $O(N)$ routine instead. You have previously inserted in the $O(N)$ code your own 'close' evaluation routine, and your own expansion of the influence coefficient(s). These routines change depending on your choice of singularities, your assumptions about the representation of the unknown(s) on the boundary, and your representation of the boundary itself.

Grue: What happens to the code when the waves you are computing are breaking? In the same context, how do you model the intersection between the free surface and a floating body, and the effect of local breaking waves?

Korsmeyer: In the 2-D model problem using the Cauchy integral formulation, all of the hydrodynamic/mathematical considerations are handled as in [1]. As I stated above, the $O(N)$ method is independent of these considerations. The method is intended as a powerful tool which will allow rapid and efficient exploration of hydrodynamic phenomena such as those you mention: breaking waves and wave-body intersections.

Kleinman: How do you decide which points go in which box?

Korsmeyer: The nodal points are allocated to boxes in a typical tree sort. This, however, does not require $O(N \log N)$ effort because there are not $O(\log N)$ levels. This is due to the fact that the sort is not 'exact' in the sense of, say, sorting words for a dictionary. To see this, consider that we have set a tolerance for the computation: p. Then no two nodes can be distinguished if they are closer than p, and hence no box can have dimension d less than p. For a unit computational domain, the dimension of a box is simply a function of its level, l. Counting up from the entire domain as level $l = 0$,

$$d = 2^{-l}.$$

From this equation, we find that at finest grain, where d may equal p,

$$l_{max} = |\log_2 p|.$$

So the number of levels is independent of N.

Reference

[1] D.G. Dommermuth, D.K.P. Yue, W.M. Lin, R.J. Rapp, E.S. Chan & W.K. Melville, 'Deep-water plunging breakers: a comparison between potential theory and experiments', *J. Fluid Mech.* **189** (1988) 423–442.