JOURNAL OF COMPUTATIONAL PHYSICS 101, 185–206 (1992)

Two-Dimensional, Viscous, Incompressible Flow in Complex Geometries on a Massively Parallel Processor

J. A. SETHIAN*

Department of Mathematics, University of California, Berkeley, California 94720

AND

JEAN-PHILIPPE BRUNET, ADAM GREENBERG, AND JILL P. MESIROV

Thinking Machines Corporation, Cambridge, Massachusetts 02142

Received July 20, 1990; revised June 17, 1991

We describe the parallel implementation of a numerical method, known as the random vortex method, for simulating fluid flow in arbitrary, complex geometries. The code is implemented on the Connection Machine CM-2, a massively parallel processor. The numerical method is particularly suited for computing complex viscous, incompressible flow across a wide range of flow regimes and characteristics. In this method, the vorticity of the flow is approximated by a collection of particles whose positions and strengths induce an underlying flow. As such, it is a Lagrangian scheme, in which the position of each particle is affected by all others at each time step. The efficient execution of this method on the Connection Machine results from a parallel N-body solver, parallel elliptic solvers, and a parallel data structure for the adaptive creation of computational elements on the boundary of the confining region. Using this method, we analyze the generation of large vortex structures, mixing and shedding under various flow geometries and inlet/outlet profiles. The data from our simulations are visualized using the real-time flow visualization environment developed on the Connection Machine.

INTRODUCTION

In this paper, we describe the development of a code to model two-dimensional fluid flow in general geometries. The random vortex method is used to approximate the equations of viscous, incompressible flow, and the code is implemented on the Connection Machine CM-2, a massively parallel processor with a hypercube interconnection network. Using this method, we model the solution to a variety of complex flow problems. We study the generation of large vortex structures, mixing and shedding in various flow configurations, and analyze the dependence of these structures on domain geometry, inlet/outlet placement, and profiles.

The random vortex method, introduced by Chorin [12], is a particularly complicated method for computing viscous, incompressible flow across a wide range of Reynolds numbers, and can accurately capture the development of large-scale flow structures in the laminar, transitional, and turbulent regimes, see [31]. In this method, vorticity is approximated by a collection of particles whose positions and strengths induce an underlying flow. As such, it is a Lagrangian method, in which the position of each particle is affected by all others at each time step. The data structures that connect elements are complex, and information is passed among all elements at each time step. Consequently, the implementation of this method on a parallel processor is a challenging task, requiring a careful marriage of architecture and algorithm. At several stages, we have significantly altered the problem, method of solution, and implementation to perform efficiently on a massively parallel machine.

Some of the key issues confronted in the design of a parallel implementation of vortex methods were:

(a) The design of efficient parallel N-body solvers.

(b) Accurate elliptic solvers in highly complex geometries.

(c) Efficient run-time data mappings between several processor topologies.

(d) Complex parallel boundary conditions to allow arbitrary geometries as input.

(e) Parallel real-time flow visualization for interactive flow diagnostics.

In particular, the implementation of a vortex code necessitates the solution of both an N-body problem and of

* Supported in part by the Applied Mathematics Subprogram of the Office of Energy Research under Contract DE-AC03-76SF00098. This author also acknowledges the support of the National Science Foundation and the Sloan Foundation.

0021-9991/92 $5.00

Poisson's equation at each time step. The N-body problem requires the evaluation of a particle–particle interaction between all pairs of particles, which is accomplished by means of a "orrery" to cycle through all possible interactions in our parallel environment. The elliptic Poisson solver is solved using an iterative method and is made efficient through the use of communication stencils in the nearest neighbor grid updates. Finally, dynamic reconfiguration of data in the processors is used as the number of Lagrangian elements increases and decreases. Additionally, we employ a real-time visualization environment based on parallel graphics computations which mimics flow diagnostic techniques in laboratory apparatus.

We study flow in a variety of configurations, such as flow in a doubly symmetric step, flow around islands, and flow in a piston/valve. Our results reveal a collection of fascinating flow phenomena. We visualize the results of these calculations using the real-time flow visualization environment developed on the Connection Machine.

PART ONE: BRIEF OVERVIEW OF VORTEX METHODS:

I.A. Equations of Motion

Vortex methods are attractive techniques for calculating viscous, incompressible, turbulent flow. The critical flow quantity in this approach is the *vorticity*, which is the curl of the velocity and represents the amount of rotation at a point in the flow. Instead of an Eulerian finite difference mesh, a Lagrangian approach is taken in which the initial vorticity field is discretized into a large number of vortex elements whose ensuing motion describes the evolution of the flow. At any time, the velocity of the fluid may be recovered from the positions and strengths of the vortex elements.

The advantage of this technique is twofold. First, since no grid is introduced, this Lagrangian approach avoids the introduction of numerical viscosity which swamps the real physical viscosity. Second, the method is dynamically adaptive: computational elements are naturally clustered in regions of high vorticity where flow gradients are large and accuracy is required. As such, vortex methods have proven to be a powerful technique for modeling much of the intricate, complex behavior of turbulent flow (see [29]).

The starting point is the voriticity transport equation in two dimensions for the *vorticity* vector $\xi = \nabla x \mathbf{u}$, namely,

$$\frac{\partial \xi}{\partial t} + (\mathbf{u} \cdot \nabla)\xi = \frac{1}{R}\nabla^2 \xi, \qquad \mathbf{u} = 0 \text{ on } \delta D. \tag{I.1}$$

In order to "close" Eqn. (I.1), we must recover $\mathbf{u}$ from the vorticity. Given that $\nabla \cdot \mathbf{u} = 0$ and $\xi = \nabla x \mathbf{u}$, we know that there exists a vector function $\psi(\mathbf{x})$ such that $\mathbf{u} = \nabla x \psi$ and $\nabla^2 \psi = -\xi$. Thus, we may write ψ, and consequently the velocity $\mathbf{u}$, in terms of ξ by making use of the fundamental solution to the Laplace operator ∇^2. Recall that the solution to this Poisson equation for the stream function is given by $\psi(\mathbf{x}, t) = \int L(\mathbf{x} - \mathbf{z})\, \xi(\mathbf{z})\, d\mathbf{z}$, where $L(\mathbf{x}) = (-1/2\pi) \log |\mathbf{x}|$. Since $\mathbf{u} = \nabla \mathbf{x} \psi$, we have that

$$\mathbf{u}(\mathbf{x}, t) = \int K(\mathbf{x} - \mathbf{z})\, \xi(\mathbf{z})\, d\mathbf{z}, \tag{I.2}$$

where the kernel K is defined by $(1/2\pi)((-x_2, x_1)/|\mathbf{x}|^2)$.

If we envision the initial condition $\xi(x, 0)$ as describing the vorticity of the particle initially located at $\mathbf{x}$, we may then ask for the ensuing motion of particles located at all possible starting points. This leads to a *Lagrangian* formulation for the particle trajectories, which we now formulate. Assuming inviscid flow for the moment, we may numerically approximate the Lagrangian formulation by following the evolution of a discrete number of particles, each carrying vorticity. Unfortunately, since the kernel is singular, as particles come close together, they can exert extremely large velocities on each other which can result in numerical instability. The essential numerical idea, introduced by Chorin [12], is to replace the singular kernel by a smoother one, obtained by convolving K with a smoothing function. The original smoothed kernel chosen simply set the radial velocity inside a given cutoff size δ to be constant, thus eliminating the singularity. The first convergence proof for a vortex blob method was given by Hald [20]. Since that time, a large number of smoothed kernels have been constructed, providing vortex methods of various orders of accuracy. For analysis of the various properties of cores, see [8–10, 19–21, 26]. For the rest of this section, we shall simply assume that the singular kernel K has been replaced by an appropriate smoothed kernel K_δ.

Thus, for two-dimensional flow, the basic idea behind vortex methods is to discretize in both space and time the initial value problem described by the evolution of a discrete set of particles. Given an initial vorticity distribution $\xi(\mathbf{x}, 0)$, we begin by constructing a lattice Λ^h in R^2, with mesh size h. Let $\mathbf{j}h$ be the mesh points of Λ^h, where $\mathbf{j}$ is an ordered pair with integer coefficients. Then the motion (Eq. (I.2)) of the set of particles originally located on the nodes of this lattice may be approximated by

$$\frac{d\mathbf{X}(\mathbf{j}h, t)}{dt} = \sum_{\mathbf{i}h \in \Lambda^h} K_\delta(\mathbf{X}(\mathbf{j}h, t) - \mathbf{X}(\mathbf{i}h, t))\, \xi(\mathbf{X}(\mathbf{i}h, t), t)\, h^2. \tag{I.3}$$

Equations (I.3) form a finite system of coupled differential equations. The time derivative is approximated by a suitable finite difference operator to provide a complete algorithm

for updating the positions of the particles in time. This completely specifies the vortex method for inviscid flow.

The extension of vortex methods to viscous flow requires treatment of the viscous diffusion term $1/R\,\nabla^2\xi$ in Eq. (I.1). We follow the technique introduced in [13] and allow the vortex elements to undergo a random walk to simulate viscous diffusion. Thus, to accomplish both advection and diffusion, we update the positions of the infinite system of particles by (1) advancing them by their induced velocity field and (2) adding an appropriately chosen random step. Details about the random walk approximation to the viscous term in vortex methods may be found in [13, 16, 17, 30]. Convergence proofs of various aspects of the inviscid and viscous vortex method may be found in [8–10, 19–21, 23].

For both inviscid and viscous flow, the addition of the normal boundary condition is conceptually straightforward. Let $\mathbf{u}_{\text{vor}}(\mathbf{x}, t)$ be the velocity field obtained from the distribution of vorticity. Given a region D, the normal boundary condition requires that $\mathbf{u}\cdot\mathbf{n}=0$ on the boundary ∂D, where $\mathbf{n}$ is the inward normal vector. Suppose we find a potential flow $\mathbf{u}_{\text{pot}}=\nabla\phi$ such that $(\mathbf{u}_{\text{pot}}+\mathbf{u}_{\text{vor}})\cdot\mathbf{n}=0$ on ∂D. Then superposition of the vorticity flow $\mathbf{u}_{\text{pot}}$ with the potential flow $\mathbf{u}_{\text{vor}}$ yields a flow which satisfies the normal boundary conditions by construction and has the same vorticity (since $\nabla x\,\mathbf{u}_{\text{pot}}=\nabla x\,\nabla\phi=0$). Note that this potential function must be found at every time step. Some possibilities are conformal maps, the method of images, and fast Poisson solvers. In the case of complex geometries, considerable work may be involved in constructing the potential flow.

The addition of the tangential no-slip condition ($\mathbf{u}\cdot\tau=0$) (τ is the unit tangent to solid walls) on ∂D for viscous flow adds considerable complexity. A thin boundary layer transition zone must develop between the vanishing tangential velocity at the wall and the rapidly moving flow away from the wall. Thus, we must create and release vorticity from the solid. One technique for doing so was introduced in [13] and described in detail in [30]. Close to solid walls, the full Navier–Stokes equations are replaced by the Prandtl boundary layer equations, which are derived under the assumption that $\partial v/\partial x \ll \partial u/\partial y$, and that diffusion of vorticity occurs mostly in a direction normal to the wall. Thus, vorticity in the boundary layer is discretized by finite length vortex sheets. The jump in the tangential velocity across the sheet determines its strength. Just as for the vortex blobs, the velocity of each sheet can be constructed by summing the influence of all sheets located in a narrow neighborhood nearby. Thus, we may derive a force law for the velocity induced on a vortex sheet by others (a full derivation may be found in [34]). Imagine a collection of k vortex sheets, which are short line segments parallel to the wall across which the tangential velocity jumps. We define the strength ξ_k of each sheet k, $1\leqslant k\leqslant N$, located at (x_k, y_k), to be the negative of the tangential velocity above, minus the tangential velocity below. That is,

$$\xi_k(x_k, y_k) = -(u^+(x_k, y_k) - u^-(x_k, y_k))$$

where the superscript + and − refer to the limit from above and below, respectively, and u is the velocity tangential to the solid walls. We represent the velocity as seen at ∞ in the Prandtl boundary layer equations by a line of vortex sheets at the edge of the boundary layer with strengths at each time step equal to the current tangential velocity induced by the interior vortex blobs plus the potential function. Then the single force law that gives the u component of the velocity at vortex sheet i, given by all the others in the boundary layer, is just

$$u(x_i, y_i) = \sum_{j=1}^{N} \xi_j \max(0, 1-|x_i - x_j|/h) \times\left(\frac{1+\operatorname{sign}(y_j - y_i)}{2}\right). \tag{I.4}$$

Using the incompressibility of the flow, a similar expression may be derived for the normal component v of the sheet velocity, namely,

$$v(x_i, y_i) = -(1/h) \sum_{\text{all sheets } j=1, N} [\xi_j \min(y_i, y_j) \times \max(0, 1-|x_i + h/2 - x_j|/h) - \max(0, 1-|x_i - h/2 - x_j|/h)]. \tag{I.5}$$

Finally, in order to satisfy the no-slip condition along the solid walls, we create sheets at discrete points along the boundary whose strengths exactly cancel the existing tangential velocity, satisfying the no-slip condition. Complete details may be found in [29, 34].

To summarize, at each time step we must:

(1) Calculate two N-body problems (interior and boundary)

(2) Solve a potential flow for the no-flow condition

(3) Dynamically create vortex elements for the no-slip boundary condition.

I.B. Implementation

In this section, we outline how the various stages of the vortex method may be efficiently mapped onto the different communication patterns of the Connection Machine CM-2 (for details about the Connection Machine, see Appendix A). Here, we only note that for structured communication patterns, the machine processors may be naturally configured as a k-dimensional grid with nearest neighbor communications.

To begin, we wish to accommodate arbitrary rectilinear geometries. Thus, to initialize the geometry and the flow conditions on input, we configure the processors as a two-

dimensional rectangular grid. The bounding confinement geometry is then drawn on grid lines, with those processors corresponding to points within the flow geometry being active. Various flags then determine whether a processor is located inside the domain, on the boundary, or outside the domain.

The vortex elements are stored as arrays with the processors configured as a one-dimensional grid. The number of elements in this array can rise and fall as vortex elements are both created at solid walls and destroyed as they leave the computational domain. This can leave holes in the array as processors corresponding to exiting elements are turned off. To efficiently utilize this array, we perform "garbage collection" which removes these holes and collapses the list to include only the active processors.

The N-body problem is solved by a parallel implementation of the direct method. Thus, we evaluate all possible pairwise interactions using the exact Biot–Savart force law. There are several reasons why we chose a direct method, rather than a fast summation technique, such as local corrections [5], hierarchical models [3, 7], and multi-pole techniques [18] for approximating the interaction. First, we wanted to perform a careful study of the speedup that would result from a parallel implementation of the direct method, to provide comparison with a serial implementation. Second, the interaction between computational elements in the boundary layer is extremely complex, including several Heaviside functions and switches, unlike the straightforward inverse distance force law in the interior. These switches locate nearby computational elements, which are the only ones that contribute in the boundary force law expression. The original boundary layer algorithm by Chorin [13] evaluated these switches by computing the distance between all pairs of elements, which is an $O(N^2)$ operation, see [30]. Baden and Puckett [5] perform a bin mechanisms to efficiently locate nearby elements, but this becomes intractable for the highly complex geometries under consideration here. While a multi-pole type expansion might be appropriate for the force law in the interior, to the best of our knowledge the intricate boundary force law cannot be easily cast in such a framework. Consequentely, we chose a different approach, namely, to pass *all* the computational elements to the N-body solver, and rely on the efficiency of our parallel N-body solver to calculate the appropriate interactions. Our solver relies on a replicated "orrery" to cycle through all the possible interactions. In this technique, the processors are configured as a three-dimensional grid, with the location and strength of the vortex elements stored in the memory of the processors on the front face of this three-dimensional cube. Copies of this data are then spread to the other processors throughout the cube in such a way that all vortex–vortex interactions can be accumulated by nearest neighbor communications through the depth of the cube. (A different technique for evaluating the N-nody interaction by means of an all-to-all broadcast is described in Appendix B.)

Finally, we must solve for the potential flow at each time step. Of course, a large number of techniques are available for solving such problems, including direct methods and conformal maps. Our consideration of arbitrary rectilinear geometries and massively parallel architectures leads quite naturally to iterative techniques for three reasons. First, given highly complex geometries with the possibility of thin narrow regions, multiple 90° corners and interior islands, the linear system associated with a finite difference approach is easy to formulate when compared with image techniques or conformal maps. Second, the fast nearest neighbor communication in a parallel setting allows the simultaneous update of all interior points every step of the iteration. Finally, while iterative techniques require good initial guesses, an excellent starting value is provided by the solution to the potential flow at the previous time step. For our initial implementation, we chose overrelaxation with red–black ordering and Chebyschev acceleration. (For an implementation involving conjugate gradients techniques, see Appendix B.)

PART TWO: PARALLEL DATA LAYOUT, N-BODY SOLVERS, AND ELLIPTIC SOLVERS

II. Input: Organization of Input of Arbitrary Geometry and Flow Conditions

In this section, we describe the organization and data mappings of the input of arbitrary geometry and flow conditions. We consider the processors of the machine connected as a two-dimensional grid and label this the "compute region." The user may specify any rectilinear geometry, complete with interior islands, lying on grid lines within the compute region. The user also specifies the location of inlet and outlet locations, as well as inflow/outflow profiles. The setup is shown in Fig. 1.

Our basic plan is as follows. We tag the processors according to their location relative to the flow geometry within the compute region. These "processor tags" will be used in several ways:

(1) The iteration for the Poisson solver is performed over the entire compute region. However, masks built out of the processor tags will construct the correct stencil update at the boundary of the flow geometry. Thus all processors within the compute region may be updated simultaneously in parallel, with no special attention applied to boundary points.

(2) Processsor tags along the boundary are used to determine the local orientation and interaction of vortex sheets within the numerical boundary layer.

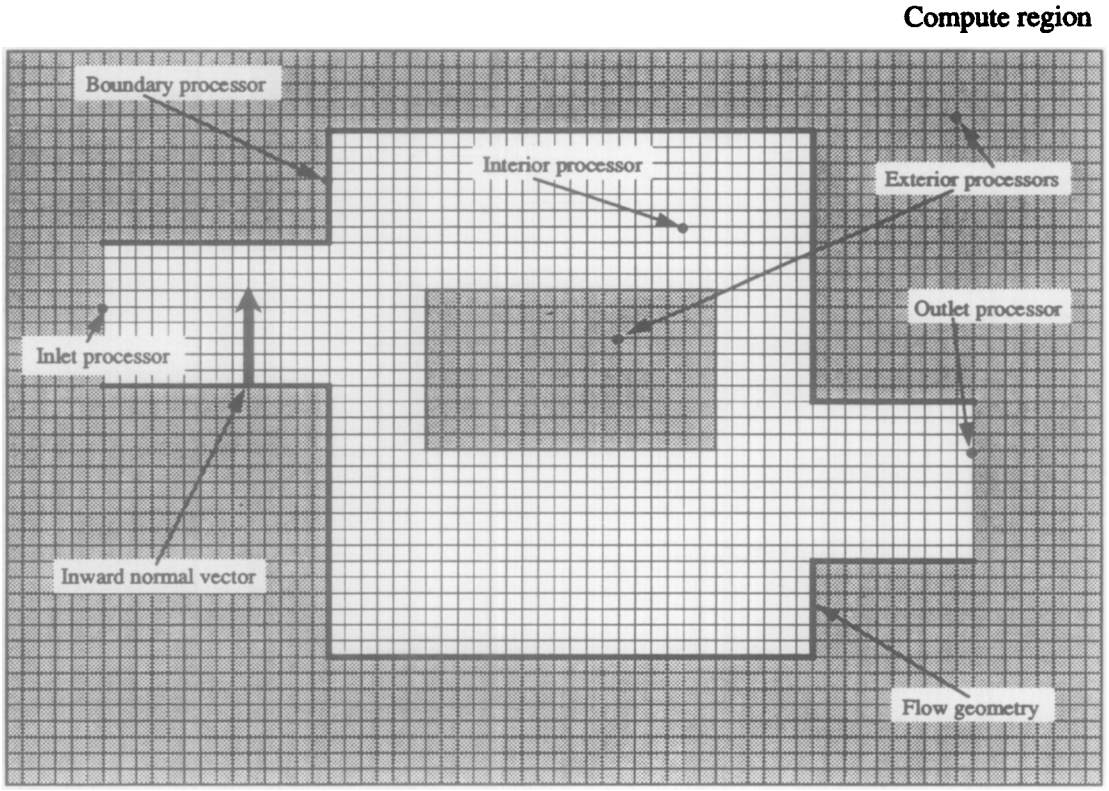

FIG. 1. Relationship between compute region and flow geometry.

(3) Processor tags are used to locate the vortex elements as they move and determine whether they should be traded across the numerical boundary layer, reflected across solid walls, or eliminated as they leave the flow geometry.

Thus, the processor tags connect the underlying flow geometry with the moving vortex elements. These tags are built as follows: First, the processors are tagged corresponding to their location, either interior, boundary, or exterior. The boundary processors are then divided into solid wall processors or inlet/outlet processors. Next, all boundary processors are assigned a unit vector (bx, by) pointing in the inward normal direction. Next, tags are built to determine the distance from a processor to the boundary, to be used in determining the transition point from sheets to blobs and vice versa. Finally, an input variable (flow-value) determines whether or not the segment is a solid wall or an inlet/outlet. If no argument is supplied, then the segment is a solid wall. If a real number value is prescribed, then the flow is assumed to be moving through that segment with velocity vector (*flow-value* $|bx_{i,j}|$, *flow-value* $|by_{i,j}|$). Note that a positive value for "flow-value" means that the fluid is moving to the right (if $by_{i,j}=0$), or up (if $bx_{i,j}=0$), independent of whether this motion carries fluid in or out of the domain.

III. The N-Body Solver

In this section, we describe the implementation of the N-body solver on the CM-2 parallel architecture. Two calls to the N-body solver are required: one for the vortex blobs and one for the vortex sheets. Because the only difference between the two calls is the particular force law, we focus on N-body solvers for a general force law.

A direct implementation of an N-body algorithm on a serial machine requires $O(N^2)$ operations, since all pairwise interactions must be computed. At the other extreme, imagine a parallel computer with N^2 processors. In this case, each processor can perform one of the pairwise interactions, thus all the force interactions can be computed simultaneously in $O(1)$ time, and the complexity of the algorithm depends only on the communications involved in summing over the computed interactions. In a hypercube interconnection scheme, this can be done in $O(\log N)$ time.

In between these two extremes lies another canonical case, where one has the same number of processors P as bodies N. Imagine that $N=P$ processors are connected as a ring and that processor i accumulates the forces on body i. At each time step, N of the pairwise interactions are computed, then the appropriate data are passed around the ring so that in N steps all N^2 interactions have been computed and accumulated into the correct processor (see Fig. 2a).

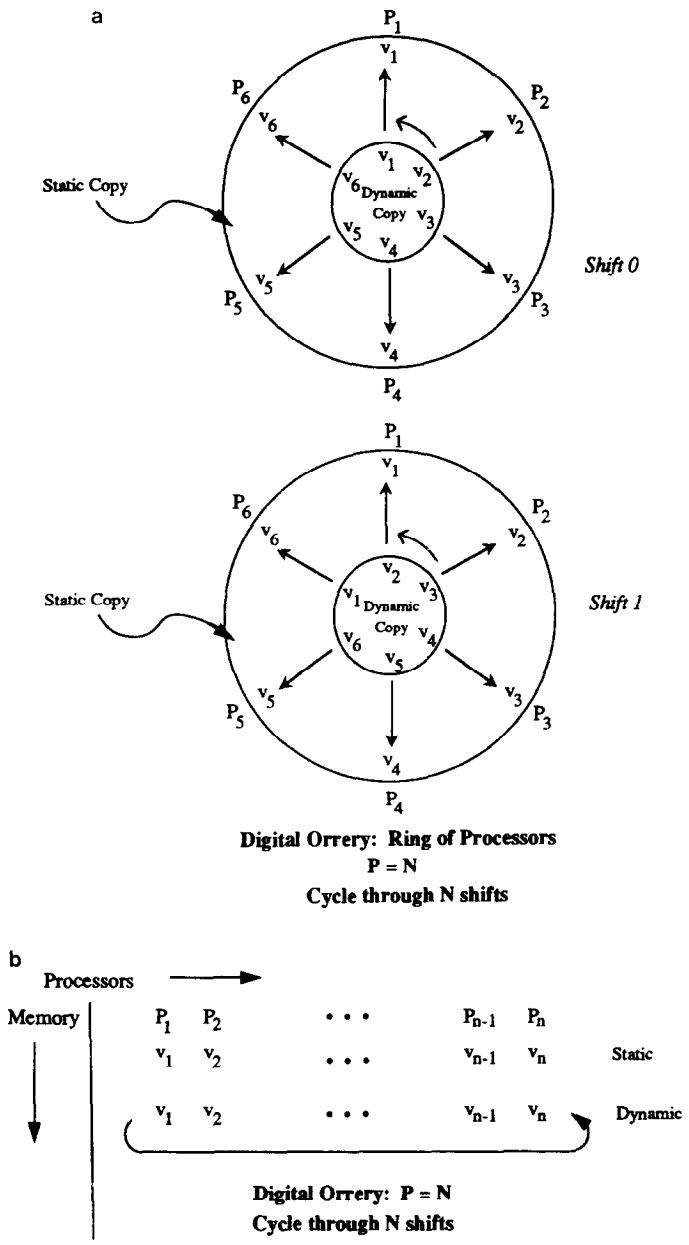

FIG. 2. Digital orrery, $P = N$: Ring; List 2(b).

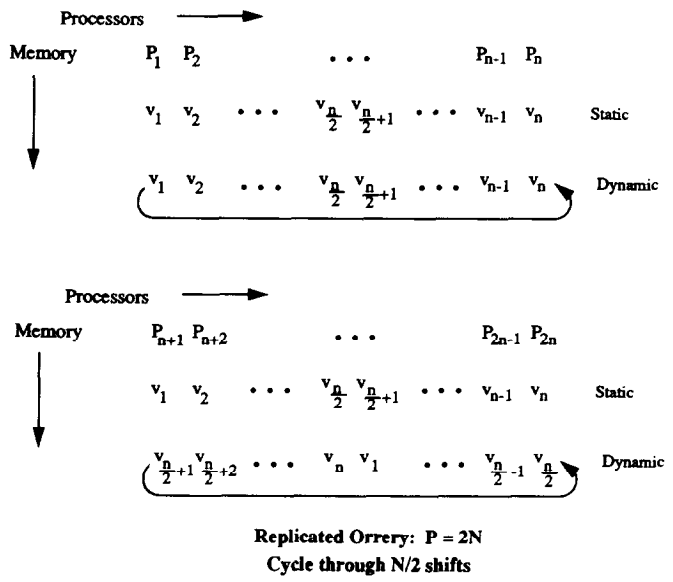

FIG. 3. Replicated orrery, $P = 2N$.

This particular parallel implementation of the N-body algorithm has been referred to as the "digital orrery" in [4], which describes a hardware implementation of the algorithm. The name makes allusion to an apparatus for representing the motions and phases of the planets and moons in the solar system. A different drawing of the same process is given in Fig. 2b. If we have $2N$ processors the computation takes place on two rings of processors each of which passes its data only $N/2$ times. This case is shown in Fig. 3. More generally, if we have MN processors, then we use M rings of N processors each. At each computation step, MN interactions are computed, and N/M steps with cyclic message passing of the data are required to compute the N^2 interactions.

An efficient processor topology comes from mapping these replicated orreries onto a three-dimensional cube of processors, as observed in [25]. The dimensions of this cube of processors can be easily changed to accommodate a family of parallel N-body solvers, where the number of processors ranges from N to N^2. In particular, if we have MN processors, the solver runs in $O(N/M) + O(\log M)$ time assuming that only $O(\log M)$ communications are needed to sum the contents of M processors. The latter is true, for example, if the cube is embedded in a larger dimensional hypercube, which is exactly the case for a Connection Machine computer. This gives us the correct complexity bounds for the extremes, i.e., when $M = 1$ we obtain $O(N)$ and for $M = N$ we obtain $O(1) + O(\log N)$. Let us point out another view of the complexity of the algorithm. Note that $NM = P$, where P is the number of processors. In fact it is P that is usually fixed, given some particular piece of hardware. In terms of P, the complexity bound is $O(N^2/P) + O(\log P/N)$. Since $N \leqslant P \leqslant N^2$ for the fieldwise model of the CM-2 (see (Appendix A), the former notation is appropriate.

The cube data mapping is critical to the code since the ratio of vortices to processors is dynamic; the number of vortices increases as they are shed from the boundary into the interior of the domain and decreases as they exit the domain through outlets. We can take advantage of the ability to dynamically reconfigure the CM-2 as three-dimensional grids of varying sizes. Thus, the dimensions of the three-dimensional cube change in response to the changing number of vortex bodies. Moreover, this model facilitates porting the code to different sizes of CM-2s.

Through its virtual processor mechanism, the CM-2 allows the user to transparently emulate a machine with many more processors than are physically present.

In configuring the CM-2 as k-dimensional grids, we are required to restrict the lengths in each dimension to a power of 2. Thus, the data mapping of the replicated orrery onto a three-dimensional grid proceeds as follows. Pick n such that 2^n is the first power of 2 greater than or equal to N. Then we will actually use an orrery of size 2^n with only N valid entries. We are assuming that $N \leqslant P \leqslant N^2$, so we can write $P = 2^{n+m}$, where $M \leqslant n$. We will want to replicate the orrery $M = 2^m$ times. We configure a 2^{n+m} processor CM-2 as an 2^{n-m} by 2^m by 2^m cube (N/M by M by M). This configuration allows us to use versions of the parallel prefix **scan** operations (Appendix A) to spread or accumulate data along any of the three coordinate directions in logarithmic time and ensures the correct wrap-around for grid communications. The cube mapping, complete with sends, spreads, and accumulates, is drawn in Fig. 4.

There will be two copies of the data for each body, one dynamic and one static. The static data is initially resident on the front face of the cube. It is sent to the top of the cube by a low density **send**. The data on the front face is spread back along the y-dimension of the cube, and the data on top is spread down along the z-dimension of the cube. The data is now distributed through the cube so that each plane parallel to the xz-axis contains a copy of the orrery. The computation now proceeds by computing a pairwise interaction according to the relevant force law and doing a one-dimensional wrap-around NEWS communication along the x-axis. This process is repeated 2^{n-m} times until all N^2 pairwise interactions have been completed. Now the contributing forces for each individual body are accumulated by doing a **plus-scan** along the y-axis back to the front face of the cube.

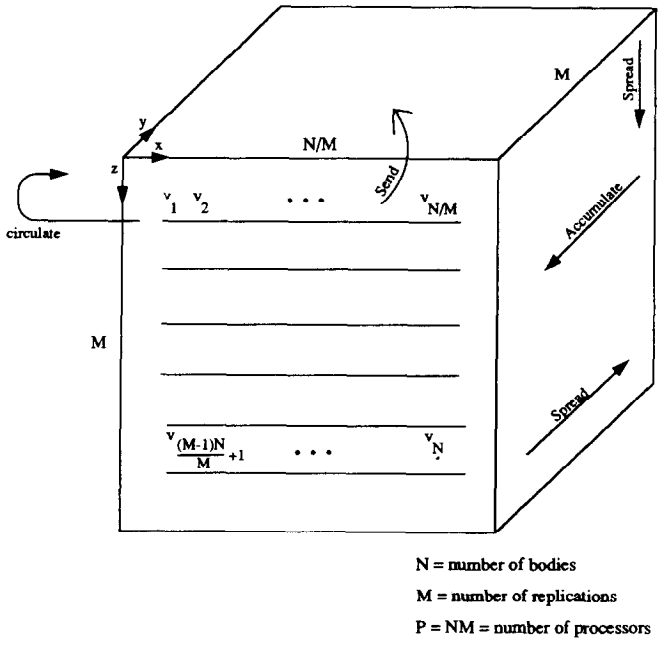

FIG. 4. Cube data mapping.

Because of the power of two restrictions noted above, the time required to do the computation for any value of N between 2^n and $2^{n+1}-1$ is the same. As N increases above the next power of 2, we can reconfigure the grid to the appropriate size.

IV. The Poisson Solver

A method to solve Poisson's equation is required at two stages in the vortex method: first, to compute the potential flow which by superposition with the vorticity-induced velocity field satisfies the no-flow condition, and second, as part of a vortex-in-cell calculation to compute the stream function for display purposes. Given the configuration of the processors in a two-dimensional grid covering the compute region, we have chosen an iterative technique to solve Poisson's equation because of the efficiency of the nearest neighbor stencil update on a parallel machine. Since each update is executed in parallel, the operation count is equivalent to the number of iterations required for convergence. In addition, an excellent starting guess for the iteration comes from the converged solution at the previous time step. We use overrelaxation with red–black ordering and Chebyschev acceleration.

In more detail, we wish to solve the equation

$$\nabla^2 \psi = f \tag{IV.1}$$

with the boundary conditions $d\psi/dn$ given on walls. We assume that we are in a geometry such that each section of the boundary lies on grid points and is parallel to either the x or y axis. We assume that $bx_{i,j}$, $by_{i,j}$ are given everywhere, where $bx_{i,j}$ and $by_{i,j}$ are defined by the inward normal $\mathbf{n}$ at boundaries, namely $(bx_{i,j}, by_{i,j}) = \mathbf{n}$. By convention, $\mathbf{n}$ is set equal to $(0, 0)$ in the interior of the flow. We write the five-point Laplacian to approximate Eq. (VII.1), namely,

$$\frac{\psi_{i+1,j} + \psi_{i-1,j} + \psi_{i,j+1} + \psi_{i,j-1} + (-4)\psi_{i,j}}{h^2} = f_{i,j}. \tag{IV.2}$$

Our goal is to formulate this expression for both the interior and the boundary in terms of a linear system of equations

$$a\psi_{i+1,j} + b\psi_{i-1,j} + c\psi_{i,j+1} + d\psi_{i,j-1} + e\psi_{i,j} = g_{i,j}. \tag{IV.3}$$

In the interior, all points in the five-point stencil exist, and we may simply write

$$(1)\psi_{i+1,j} + (1)\psi_{i-1,j} + (1)\psi_{i,j+1} + (1)\psi_{i,j-1} + (-4)\psi_{i,j} = h^2 f_{i,j}. \tag{IV.4}$$

For wall points that are not corners, we use the Neumann boundary conditions to create points outside the domain, since we do not have values for $\psi_{-1,j}$, $\psi_{M+1,j}$, $\psi_{i,-1}$, $\psi_{i,N+1}$. As an example, consider the left wall, that is, where $(bx, by) = (1, 0)$. Let $given_x$ be the normal derivative of ψ in the positive x-direction. Then we may create a virtual point just outside the boundary by noting that

$$\frac{\psi_{1,j} - \psi_{-1,j}}{2h} = given_x;$$

thus we have that

$$\psi_{-1,j} = \psi_{1,j} - (2h)\, given_x.$$

Substituting into Eq. (VII.4) for $\psi_{-1,j}$, we have

$$\frac{\begin{pmatrix}\psi_{i+1,j} + \psi_{i+1,j} - (2h)\, given_x \\ + \psi_{i,j+1} + \psi_{i,j-1} + (-4)\,\psi_{i,j}\end{pmatrix}}{h^2} = f_{i,j}. \quad \text{(IV.5)}$$

We then rewrite Eq. (VII.5) in the standard form as

$$(0)\,\psi_{i+1,j} + (2)\,\psi_{i-1,j} + (1)\,\psi_{i,j+1} + (1)\,\psi_{i,j-1} + (-4)\,\psi_{i,j} = h^2 f_{i,j} + (2h)\, given_x. \quad \text{(IV.6)}$$

Similar expansions hold for the other walls. In the case of outward corners, we create virtual points in both directions using the Neumann boundary conditions. In the case of inward corners, all points of the stencil exist, and we may use the usual five-point Laplacian.

Next, we symmetrize the resulting system by multiplying each equation by the scale factor $sf_{i,j}$ defined by:

$$sf_{i,j} = \begin{cases} 1, & \text{in the interior} \\ \frac{1}{2}, & \text{on walls and inward corners} \\ \frac{1}{4}, & \text{on outward corners.} \end{cases}$$

The linear system $\bar{\mathbf{A}}\mathbf{x} = \bar{\mathbf{b}}$, thus defined by

$$a\psi_{i+1,j} + b\psi_{i-1,j} + c\psi_{i,j+1} + d\psi_{i,j-1} + e\psi_{i,j} = g_{i,j} \quad \text{(IV.7)}$$

where

$$a = (1 + bx_{ij})\, sf_{i,j}, \qquad b = (1 - bx_{i,j})\, sf_{i,j},$$
$$c = (1 + by_{i,j})\, sf_{i,j}, \qquad d = (1 - by_{i,j})\, sf_{i,j},$$
$$e = (-4)\, sf_{i,j},$$
$$g_{i,j} = [(h^2 f_{i,j}) + (2h)(bx_{i,j})\, given_x + (2h)(by_{i,j})(given_y)]\, sf_{i,j},$$

is a symmetric linear system and thus has real eigenvalues. Finally, we note that the Poisson equation subject to the Neumann boundary condition is well-posed only if the Neumann compatibility is satisfied, which is equivalent to the fact the matrix $\mathbf{A}$ is singular, due to the zero eigenvalue corresponding to the eigenvector $\mathbf{e}_1^T = (1, 1, 1, ..., 1)$. If we multiply both sides of the expression $\bar{\mathbf{A}}\mathbf{x} = \bar{\mathbf{b}}$ by this eigenvector, we obtain that $\mathbf{e}_1^T \cdot \bar{\mathbf{b}} = 0$, which is the discrete form of the Neumann compatibility condition. In a discrete approximation using a finite number of grid points, the resulting system may fail to satisfy this condition. In order to make this problem well-posed numerically, we must enforce this requirement. We do so by projecting out of the null-space by defining a new vector $\bar{\bar{\mathbf{b}}} = \bar{\mathbf{b}} - [\mathbf{e}_1^T \cdot \bar{\mathbf{b}}/|\mathbf{e}_1|]\, \mathbf{e}_1^T$. Then the linear system $\bar{\mathbf{A}}\mathbf{x} = \bar{\bar{\mathbf{b}}}$ is a symmetric banded diagonal system satisfying the discrete Neumann compatibility condition.

We solve this system by red-black overrelaxation with Chebyschev acceleration. The relaxation factor is determined by the spectral radius. For a square matrix representing a square geometry, there are explicit formula for the radius. However, in our case we allow arbitrary input of rectilinear geometries as part of the user input. Thus, we determine the relaxation factor ω as follows. Once the initial matrix is built, we perform a set of initialization trials to determine ω. We calculate the number of iterations required for convergence as a function of possible values for ω and use a method of bisection to determine the optimal choice. Since this experiment need be performed only once per input geometry, the faster convergence is well worth the time spent determining a suitable value for ω.

PART THREE: THE PARALLEL IMPLEMENTATION OF VORTEX METHODS

V. Algorithm Flow

In this section, we describe the assembly of the various pieces of the vortex algorithm. After the particular flow geometry has been initialized, imagine that at time $n\,\Delta t$, we have a collection of vortex elements, both within the numerical boundary layer and in the interior. To obtain the new position of vortex elements, we must:

Step 1. Use the N-body solver to compute the vorticity field $\mathbf{u}_{\text{vor}}$ induced by the vortex blobs.

Step 2. Use the potential solver to compute the potential field $\mathbf{u}_{\text{pot}}$.

Step 3. Use the N-body solver to compute the velocity field induced by vortex sheets.

Step 4. Create new vortex sheets to satisfy the no-slip condition.

Step 5. Move all vortex elements.

Step 6. Trade elements across the numerical boundary layer.

Step 7. Eliminate vortex elements that leave the domain.

Step 8. Return to Step 1.

An efficient parallel implementation requires the appropriate processor configuration at each step of the algorithm. Figure 5 shows the algorithm flow together with the processor configuration for each stage of the process. The details of the data flow are given below.

Let δ be the size of the boundary layer. Suppose we are at time step n with the following objects in the domain of the flow:

(1) N_{blobs} vortex blobs located more than a distance δ away from any wall, in whatever positions they end up. Each blob is specified by its x and y coordinates and circulation C_{blob}.

(2) N_{sheets} vortex sheets located within δ of a solid wall. Each sheet is specified by its x and y coordinates and strength ξ, as well as a vector $(bx_{\text{sheet}}, by_{\text{sheet}})$ labeling which wall is closest.

(3) A collection of boundary probe objects $Nprobe_{\text{wall}}$, with circulation $C_{\text{blob}} = 0$, located at the intersection of the Laplace grid and ∂D.

(4) N_{create} stations located at intervals along the solid walls, where vortex sheets are created to satisfy the no-slip condition. The location of each of these objects is fixed at the beginning of the calculation. At the beginning of every time step, each is given strength $\xi = 0$, and vector $(bx_{\text{sheet}}, by_{\text{sheet}})$ according to the type of wall on which it resides.

(5) A collection of $Nprobe_{\text{edge}}$ edge probes, located a distance *edge* in from the solid walls. The location of each of these objects is fixed at the beginning of the calculation. At the beginning of each time step, each is given strength $\xi = 0$, and a vector $(bx_{\text{sheet}}, by_{\text{sheet}})$ according to the type of wall on which it resides.

Figure 6 shows the data elements in the domain. The algorithm moves from time step n to time step $n+1$ as follows:

(1) Pass to the N-body solver the collection of vortices N_{blobs}, as well as the dummy vortices $Nprobe_{\text{wall}}$ and the

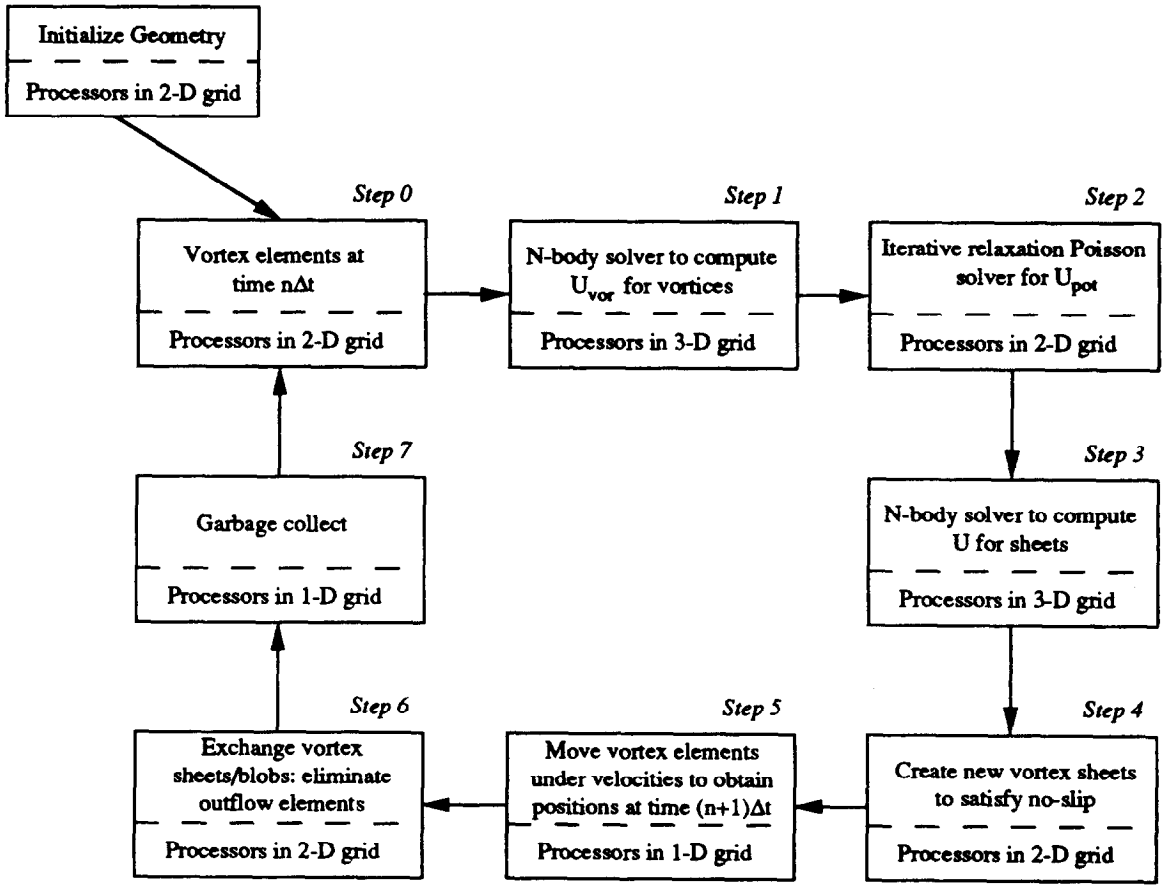

FIG. 5. Complete vortex algorithm and processor communications.

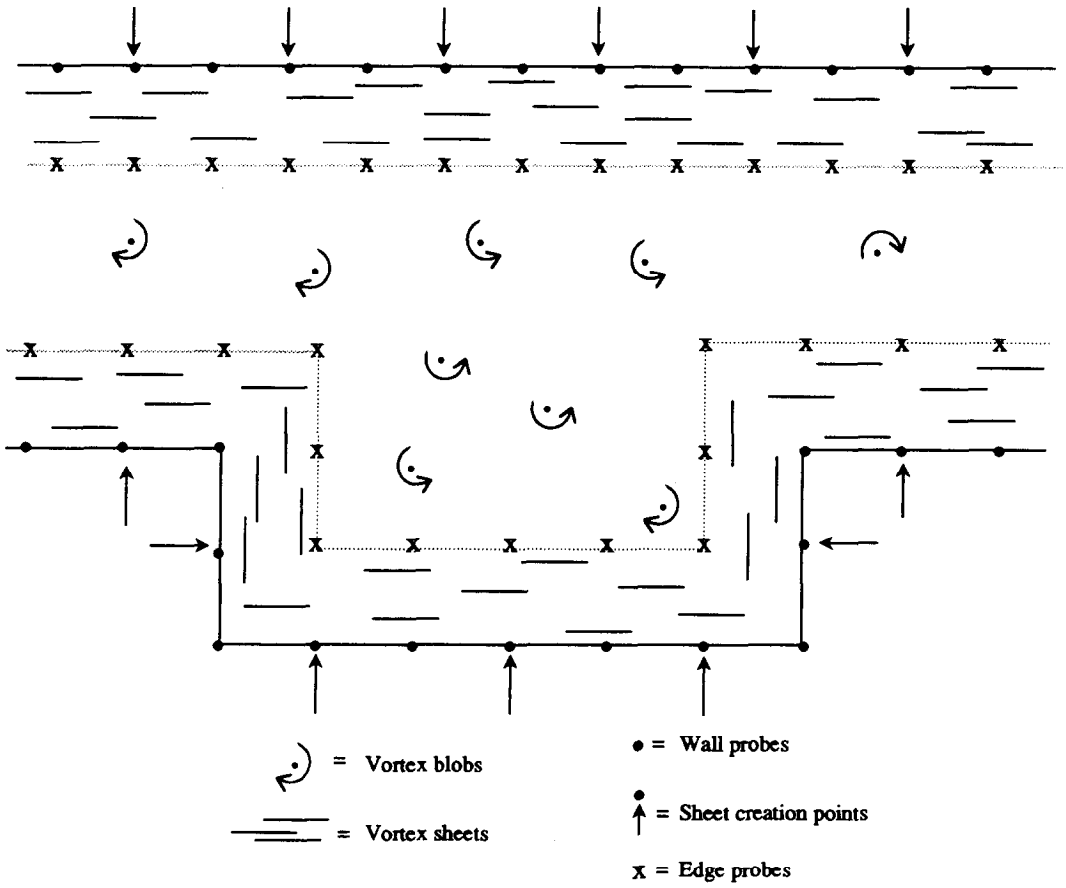

FIG. 6. Data elements probe points.

points $Nprobe_{\text{edge}}$. Using the N-body solver, find the u_{vor} and v_{vor} velocities for each of objects, using the vortex blob force law described below.

(2) Using the normal velocities at the boundary points $Nprobe_{\text{wall}}$, solve by successive overrelaxation for the potential function at all the grid points. Using central differences, compute u_{pot} and v_{pot} at all the grid points. Using linear interpolation, compute velocities at the objects N_{blobs}, $Nprobe_{\text{wall}}$, $Nprobe_{\text{edge}}$.

(3) Give the objects $Nprobe_{\text{edge}}$ strengths equal to their velocity component in the direction tangent to the wall that owns them.

(4) Pass to the N-body solver the points N_{sheet}, $Nprobe_{\text{edge}}$, N_{create}, and compute u and v for each of these, using the vortex sheet force law described below.

(5) Create new real sheets (that is, add to the list of sheets) at the points N_{create} with strength equal to the negative of the velocity component tangential to wall. Set both the u and v velocity of these newly created sheets to zero.

(6) Move all vortex elements with Euler's method according to their u and v velocities, adding a random step. In the case of the sheets, suppress the component of the random step in the direction tangential to the wall.

(7) Convert sheets that have moved into the interior into blobs. Convert blobs that have moved into the boundary layer into sheets, with appropriate flags. Reflect sheets that have gone out of bounds. Eliminate sheets and blobs that have exited through the orifices.

(8) Return to step one.

We assume that the inward-pointing vectors $(bx_{i,j}, by_{i,j})$ are defined everywhere, with the convention that $(bx_{i,j}, by_{i,j})$ is zero in the interior. In addition, we have $flow_x$ and $flow_y$, which are the prescribed inlet/outlet velocities at the boundaries. These are non-zero only where the boundary is open far an inlet/outlet.

Computation of Velocities Induced by Vortices

We pass the elements N_{blobs} and $Nprobe_{\text{walls}}$ to the N-body solver to compute the induced velocities $(u_{\text{blob}\,k}, v_{\text{blob}\,k})$, and velocities $(u_{\text{boundary}\,k}, v_{\text{boundary}\,k})$. Here,

we pass all the elements at once. The force law between the elements is

$$(u_k(x_k, y_k), v_k(x_k, y_k)) = \sum_{l = \text{all blobs and probe stations}} \left(\frac{C_l}{\left(2\pi[\max((x_l - x_k)^2 + (y_l - y_k)^2), \sigma^2]\right)} \right) \times ((y_l - y_k), -(x_l - x_k)). \quad \text{(V.1)}$$

Here, we have dropped the subscripts *blobs* and *boundary* to indicate that we do not distinguish between the two in the N-body force law calculation. On exiting from the N-body solver, we have computed the velocity induced by all the vortices at all vortex blobs and boundary points.

Addition of Potential Flow

Given a region D, we require that $\mathbf{u} \cdot \mathbf{n} = 0$ on the boundary ∂D, where $\mathbf{n}$ is the inward normal vector. If the domain D is not closed but has inlets and outlets, we must simply add the inflow/outflow boundary conditions at these orifices to the negative of the vortex induced flow. We pass to the Poisson solver the problem $\nabla^2 \phi = 0$ subject to the Neumann boundary conditions

$$\frac{d\phi_{\text{boundary}_l}}{d\mathbf{n}_{\text{boundary}_k}} = (flow - value - u_{\text{boundary}_k}, flow - value - v_{\text{boundary}_k}) \cdot (|bx_{\text{boundary}_k}|, |by_{\text{boundary}_k}|). \quad \text{(V.2)}$$

Here, we employ a slight abuse of notation, namely that $\mathbf{n}_{\text{boundary}_k}$ is taken as the normal pointing in only non-negative directions, not the inward pointing normal. This allows us to plug directly into the $given_x$ and $given_y$ notion of the Poisson solver described earlier. In addition, we have used the subscript $_{\text{boundary}_k}$ to indicate boundary points. Finally, *flow* – value is zero at solid walls and the prescribed profile at the given orifice. There is a mapping from the one-dimensional description of the list of boundary points to the two-dimensional Laplace geometry which gives the two-dimensional grid coordinates (i, j).

On output, the Poisson solver will return the value of the potential function ϕ. We calculate the potential velocities on the grid, namely $(u_{\text{pot-grid}_{ij}}, v_{\text{pot-grid}_{ij}})$ by central differences, with one-sided differences on the boundary points. Once again, we build coefficient masks to handle the boundary conditions in parallel, similar to those used in the linear system obtained from the Laplace operator. Finally, we must interpolate this potential velocity back to the vortices. Given a vortex blob located at $(x_{\text{blob}_k}, y_{\text{blob}_k})$, let $(u_{\text{interp}_k}, v_{\text{interp}_k})$ be the potential velocity at $(x_{\text{blob}_k}, y_{\text{blob}_k})$, obtained through bilinear interpolation among the four neighboring cell velocities. Then the total velocity for each vortex k is thus given by

$$(u_{\text{blob total}_k}, v_{\text{blob total}_k}) = (u_{\text{blob}_k}, v_{\text{blob}_k}) + (u_{\text{interp}_k}, v_{\text{interp}_k}). \quad \text{(4.3)}$$

This gives the velocity components for each vortex blob.

Boundary Layer

Our strategy in the boundary is to send all the elements to the N-body solver at once. A single force law is devised so that sheets lying on different walls do not interact. Imagine that at time step n we have N_{sheets} sheets located in the boundary layer. Each of these sheets $1 \leqslant k \leqslant N_{\text{sheets}}$ is described by a position $(x_{\text{sheet}_k}, y_{\text{sheet}_k})$, a strength ξ_{sheet_k}, and an orientation vector $(bx_{\text{sheet}_k}, by_{\text{sheet}_k})$. For the purposes of this discussion, we label sheets in the boundary layer as red sheets. (At the first step, there are no sheets in the boundary layer. The construction below will create sheets.)

We begin by creating two additional types of sheet:

(a) *Blue sheets.* These are sheets that represent the flow at infinity as seen from the boundary layer. We create a set of stations parallel to the walls and in a distance *edge*. (In the notation of Section IV.B, these are the points $Nprobe_{\text{edge}}$.) At each of these points $(x_{\text{probe}_k}, y_{\text{probe}_k})$, $1 \leqslant k \leqslant Nprobe_{\text{edge}}$, we create a vortex sheet, assigned the color blue, with strength given by the tangential velocity computed from the velocity induced by the vortex blobs plus the potential flow velocity. The sheets are given the orientation vector $(bx_{\text{sheet}_k}, by_{\text{sheet}_k})$ corresponding to the wall which owns them. Their contribution in the force law represents the velocity as seen at infinity.

(b) *Black sheets.* These are sheets that collect the total tangential flow along the boundary. We create a set of stations along the walls at points $(x_{\text{wall}_k}, y_{\text{wall}_k})$. (In the notation of Section IV.B, these are the points $Nprobe_{\text{wall}}$.) At each of these points, we create a vortex sheet, assigned the color black, with zero strength. The sheets are given the orientation vector $(bx_{\text{sheet}_k}, by_{\text{sheet}_k})$ corresponding to the wall which owns them. These sheets will enter into the force law calculation and exit that calculation with the amount of vorticity that must be created in order to satisfy the no-slip condition.

Thus, as input to the boundary layer algorithm, we have a collection of vortex sheets characterized by positions $(x_{\text{sheet}_k}, y_{\text{sheet}_k})$, strengths ξ_{sheet_k}, and orientation vectors $(bx_{\text{sheet}_k}, by_{\text{sheet}_k})$. (The colors of the various sheets are not relevant during the pass to the N-body solver).

Pass to N*-Body Solver*

Our goal is to determine the velocity at each of these sheets induced by all others. That is, we wish to find the

velocity $(u_{\text{sheet}_k}, v_{\text{sheet}_k})$ for each sheet. In order to do so, we note that the sheets have different orientations. The definition of normal (perpendicular to the sheet segment) and tangential (along the sheet segment) motion depend on the sheet. In order to use the same force law, we must transform into a uniform coordinate system. This may be done by means of the orientation vectors $(bx_{\text{sheet}_k}, by_{\text{sheet}_k})$. The height N_{sheet_k} in the normal direction (away from the wall) is given by

$$\begin{aligned} T_{\text{sheet}_k} &= (x_{\text{sheet}_k}, y_{\text{sheet}_k}) \cdot \hat{t}_{\text{sheet}_k}, \\ N_{\text{sheet}_k} &= (x_{\text{sheet}_k}, y_{\text{sheet}_k}) \cdot \hat{n}_{\text{sheet}_k}, \end{aligned} \tag{V.4}$$

where the tangential vector $\hat{t}_{\text{sheet}_k}$ and the normal vector $\hat{n}_{\text{sheet}_k}$ are given by

$$\begin{aligned} \hat{t}_{\text{sheet}_k} &= (-by_{\text{sheet}_k}, bx_{\text{sheet}_k}), \\ \hat{n}_{\text{sheet}_k} &= (bx_{\text{sheet}_k}, by_{\text{sheet}_k}). \end{aligned} \tag{V.5}$$

The normal and tangential velocities, that is, $(Utan_{\text{sheet}_k}, Unor_{\text{sheet}_k})$, may be determined from the force law according to Eqs. (1.4), (1.5) as

A. *The tangential velocity.* $U_{\text{tan:}} = \sum_{j=1}^{N} \xi_{\text{sheet}_j} [A][B][C][D]$, where

$$A = \left[\max\left(0, 1 - \frac{|T_{\text{sheet}_k} - T_{\text{sheet}_j}|}{h}\right)\right],$$

$$B = \left[\frac{1}{2}(1 + \text{sign}(N_{\text{sheet}_j} - N_{\text{sheet}_k}))\right]$$

$$C = [|(bx_{\text{sheet}_k}, by_{\text{sheet}_k}) \cdot (bx_{\text{sheet}_j}, by_{\text{sheet}_j})|] \times \left[\frac{1}{2}(1 + \text{sign}(bx_{\text{sheet}_j} - bx_{\text{sheet}_k}))\right] \times \left[\frac{1}{2}(1 + \text{sign}(by_{\text{sheet}_j} - by_{\text{sheet}_k}))\right]$$

$$D = \left[\frac{1}{2}(1 - \text{sign}(N_{\text{sheet}_j} - N_{\text{sheet}_k} - 2edge))\right].$$

This expression represents the tangential velocity as given in Eq. (III.1). While somewhat complicated, this expression contains all the interactions:

(a) The first term $[A]$ linearly interpolates the effect of the vorticity of sheet j on sheet k, with the effect ranging from 0 (none of sheet j covers sheet k) to 1 (all of sheet j covers sheet k).

(b) The second term $[B]$ allows only those sheets above sheet k to contribute to the velocity of sheet k.

(c) The third term $[C]$ makes sure that sheets corresponding to different walls do not interact.

(d) The fourth term $[D]$ makes sure that those sheets located more than twice the boundary layer size away from sheet k in the normal direction do not contribute. This is necessary so that in complicated geometries the sheets on two different walls with the same orientation vector do not interact.

B. *The normal velocity.* $U_{\text{nor:}} = \sum_{j=1}^{N} \xi_{\text{sheet}_j} [A][B][C][D]$, where here

$$A = [\min(N_{\text{sheet}_j}, N_{\text{sheet}_k})]$$

$$B = (-1/h)\left[\max\left(0, 1 - \frac{|T_{\text{sheet}_k} + h/2 - T_{\text{sheet}_j}|}{h}\right) - \max\left(0, 1 - \frac{|T_{\text{sheet}_k} - h/2 - T_{\text{sheet}_j}|}{h}\right)\right]$$

$$C = [|(bx_{\text{sheet}_k}, by_{\text{sheet}_k}) \cdot (bx_{\text{sheet}_j}, by_{\text{sheet}_j})|] \times \left[\frac{1}{2}(1 + \text{sgn}(bx_{\text{sheet}_j} - bx_{\text{sheet}_k}))\right] \times \left[\frac{1}{2}(1 + \text{sgn}(by_{\text{sheet}_j} - by_{\text{sheet}_k}))\right]$$

$$D = \left[\frac{1}{2}(1 - \text{sgn}(N_{\text{sheet}_j} - N_{\text{sheet}_k} - 2edge))\right]$$

The expression represents the vortex interaction given in Eq. (I.5):

(a) The first term $[A]$ places an upper bound on the sheet interaction: only

(b) The second term $[B]$ uses the tangential velocity in a centered difference approximation to the derivative around the point $(T_{\text{sheet}_k}, N_{\text{sheet}_k})$.

(c) The third term $[C]$ makes sure that sheets corresponding to different walls do not interact.

(d) The fourth term $[D]$ makes sure that those sheets located more than twice the boundary layer size away from sheet k in the normal direction do not contribute. This is necessary so that, in complicated geometries, sheets on two different walls with the same orientation vector do not interact.

We compute the above velocities above by passing to the N-body solver.

Creation of Vortex Sheets at Boundary

Given the above, we calculate the actual velocities of the vortex sheets in the x–y coordinate system of the two-dimensional grid by transforming back, that is,

$$\begin{aligned} u_{\text{sheet}_k} &= (Utan_{\text{sheet}_k}, Unor_{\text{sheet}_k}) \cdot \hat{t}_{\text{sheet}_k}, \\ v_{\text{sheet}_k} &= (Utan_{\text{sheet}_k}, Unor_{\text{sheet}_k}) \cdot \hat{n}_{\text{sheet}_k}. \end{aligned}$$

Finally, we must create vortex sheets to satisfy the no-slip condition. The velocities of the red sheets are as given above. The velocities of the black sheets are ignored. At the same time, at each of the stations $Nprobe_{\text{wall}}$, where there is a black sheet, we create a red sheet with the same position, same orientation vector, and same strength given by $-Utan_{\text{sheet}_k}$, where $Utan_{\text{sheet}_k}$ is as found above. In practice, we break each newly created red sheet into a user-specified number of red sheets.

Updating Vortex Positions

We now have velocities for each of the vortex elements. That is, for blobs $k = 1, N_{\text{blobs}}$, we have positions $(x_{\text{blob}\,k}, y_{\text{blob}\,k})$ and velocities $(u_{\text{blob}\,k}, v_{\text{blob}\,k})$. For sheets $k = 1, N_{\text{sheets}}$, we have positions $(x_{\text{sheet}\,k}, y_{\text{sheet}\,k})$ and velocities $(u_{\text{sheet}\,k}, v_{\text{sheet}\,k})$, as well as orientation vectors $(bx_{\text{sheet}_k}, by_{\text{sheet}_k})$. In addition to advection of the vortex element according to the velocity field, a random step is superimposed on the motion (see Section II). This diffusion step is a random variable drawn from a Gaussian distribution with mean zero and variance $2\Delta t/R$, where, again, R is the Reynolds number and Δt is the time step. In the boundary layer, the random step is taken only in the direction normal to the wall. Thus, we update the positions of the vortex elements according to

$$(x^{n+1}_{\text{blob}\,k}, y^{n+1}_{\text{blob}\,k}) = (x^n_{\text{blob}\,k}, y^n_{\text{blob}\,k}) + \Delta t(u^n_{\text{blob}\,k}, v^n_{\text{blob}\,k}) + (\eta_1, \eta_2) \tag{V.8}$$

$$(x^{n+1}_{\text{sheet}\,k}, y^{n+1}_{\text{sheet}\,k}) = (x^n_{\text{sheet}\,k}, y^n_{\text{sheet}\,k}) + \Delta t(u^n_{\text{sheet}\,k}, v^n_{\text{sheet}\,k}) + (bx_{\text{sheet}}\eta_1, by_{\text{sheet}}\eta_2), \tag{V.9}$$

where (η_1, η_2) are random variables each drawn from a Gaussian distribution with mean zero and variance $2\Delta t/R$.

Exchange Sheets and Blobs, Sheet Reflection, and Removing Blobs Outside the Domain

After the vortex elements (sheets + blobs) have moved, their positions are checked:

(1) If any vortex blobs left the domain, eliminate them.

(2) If any vortex sheets left the domain, reflect them back in.

(3) If any vortex blobs entered the boundary layer, transform them to sheets.

(4) If any vortex sheets left the boundary layer, transform to sheets.

In order to determine which of the above choices is appropriate, we must determine the location of the vortex element after the update. This may be done *in parallel* by using the signed distances described in Section V.B. As described there, the location is determined by using the signed distance variable for the processor associated with the cell containing the vortex element. The distance is then used to determine which of the above tests should be executed. The complete specification of the signed distance flag and the parallel execution of the above tests is described in excruciating detail in [34]. Finally, we perform garbage collection to remove the holes in the arrays.

VI. Display

To display the results, we use a real-time visualization environment [32] to analyze the results of the computation. Starting from a precomputed discrete set of time-dependent flow quantities, such as velocity and density, this environment allows the user to interactively examine the data on a framebuffer using animated flow visualization diagnostics, such as tracer particles and dye injection, that mimic those in the experimental laboratory. As input, we perform a stream function vortex-in-cell calculation to obtain the velocity field on a grid. On an 8,192 processor CM-2, images are updated on the framebuffer at nine frames per second, producing essentially real-time motion and an effective way to study the solution, analyze fluid flow mechanisms, and compare numerical results with experiment. For details, see [32].

VII. Timings

In this section we analyze the efficiency of the two main kernels of the code, the N-body solver and the iterative Poisson solver.

In order to give accurate timings for the N-body kernel of the vortex code, we extracted that portion of the code and ran simulations of the interaction of point vortices in two dimensions. This code was written using a test kernel written in *LISP. The runs were made on two CM-2 configurations, 4K and 8K, with varying numbers of particles. We list the timings for the replicated orrery algorithm in Table I. P is the number of processors, N is the number of bodies, and M is the number of times the orrery was replicated. T is the

TABLE I

Timings for N-body Kernel

P	N	M	T	T_{comm}
8K	2048	4	0.33	0.1
	4096	2	1.30	0.37
	8192	1	5.1	1.48
4K	2048	2	0.63	0.19
	4096	1	2.5	0.76

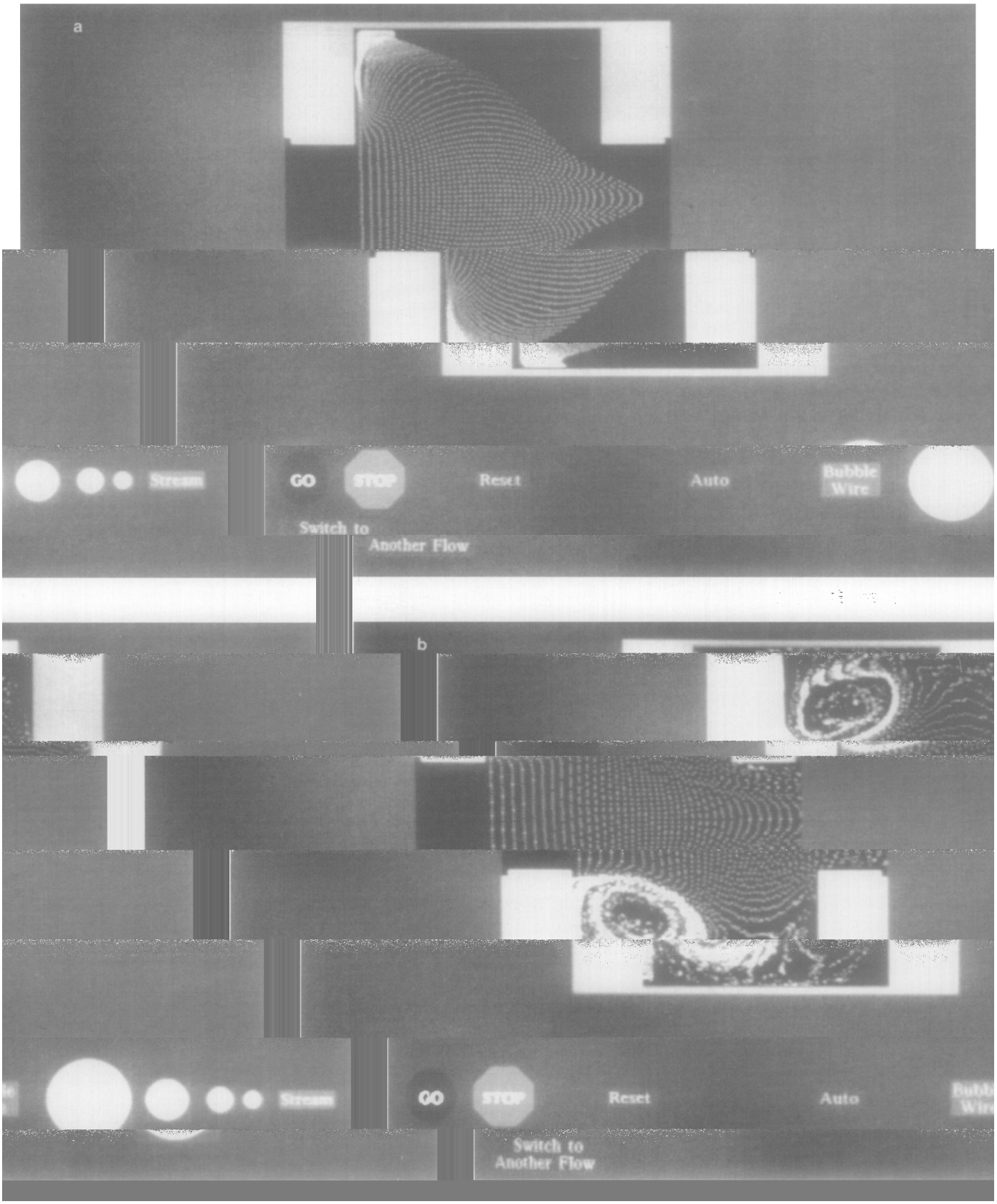

FIG. 7. Expanding/contracting nozzle.

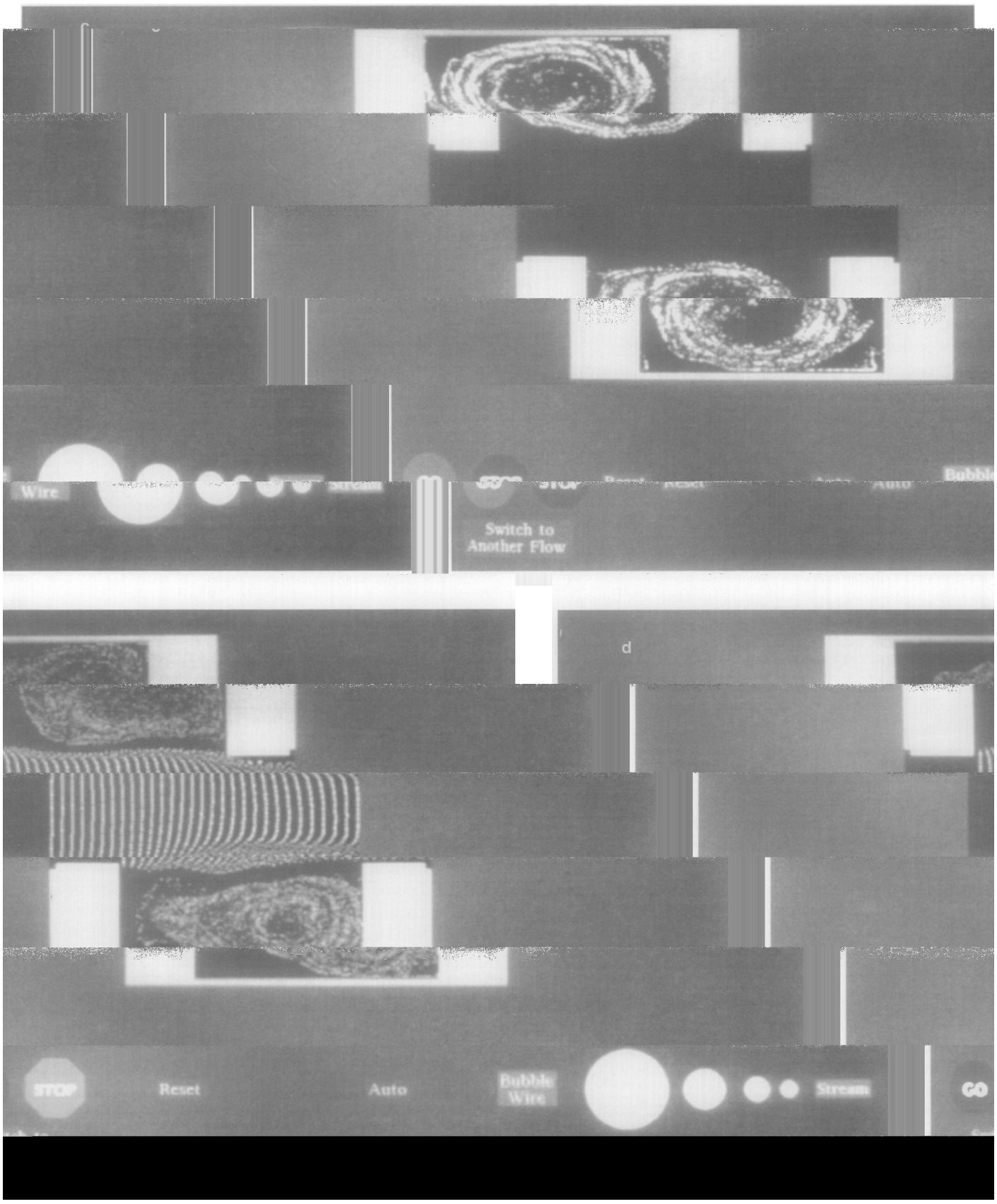

FIGURE 7—*Continued*

total time, in seconds, to compute all N^2 interactions, and T_{comm} is just the time spent in communications.

Note that in all cases, the communications account for about 30% of the total running time for the algorithm. As we described previously, the complexity of the replicated orrery algorithm is $O(N/M) + O(\log M)$. In the cases run above, the $O(N/M)$ term dominates. Note that, in fact, we see the correct behavior as N increases for a fixed number of processors. For example, for $P = 8K$, the times for $N = 2048, 4096, 8192$ should vary as 1, 4, 16, which they do.

A factor of 2–5 improvement in performance for the same configuration and a similar number of bodies can be obtained by using the multi-wire orrery described in Appendix B. Using this approach, as the number of bodies grows, for a fixed number of processors, the percentage of time spent in communications decreases to less than 1%. We are in the process of implementing this improved kernel in our general code.

Next, we analyze the iterative Poisson solvers. After the two-dimensional *LISP implementation was complete, we began a CM Fortran version of the code, which will take advantage of the slicewise compiler. At the time this article went to press, the elliptic Fortran kernels were complete. All of the timings below for the iterative Poisson solver refer to this elliptic solver. We give timings for three different iterative solvers: (1) SOR, (2) conjugate gradient, and (3) preconditioned conjugate gradient to solve the Poisson equation. Timings are given as the time per iteration. We studied problems on a 256×256 mesh of points. Results were executed on an 8K machine, giving a virtual processor ratio of 8 for the 256×256 mesh. We study the timings for the SOR algorithm, as well as a conjugate gradient algorithm and a preconditioned conjugate gradient technique implemented in our vortex code. SOR required 0.006783 per iteration; conjugate gradient required 0.008653 s, and a preconditioned conjugate gradient required 0.009072 s. On a full machine (64K), these timings should be divided by a factor of 8. Convergence was determined by variation in the max norm of the residual over one iteration. With a convergence criteria on 10^{-4}, to compute the potential flow for a square with left to right flow, SOR required 1132 iterations to converge, conjugate gradient (CG) required 335, and preconditioned conjugate gradient (PCG) required 187. With a convergence criterion of 10^{-5}, SOR required 1795 iterations, CG required 400 iterations, and PCG required 363 iterations. On a more complex geometry (the valve described later), 10^{-4} convergence required 3559 SOR steps, 919 CG steps, and 909 PCG steps.

PART FOUR: RESULTS

Expanding/Contracting Nozzle

We begin with flow in an expanding and contracting nozzle. Flow enters from the left, and encounters a sudden expansion. After some distance, the channel suddenly contracts, and flow exits on the right.

We took a uniform entrance and exit profile of unit speed, and the domain is inscribed in a box of side length unity. The Reynolds number is 5000, and the time step is $\Delta t = 0.025$. We chose a minimum sheet strength of 0.025 and solved the Poisson equation on a 128×128 grid.

The flow portrait is as follows: As $t = 0$, the no-slip condition is instantaneously imposed, revealing the characteristics of startup. The flow entering from the left is visualized using a hydrogen bubble wire (Fig. 7a), stretched vertically across the entrance. As the flow enters the expansion section, the development of large, counterrotating fluid structures in the upper and lower corners pull the flow around and back into the "arms" of the domain (Fig. 7b). At later times, (Fig. 7c), two large, counterrotating eddies occupy the arms, effectively shunting off these sections of the domain and turning the domain into a constant width channel. Note that the flow itself has changed the effective geometry of the domain. The majority of the flow now entering from the left cannot tell the difference between this "blocked" flow and flow in a straight channel. There is little mixing between the central channel and the arms when compared to initial times. Finally, at still later times, (Fig. 7d), the counterroting eddy in each arm has moved to the right and is joined by an opposite moving eddy trapped in the upper and lower left corners. This final configuration remains essentially fixed for the rest of the calculation.

Flow around an Island

Next, we compute flow in a fairly complicated domain with an interior island (Fig. 8). Flow enters through a narrow slit on the left with a uniform profile and immediately encounters an island. Around that island on the upper right is a narrow exit, past a large, vertical cavity.

We took a uniform entrance and exit profile of unit speed, and the domain is inscribed in a box of side length unity. The Reynolds number is 5000, and the time step is $\Delta t = 0.025$. We chose a minimum sheet strength of 0.07 and solved the Poisson equation on a 256×256 grid. The value of the *sheet-factor* was 2, and we used a tolerance criterion of 10^{-2} for both the potential solver and the stream function solver. After 360 time steps, there were 14,000 vortices and 5000 sheets.

In Fig. 8, we placed a hydrogen bubble wire across the left entrance. The flow is split around the island and passes through the middle constriction, creating counterrotating eddies in all the corners. Above the island is a small indentation, which produces a small driven cavity problem, revealing a counterrotating eddy structure. The large cavity remains fairly quiet with minimal acitivity. We suspect that this is the result of underresolution; typical wall velocities are small relative to the minimum sheet strength, and thus few sheets are triggered.

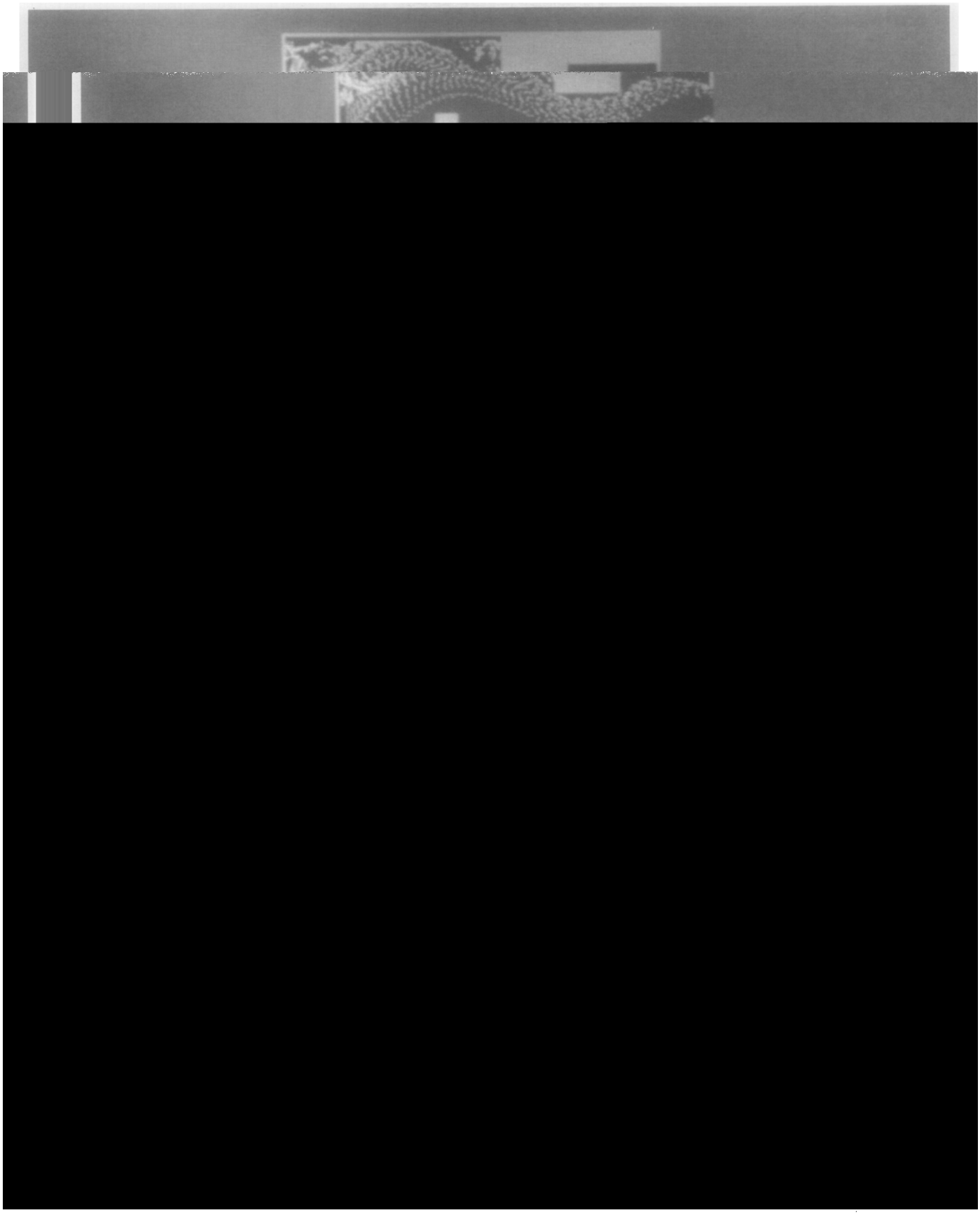

FIG. 9. Flow through CM-2.

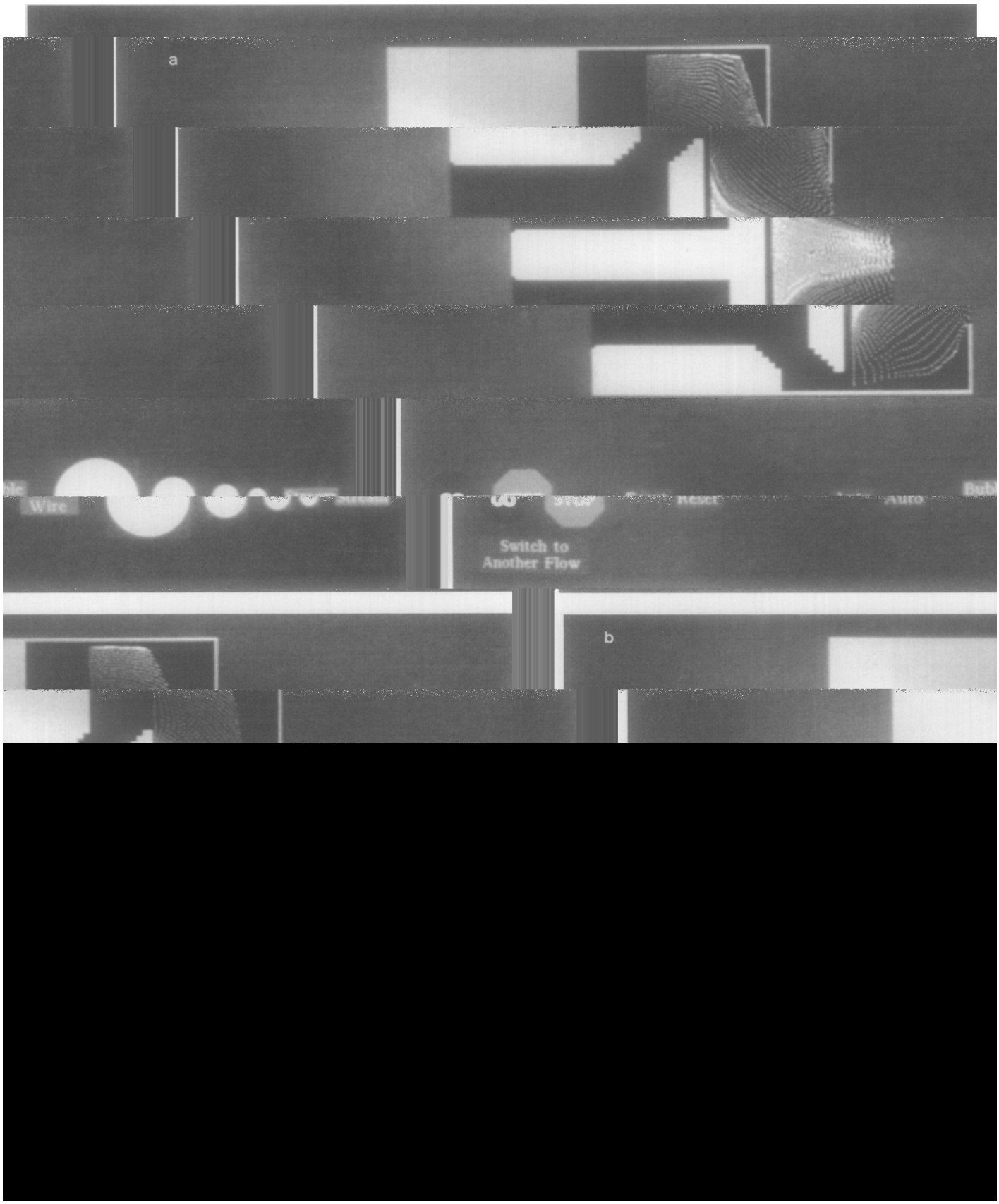

FIG. 10. Flow through piston/valve.

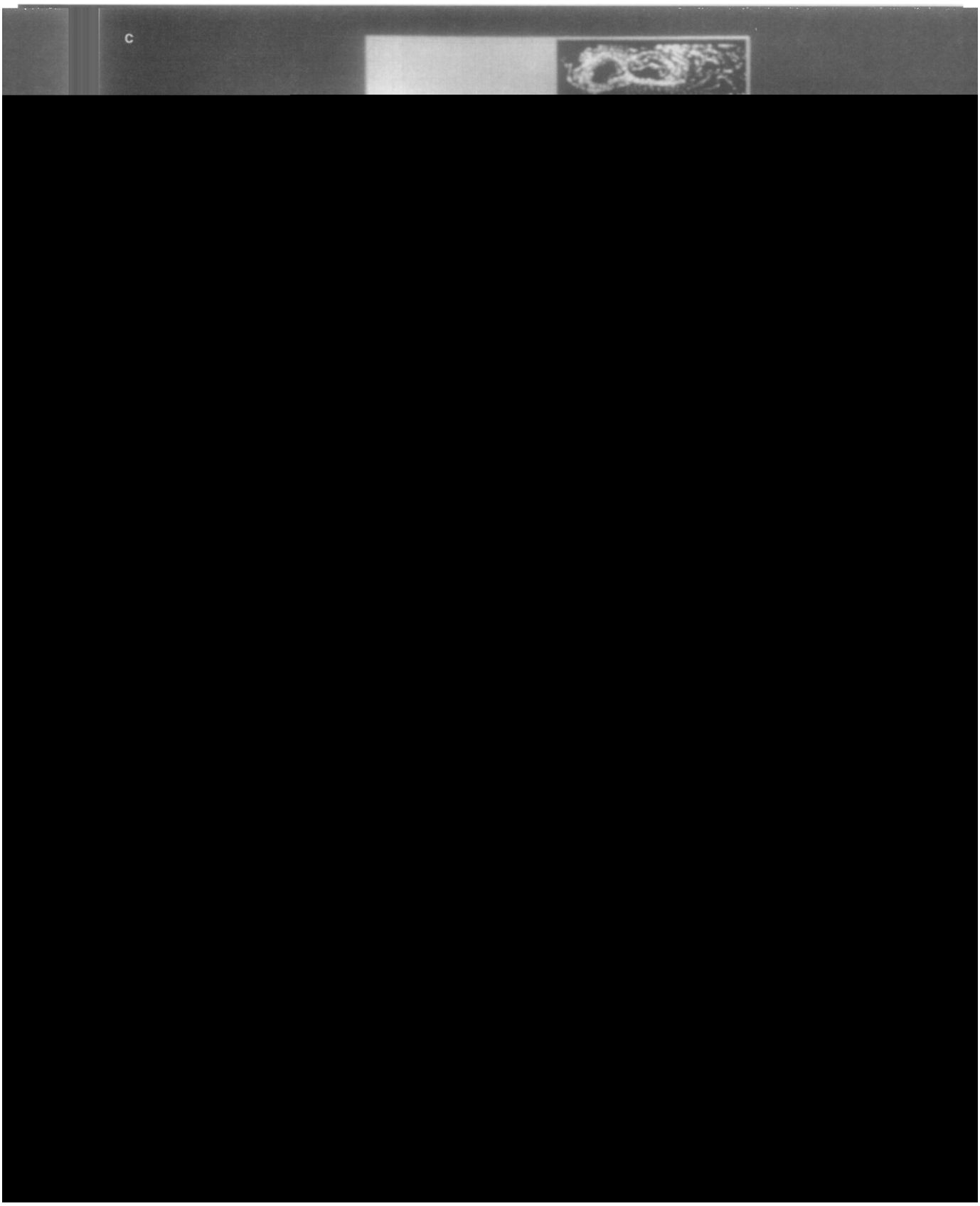

FIGURE 10—*Continued*

Flow through the Connection Machine CM2-A

Next, we model flow through a two-dimensional cross section of the Connection Machine CM2-A (see Fig. 9). Air enters at the bottom left and right and moves horizontally towards the "circuit boards" (the vertical islands in the domain). After flowing past the boards, the pathway splits and moves down the outer walls, exiting on either side.

We took a uniform entrance and exit profile of unit speed, and the domain is inscribed in a box of side length unity. The Reynolds number is 5000, and the time step is $\Delta t = 0.025$. We chose a minimum sheet strength of 0.8 and solved the Poisson equation on a 256×256 grid. The value of the *sheet-factor* was 2, and we used a tolerance criterion of 10^{-5} for both the potential solver and the stream function solver. After 300 time steps, there were 8000 vortices and 10,000 sheets.

The simulation of this flow shows that the two streams entering along the bottom from the left and right meet below the circuit boards and create a pair of counterrotating eddies which turbulize the flow. This turbulized flow enters the narrow channels of the circuit boards, with an oscillatory profile, exits the top, and forms a pair of large, stagnant vortices which effectively narrow the exit orifice. Past these eddies, the flow travels down the wall, creating counterrotating vortices in each of the corners before exiting out the sides.

Flow around a Piston/Valve

Finally, we modeled flow around a fixed valve in a piston chamber. Flow enters from the left along two inlet streams and encounters a quarter-open valve (Fig. 10). On either side of the valve opening, the flow encounters a large chamber with two vertical exits cut on the rightmost wall. We took a uniform entrance and exit profile of unit speed, and the domain is inscribed in a box of side length unity. The Reynolds numbers is 5000, and the time step is $\Delta t = 0.025$.

After the no-slip condition is instantaneously imposed at $t = 0$, entering flow curls around the valves on either side and exits through the slits (Fig. 10a). As the calculation progresses, the flow rolls up behind the valves into two large eddies which trap considerable fluid and remain as fixed, rotating structures behind the valve (Fig. 10b, c). This pair of eddies serves to narrow the passage around the valve, inducing the growth of counterrotating eddies in the top and bottom left corners of the main chamber, which further restricts the flow around the valve. These coherent fluid structures cause further mixing of the internal flow (Fig. 10d).

APPENDIX A: THE CONNECTION MACHINE ARCHITECTURE

Here, we briefly describe the basic design of the Connection Machine CM-2. For details, see [15]. The Connection Machine CM-2 is composed of a sequencer and a maximum of 65,536 single-bit processing elements. The processors run in SIMD (single instruction multiple data) mode, with the instruction stream broadcast by the sequencer. It is possible to deselect any subset of the processors, so that an instruction is only performed by those processors in the currently selected set. The sequencer is controlled by an external front end machine, usually a SUN™, SYMBOLICS™ Lisp Machine, or VAX™.

Each processor has 64K or 256K bits of local RAM, with a single high-speed floating point unit for every 32 processors. There are 16 processors on a CM-2 chip and the chips are connected in a boolean n-cube topology, e.g., a 12-cube for a 64K processor machine. The system software supports the notion of **virtual processors**. This allows the programmer to implement code with the number of processors appropriate for the application. Virtual processors are mapped to physical processors by evenly segmenting the memory of the physical processors and time multiplexing the physical processors. The **virtual processor ratio** is the number of virtual processors assigned to each physical processor by the mapping. In the N-body model, each body is assigned to a virtual processor reponsible for computing the accumulated forces on the body.

The CM-2 supports two basic communication mechanisms. There is general pointer-based communication by which data can be exchanged between the memories of different processors as necessary to complete any computation. We refer to this as a **send**. For more structured communication patterns, the machine can be efficiently configured as a k-dimensional grid; these grids are automatically superimposed by the system software onto the boolean cube using a multi-dimensional Gray code mapping. These communications patterns are periodic in each dimension of the grid. Motion of the data from all processors to their nearest grid neighbors is known as a NEWS communication. A particularly useful primitive available on the Connection Machine computer are **scans**, which combine computations and communications. In logarithmic time, these operators allow one to **spread** data through the CM-2, as well as accumulate summands (**plus-scan**) from each processor.

There are two programming models for the machine. Calculations on the machine are performed in two ways. In the standard **fieldwise** model, the storage of a 32-bit word would be allocated in 32 consecutive bits of a physical processor's memory. However, when performing floating point computations, the data must be transposed before being loaded into floating point hardware, thus better performance is usually obtained by viewing the processors in a **slicewise** configuration. That is, we consider processing nodes on the machine to be the ensemble of a floating point unit and the memory of the 32 associated physical processors of the CM-2. In this approach, a word is stored

in a 32-bit slice across the memories of the 32 processors in the node, i.e., one bit per processor. From this viewpoint, a 64K processor CM-2 becomes 2048 floating point nodes, connected as an 11-dimensional hypercube with two communication channels between connected nodes instead of

with the way in which the floating point units actually access data from their associated processors, i.e., in one cycle a 32-bit slice across the processors is read into the floating point unit. When the original version of this code was implemented, the slicewise model was not available from high level software.

Finally, a 1280×1024 frame buffer with parallel I/O is directly connected to the memory of a Connection Machine.

APPENDIX B: FURTHER WORK

Between when this paper was first submitted and the time of its acceptance for publication, we began a new implementation of the code, taking advantage of the slicewise compiler which became accessible from high level Fortran. This has led to the work on multi-wire N-body solvers and conjugate gradient techniques discussed below.

Multi-Wire N-Body Solvers

An alternate N-body solver results from assigning several bodies to each floating point node and performing the orrery rotation by introducing a multi-wire all-to-all broadcast which makes optimal use of the communication bandwidth of the hypercube using "rotated and translated Gray codes" described in [22] and summarized below. As described in Appendix A, using the slicewise programming model, a 65,536 processor CM-2 becomes a 2048 node hypercube ($d = 11$) with 22 communication channels per node. A typical vortex calculation for fluid within complex geometries generates many thousands of vortex elements and thus, in this case, the replicated orrery algorithm is no longer appropriate. Instead, we map N/P bodies to each node. There are two copies of the data: a static copy responsible for accumulating the forces for those bodies and a dynamic copy which will circulate through every other node of the hypercube so that all N^2 interactions are performed. The motivation for the algorithm is to have the dynamic copy circulate in such a way that we use all of the hypercube wires at each communications step.

The communication pattern is constructed by finding as many conflict-free paths as possible leaving a node, visiting all others, and returning to its starting point. By translating the starting point of each of these paths to every node of the hypercube, the data in each node circulates to every other one in as short a time as possible.

More precisely, given a d-dimensional cube, a Gray code yields a Hamiltonian cycle through the cube, i.e., a path that visits every node of the cube once and returns to its origin node. It takes $2^d - 1$ steps for any such path to traverse every node in the cube. From this path we can generate $d - 1$ additional paths by rotating through each dimension of the hypercube. We can take these d paths and translate

node circulates to every other one. The resulting paths are timewise edge independent; that is, at each step none of the $d \cdot 2^d$ Hamiltonian cycles traverse the same edge in the same direction. We allow edges to be traversed in different directions on the same step because the wires of the CM-2 can be assumed to be bidirectional.

After each edge of the path is traversed, we perform those interactions that depend on the data available in each computational node. This eliminates the need to store the circulating data. The computational complexity of this approach is $O(N^2/P)$ in arithmetic and $O(\text{ceiling}(N/2^d \cdot 1/2d) \cdot (2^d - 1))$ in communications. The latter estimate derives from the number of steps it takes to send N/P data items out of a node on $2d$ wires times the length of the communications path. Further details of the multiwire N-body solver based on a slicewise model may be found in [11].

Conjugate Gradient Methods for the Elliptic Solvers

The iterative relaxation solver for the Poisson equation in original code can be replaced by a conjugate gradient method. In later versions of the code we have done so, and examined the potential benefits in both conjugate gradient and preconditioned conjugate gradient. As discussed in the section on timings, changing to a gradient method substantially reduced the iteration count. However, beyond diagonal preconditioning, we were unable to produce an effective preconditioner that works effectively in arbitrary geometries.

ACKNOWLEDGMENT

We wish to thank Bruce Boghosian for his considerable insight and advice during the design of this code, Alan Edelman and Washington Taylor for their contributions to the N-body solvers, James Salem for his work on the visualization package, and Carmen Crocker for her help in preparing the figures. The vortex code described within was written in *Lisp, a parallel extension of the programming language Lisp. All calculations were performed at Thinking Machines Corporation on the Connection Machine CM-2.

REFERENCES

1. C. R. Anderson, *SIAM J. Numer. Anal.* **23**, 413 (1986).
2. C. Anderson and C. Greengard, *SIAM J. Numer. Anal.* **22**, 413 (1985).
3. A. W. Appel, *SIAM J. Sci. Stat. Comput.* **6**, 85 (1985).
4. J. F. Applegate, M. R. Douglas, Y. Gursel, P. Hunter, C. Seitz, and G. J. Sussman, *IEEE Trans. Comput.* **84**, 822 (1985).
5. S. B. Baden and E. G. Puckett, *J. Comput. Phys.* **91** (1990).

6. J. Barnes, "Multiple-Timestep N-body Algorithms for Parallel Computers," Institute for Advanced Study, Princeton, (unpublished).
7. J. Barnes and P. Hut, *Nature* **324**, 446 (1986).
7. J. T. Beale and A. Majda, *J. Comput. Phys.* **58**, 188 (1985).
9. J. T. Beale and A. Majda, *Math. Comput.* **39**, 1 (1982).
10. J. T. Beale and A. Majda, *Math. Comput.* **39**, 29 (1982).
11. J-Ph. Brunet, A. Edelman, and J. P. Mesirov, Hypercube algorithms for parallel direct N-body solvers, in preparation.
12. A. J. Chorin, *J. Fluid Mech.* **57**, 785 (1973).
13. A. J. Chorin, *J. Comput. Phys.* **27**, 428 (1978).
14. A. J. Chorin, *SIAM J. Sci. Stat. Comput.* **1**, 1 (1980).
15. *The Connection Machine: Technical Summary* (Thinking Machines Corporation, Cambridge, MA, 1989).
16. A. F. Ghoniem and F. S. Sherman, *J. Comput. Phys.* **61**, 1 (1985).
17. C. Greengard, *J. Comput. Phys.* **61**, 345 (1985).
18. L. Greengard and V. Rokhlin, *J. Comput. Phys.* **73**, 325 (1987).
19. O. Hald, *SIAM J. Sci. Stat. Comput.* **7**, 1373 (1986).
20. O. H. Hald, *SIAM J. Numer. Anal.* **16**, 726 (1979).
21. O. H. Hald, *SIAM J. Numer. Anal.* **24**, 538 (1987).
22. S. L. Johnsson and C. T. Ho, *IEEE Trans. Comput.* **38**, 1249 (1989).
23. D. G. Long, *J. Am. Math. Soc.* **1**, 799 (1988).
24. A. Majda, "Vortex Dynamics: Numerical Analysis, Scientific Computing, and Mathematical Theory," ICIAM, '87.
25. J. P. Mesirov and W. Taylor, A continuum of N-body solvers, unpublished manuscript, (1988).
26. M. Perlman, *J. Comput. Phys.* **59**, 200 (1985).
27. E. G. Puckett, *SIAM J. Sci. Stat. Comput.* **10**, 298 (1989).
28. E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms* (Prentice–Hall, Englewood Cliffs, NJ, 1977).
29. J. A. Sethian, "A Brief Overview of Vortes Methods," in *Vortex Methods and Vortex Motion*, edited by K. Gustafson and J. Z. Sethian, Frontiers in Applied Mathematics (SIAM, Philadelphia, 1990).
30. J. A. Sethian, *J. Comput. Phys.* **54**, 425 (1984).
31. J. A. Sethian and A. F. Ghoniem, *J. Comput. Phys.* **74**, 283 (1988).
32. J. A. Sethian and J. B. Salem, *Int. J. Supercomput. Appl.* **3**(2), 10 (1988).
33. J. A. Sethian, J-Ph. Brunet, A. Greenberg, and J. P. Mesirov, A multiwire implementation of a general vortex code for 2-dimensional turbulent flow, in preparation.
34. J. A. Sethian, J-Ph. Brunet, A. Greenberg and J. P. Mesirov, CPAM Report 504, Dept. of Mathematics, Univ. of California, Berkeley, 1990 (unpublished).