Parallel Computing 7 (1988) 439–443
North-Holland

Free-Lagrange hydrodynamics with a distributed-memory parallel processor

Roy WILLIAMS

Concurrent Computation Project, California Institute of Technology, Pasadena, CA 91125, U.S.A.

Abstract. The PIFL (Parallel Irregular Free-Lagrange) code solves two-dimensional hydrodynamics with the mesh vertices moving with the fluid, with no rezoning. The irregular mesh is made of triangles and each processor deals with one or more connected domains of fluid. After each time step the mesh is topologically restructured, mesh points may be created or destroyed, and there is a local load-balance. Every few steps there is a global load balance. The code runs on a hypercube under Cubix and is designed to run most efficiently in the limit of a large number of large-memory processors.

Keywords. Hydrodynamics irregular free–Lagrange, distributed-memory parallel processor, hypercube.

Hydrodynamics calculations absorb a large fraction of the worlds supercomputing resources. It can be applied to the design of anything which moves through air or water, to the design of the combustion chamber of an internal combustion engine, to explosive devices, to the pipes and valves of chemical refineries, to the design of artificial heart valves.

In many cases, the interesting aspect of the fluid flow is the motion of a bounding surface, which may be an interface between two fluids such as a free surface, or for example an aircraft wing vibrating in response to the flow. At the same time, the flow may be complicated only in certain regions, such as the boundary layer on the vortex street, but smooth and uniform everywhere else. Free-Lagrange hydrodynamics becomes the method of choice for these highly inhomogeneous problems where there may be complex moving boundaries.

Conventional Eulerian hydrodynamics calculates the time-dependent state of the fluid at fixed mesh of points. Although there is the advantage that the mesh can be carefully set up in advance, the problem is that the future state of the fluid at a point depends on what is happening some distance upwind; and for high fluid velocity this nonlocality is exacerbated. In addition, a moving boundary may pass through the point, so that the very identity of the fluid being modelled changes discontinuously.

Free-Lagrange hydrodynamics calculates the position and state of a mesh of points which move with the fluid. The future state of a point now depends only on the local conditions and there are no upwind derivatives. If a point is on a boundary, it remains on the boundary, and if it is water it cannot become oil at a future time. In exchange for these features, the mesh is moving and loses any initial regularity. Indeed, it must continually be topologically changed to keep it from becoming tangled, and points created or destroyed to keep the density of mesh points within prescribed bounds. The irregularity of the mesh however brings its own advantage, which is that an adaptive mesh becomes easy; it is simply a matter of making the desired mesh-point density inhomogeneous.

0167-8191/88/$3.50 © 1988, Elsevier Science Publishers B.V. (North-Holland)

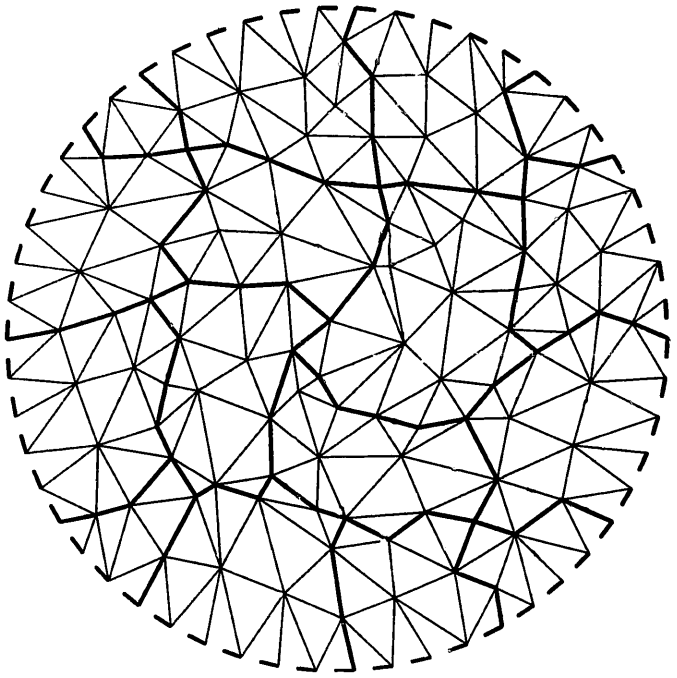

Fig. 1. Triangular mesh.

The major computational effort of hydrodynamics is the solution of a Poisson equation, usually to find the pressure field. For a fixed rectangular mesh, the Fast Fourier Transform will achieve this quickly, but for a dynamic irregular mesh, methods as fast as this are not available. The speed of the Poisson solver is heavily dependent on the 'quality' of the mesh, so that the restructuring must not only keep the mesh untangled, but also try to optimize this 'quality' measure. Quality is roughly equivalent to the condition number of the stiffness matrix, which is the matrix to be inverted for the solution of the Poisson equation.

The PIFL (Parallel Irregular Free-Lagrange) code, under development at Caltech, computes two-dimensional hydrodynamics on a distributed-memory parallel processor. The mesh is created by tesselating the area of interest with triangles the vertices of which form the mesh-points. Such a mesh is illustrated in Fig. 1, which is a triangulation of a disk. The heavy lines represent boundaries between processor domains. Figure 2 shows a representation of the same mesh with more structural details shown. The major divisions are between processors of the parallel machine, and each processor deals with a set of triangles and their associated vertices. There are two basic data structures, the triangle and the vertex: triangles are connected to exactly three vertices, and vertices connected to an arbitrary number of triangles. Position and pressure values are stored at vertices, and velocities at triangles. The heavy lines around a processor region show a pointer from a boundary vertex to the next clockwise boundary vertex around the processor domain.

Vertices at a boundary between processors are stored several times, once in each processor, and they have the same position, pressure, etc. For the purposes of Fig. 2, these sibling vertices have been artificially separated and connected by dashed lines. The two or more siblings all represent the same 'physical' vertex, and communication between processors can be thought of as occurring between siblings along the dashed lines.

Figure 2 is the result of initially assigning each of 16 processors those triangles whose centers fall within a given square: these squares form a 4×4 lattice. The boundary of the disk is a fixed wall and there is a vortex at the center, so that the roughly square processor domains are quickly distorted to the shapes shown in the figure.

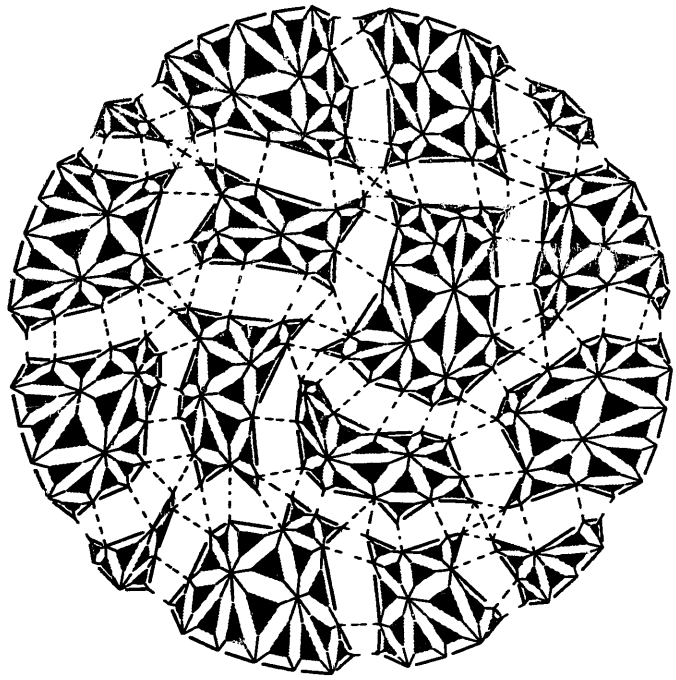

Fig. 2. PIFL mesh.

The hydrodynamics algorithm is that of Crowley, Fritts and Boris [2,5], where the vertex velocities are area-weighted averages of the surrounding triangle velocities, and the pressure is such that the discretized mass-flux divergence is zero, corresponding to an incompressible flow. The timestep is chosen to be one quarter of the smallest step which would tangle the mesh, and the algorithm uses an implicit leapfrog differential equation solver.

The code runs on a hypercube architecture under Cubix [3], and communication is by means of the Crystal Router [3] protocol; so that outgoing messages are accumulated in a buffer with a header containing the destination processor for each message. There is then a loosely synchronous communication step, after which the received messages can be extracted from a buffer.

After each time step, the mesh is restructured [6] to improve the mesh quality. Loosely speaking this means changing the connectivity of the mesh vertices to make the triangles closer to equilateral, which in turn improves the condition number of the stiffness matrix by making it diagonally dominant. In addition there are changes to increase the mesh-point density by adding more vertices, and to decrease the density be removing vertices. In this way the mesh remains untangled and the density of vertices can be kept within prescribed bounds. The desired density can be inhomogeneous, so that for example the density of mesh points close to an aircraft wing can be much greater than the density far away.

Figure 3 shows the three topological changes to the mesh used for restructuring. The first change, Fig. 3(a), exchanges the diagonal of a quadrilateral whenever the sum of the angles opposite the diagonal is greater than $\frac{1}{2}\pi$, and when all these exchanges have been done the stiffness matrix is guaranteed to be diagonally dominant [2,5,6]. The next change, Fig. 3(b), is done when an edge is too long, so that a new vertex is created at the midpoint and joined to the opposite vertices. The third change, Fig. 3(c), removes a vertex when an edge is too short: a series of diagonal exchanges takes place until the victim has exactly three neighbors. The vertex can then be cleanly removed leaving a triangular mesh. Each of these restructuring operations can happen when a processor-processor boundary passes near or through the part of the mesh being restructured; in this case however communication between the processors is needed, and

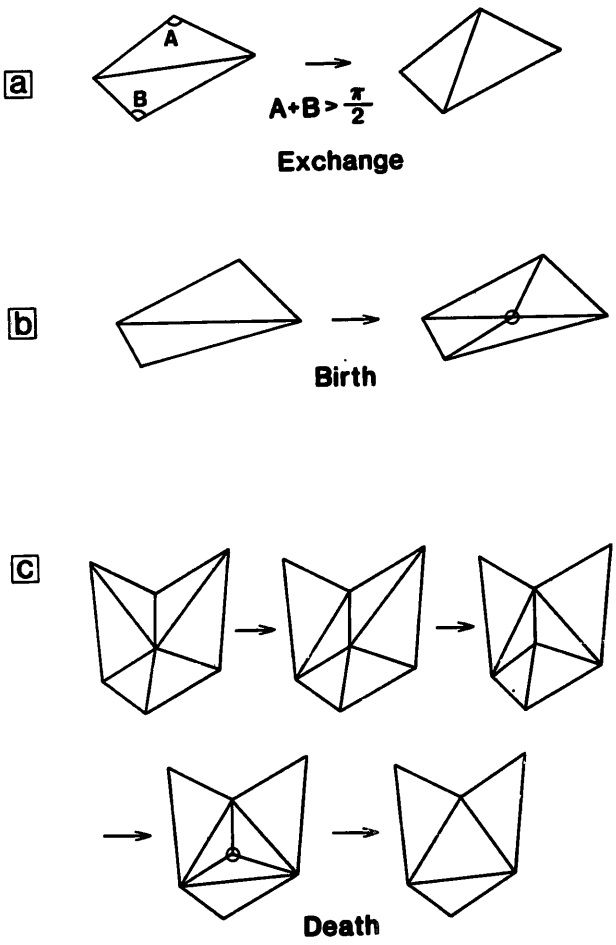

Fig. 3.

only one communication step is assigned to the restructuring whereas the internal vertices are examined many times until completion. Thus the mesh is somewhat poorer in quality near boundaries.

Dynamic restructuring provides a convenient download for a large mesh: a coarse mesh is created in one processor, then split up among several processors and the desired mesh-point density increased, then the refined mesh split again, and so on until the collective memory is filled.

Present and future work is the design of a dynamic load balancer for PIFL. This optimization algorithm must do several things under difficult conditions. Each processor must decide which, if any, of the triangles in its domain are to be given away and to whom they are to be given. This choice is based on the following criteria.

Firstly, the number of triangles per processor should be approximately equal, since for the loosely synchronous mode of parallel operation the army marches at the pace of the slowest, which is the one with most triangles. Secondly, we wish to reduce the communication as much as possible, which means reducing the total length of processor-processor boundary, weighted by the hypercube distance between the processors; this tends to deteriorate since an initially compact processor domain tends to become strung out as the simulation progresses. Thirdly, we wish to move the boundaries so that the poorer quality mesh near the boundaries is made internal to a processor and thus exposed to restructuring; this I call load balance convection.

The optimization algorithm must work in parallel [1], meaning that each processor only has information about its own domain, and possibly some information communicated from

surrounding processors. While it decides on the basis of this information its neighbor is also deciding, and the result of the simultaneous moves may be worse than the original partitioning. In addition, the load balancer must work quickly, for the time spent load balancing is unproductive housekeeping.

PIFL is very much work in progress. In addition to the dynamic load balancer, I am working on a robust and fast Poisson solver, on the introduction of interfaces between fluids, on a dynamically adaptive mesh (reacting to for example the shear rate in the fluid), and on the production of amimated videos of the moving fluid.

References

[1] F. Barajas and R.D. Williams, Optimization with a distributed-memory parallel processor, Caltech Concurrent Computation Project Report 465.

[2] W.P. Crowley, in: *Proc. 2nd International Conference on Numerical Methods in Fluid Dynamics* (Springer, Berlin, 1971).

[3] G.C. Fox, M.A. Johnson, G.A. Lyzenga, S.W. Otto, J.K. Salmon and D.W. Walker, *Solving Problems on Concurrent Processors* (Prentice-Hall, Englewood Cliffs, NJ, 1987).

[4] M.J. Fritts, in: D.L. Book, ed., *Finite-Difference Methods for Vectorized Fluid Dynamics Calculations* (Springer, Berlin, 1981); also in: M.J. Fritts, W.P. Crowley and H. Trease, eds., The Free-Lagrange Method, *Proc. 1st International Conference on Free-Lagrange Methods*, Lecture Notes in Physics **238** (Springer, Berlin, 1985).

[5] M.J. Fritts and J.P. Boris, The Lagrangian solution of transient problems in hydrodynamics using a triangular mesh, *J. Comput. Phys.* **31** (1979) 173–215.

[6] R.D. Williams, Dynamical grid optimization for Lagrangian hydrodynamics, Caltech Concurrent Computation Project Report 424.