JOURNAL OF COMPUTATIONAL PHYSICS 66, 1–20 (1986)

A Vectorized "Near Neighbors" Algorithm of Order N Using a Monotonic Logical Grid

JAY BORIS

Laboratory for Computational Physics,
U.S. Naval Research Laboratory, Code 4040, Washington, D.C. 20375

Received January 29, 1985; revised July 1, 1985

When a large number of separate objects interact, $N(N-1)/2$ interactions can occur. However, a given object usually interacts strongly with only a few of the $N-1$ others. Unfortunately, keeping lists of the other objects with which it interacts or recomputing the near neighbors each timestep is computationally expensive. This "near neighbors" problem has persisted in computational physics and computational geometry for several decades. We need efficient algorithms which select important near neighbor interactions without having to check and analyze N^2 possible interactions. To date the best algorithms which scale as N, rather than N^2, are scalar algorithms which address memory randomly.

This report introduces an efficient 3D near neighbors algorithm whose cost scales as N and which vectorizes easily using data from contiguous memory locations. A Monotonic Logical Grid (MLG) for storing the object data is defined dynamically so that objects which are adjacent in real space automatically have close address indices in the compact MLG data arrays. The data values for each object are stored at a location (i, j, k) in the MLG such that the X positions of all the objects increase monotonically with index i, the Y positions increase monotonically with index j, and the Z positions increase monotonically with index k. Such a well-structured mapping from the real positions to regular, compact data arrays can always be found. Further, when object motions result in a local violation of spatial monotonicity, another MLG always can be found nearby. This means that local changes in the object positions and hence spatial ordering do not trigger global changes in the Monotonic Logical Grid.

The data relocations required to maintain the MLG as objects pass each other in space can also be vectorized efficiently. The MLG algorithms will execute effectively in small array processors and partition to take advantage of asynchronous parallel architectures in VLSI/VHSIC-based supercomputer systems of the future. The technique seems well suited to real time applications with massively parallel distributed processing architectures.

Contents. I. Introduction and background. II. The monotonic logical grid algorithm. III. Additional aspects of monotonic logical grids. IV. Summary and Conclusions.

I. INTRODUCTION AND BACKGROUND

When N independent objects interact in space, $N(N-1)/2$ interactions might be important in determining how a given object reacts to the others at any instant. Usually exact positions and velocities of the neighboring objects must be known.

0021-9991/86 $3.00

Knowing statistical averages and general properties of the distribution of objects nearby does not provide enough data to compute local interactions accurately. At any instant a given object usually interacts strongly with only a few of the $N-1$ others. Unfortunately, keeping and updating lists of the neighbors or repeatedly recomputing which ones are near neighbors is computationally expensive. The goal is efficient, simple algorithms which select the near neighbors without a computational cost scaling as N^2. Effort on the near neighbors problem has persisted in computational physics and computational geometry for several decades. This report introduces an efficient 3D near neighbors algorithm whose practical cost scales as N and which vectorizes easily using data from contiguous memory locations.

An efficient vector solution of the near neighbors problem would advance many important applications. For an important class of molecular dynamics problems involving interactions among many atoms and molecules, the near neighbors exert the strongest forces and are the most likely candidates to enter into chemical reactions. Many important physics problems in gases, liquids, solids, and transitions among these phases require detailed many-body calculations where the close encounters are most important.

For graphics based on vertex-edge representations of complex 3D shapes, local relationships and orientations of nearby vertices determine which surfaces are visible. It is clearly advantageous to be able to construct a 2D image of a complex 3D scene, for example, using the parallelism made possible by Very Large Scale Integration (VLSI). Terrain management simulation models and multi-dimensional radiation transport models are currently limited in their ability to compute geometric obscuration. For controlling airline traffic over crowded airports, collisions with nearby planes are the most immediate danger—and demand shorter timescales for detection and corrective response. Consider the related scenario for futuristic battle area management. A one-pass engagement against many thousands of high-speed opponents requires fast redetermination of near neighbors to ensure effective targeting in real time. These applications all require rapidly updating many distinct local configurations as the objects move. Hereafter I will refer to these particles, objects, corners, etc., as nodes— concentrating on the geometric aspects of the problem.

For complex manybody problems with $N = 5000$ independent nodes, more than 30,000 degrees of freedom are required, and 12.5 million interactions exist which ideally ought to be considered. Current supercomputers deliver ~ 50 Megaflops (million floating point operations per second) on optimized but realistic problems. The straightforward recalculation of all interactions requires about 60 vectorizable operations per interaction, or 10–15 seconds of dedicated supercomputer time. This is not fast enough for applications where the data base should be updated and the neighbors recalculated every second or two in real time.

The Monotonic Logical Grid technique introduced here scales as N, in practical situations, and uses data from contiguous memory locations. A compact data structure to store the node data, called the Monotonic Logical Grid (MLG), is defined dynamically so that nodes which are adjacent in real space automatically have close

address indices in the MLG data arrays as well. As two nodes move past each other in space, their data are exchanged or "swapped" in the MLG data arrays to keep a strictly monotone mapping between the geometric locations and the corresponding storage location indices.

To construct an MLG the data values for each node are stored at location (i, j, k) in the MLG such that the X positions of all the nodes increase monotonically with index i, the Y positions increase monotonically with index j, and the Z positions increase monotonically with index k. Section II describes the algorithm in some detail.

It is not obvious but it is true that such an organized logical ordering of even random locations can always be found. In Section II an order $N \log N$ constructive algorithm for one such MLG is provided proving existence. Generally more than one MLG meeting all the monotonicity conditions seems to be possible so the technical problem of selecting the optimum MLG for a particular application has to be addressed. In one case, minimizing average distances to neighbors in the MLG may give the best grid. In other problems it may be best to maximize the shortest distance to any node which is not a near neighbor in the logical grid.

Further, when node motions result in a local violation of the monotonicity conditions on which the original MLG was based, another MLG can be found nearby. This means that local changes in the node positions and hence spatial ordering do not trigger global changes in where the node data have to be stored in the MLG. The data relocations to maintain the MLG as nodes pass each other in space can be vectorized without inefficient gather/scatter operations or variable-length (scalar) linked lists. The MLG data structure and algorithms allow contiguous-data vector operations which are long enough to be efficient for physical force sums, for $F = Ma$ orbit integrations, and for the node data "swapping" used to restructure the MLG whenever the monotonicity conditions are violated.

The cost to execute a simple test version of the model is one hour on a DEC-VAX 11/780 for 1000 particles for 1000 timesteps. We used a power series force law for the 124 nearest neighbors, assuming that the average particle separation distance is smaller than the cutoff radius R_c of the force law. Our Cray X-MP/12 treats the same calculation over 100 times faster. A minicomputer host with modest array processors would be fast enough using an MLG to integrate 5000 interacting nodes and restructure the data base thousands of times in about 15 minutes, useful for realtime applications where current supercomputers using other algorithms will be inadequate. The MLG also permits partitioning to take advantage of asynchronous multi-processor parallelism in VLSI/VHSIC-based distributed processing systems (e.g. [3]).

Brute force recalculation of all pair interactions can be vectorized but is of order N^2 and therefore costly. The best near neighbors algorithms published [1] are of order N with minimal operation counts. However, these $O(N)$ algorithms are intrinsically scalar and execute relatively poorly in parallel or pipeline-architectured supercomputers. Further, memory is addressed essentially at random so data buffering from disk or virtual memory for a large problem is time-consuming.

There are too many near neighbor algorithms and variations possible to compare all of their operation counts. Further, optimal implementations are always machine and problem dependent [2]. It is even harder to compare scalar and vector algorithms, something we would like to do in theory here but which can really only be done in practice. As a rule of thumb, efficient use of the vector hardware in supercomputers, array processors, or the new generation of parallel processors generally produces over an order of magnitude speedup over reasonable well compiled scalar code. In some cases the vector speedup factor will be much greater and in others less.

For problems where the number of important near neighbors is large enough that the computational cost is dominated by the physical interaction calculations rather than computational bookkeeping to find the near neighbors, the algorithm introduced here calculates two to three times as many interactions as minimally necessary. This is accepted as the price for simple logic and vectorized computation in contiguous memory. This means that a computer whose vector speed is only a few times the scalar speed may see no improvement over the Hockney–Eastwood PPPM techniques. In computers where the vector–scalar ratio is large, an order of magnitude improvement with an MLG is at least conceivable. More substantial gains are possible in highly parallel, multi-processor systems because the MLG algorithms partition naturally.

Let $N_{ot} = \sim 60$ be the total number of floating point operations (flops) used to evaluate each interaction between two of the $N = 5000$ nodes. The main component of the cost for a timestep will be

$$\begin{aligned} \#\ \text{Flops to compute all interactions} &= \text{Fcai} \\ &= N \times (N \times N_{ot})/2 \qquad (1.1) \\ &= 7.5 \times 10^8\ \text{flops} \rightarrow 15\ \text{sec/step at 50 Megaflops.} \end{aligned}$$

Manybody calculations which compute all interactions become prohibitively expensive with even a few hundred nodes because thousands of timesteps are required for complex problems. The operation count per timestep goes up quadratically with the number of nodes N but the effective resolution only increases as the cube root of the number of nodes. This scaling of cost with at least the sixth power of resolution is prohibitive. If the number of timesteps also has to be increased when more nodes are simulated, the scaling can be even worse. This brute force $O(N^2)$ algorithm is of interest because it vectorizes and partitions easily and is exceedingly simple.

Reduction of this computational expense is obtained by computing the details of the interactions only for pairs of nodes closer than a cutoff distance R_c. This basic nearest neighbors concept takes its most sophisticated form in the "Particle–Particle–Particle–Mesh" (PPPM) algorithms of Hockney and Eastwood [1]. Faster algorithms and data structures for implementing this nearest neighbor approximation have been the subject of much computational research in the last few decades.

Checking two locations to see if they are within a distance R_c of each other requires about $N_{oc} = \sim 10$ floating point operations. Nine or ten flops are required simply to calculate the square of the distance between the two nodes and then compare it with the square of R_c. If done for all pairs of nodes, this simple check costs 15–20% as much as the brute force calculation so relatively little is gained.

The only way to avoid the N^2 premium is to update the near neighbors list of each node using nodes in a volume larger than would be required for an interaction cutoff of R_c but much smaller than the entire system. Hockney and Eastwood define a PPPM "chaining mesh" where $dX = dY = dZ = R_c$ and check distances to objects known to be in only the nearest $13 = (3^3 - 1)/2$ cells. Only these particles might be within R_c of a particle in the chaining cell under consideration. On average only about 40% of the nodes in these 13 cells are actually within R_c. Taking L as a typical system dimension, there are $N_{cc} = (R_c/L)^3$ nodes in each of the PPPM cells on average. The number of cutoff distance checks performed in a timestep is then

$$\#\ \text{PPPM checks} = N \times 13 \times N_{cc}. \tag{1.2}$$

In the PPPM formulation, when R_c is twice the average spacing, a typical node has its distance to 104 other nearby nodes checked (13 cells × 8 nodes per cell). The corresponding number in MLG would be 62 if all interactions within two grid displacements in any direction are kept. This nominal factor of two gain in the MLG approach is lost again because all the interactions would be calculated to maintain vectorization rather than only 40% as possible with the scalar PPPM algorithm. The real gain is the ability to use efficient, contiguous memory, vector operations throughout the MLG algorithms and to cleanly partition the problem into computational subtasks.

The operation count for the overall MLG algorithm developed next in Section II is also problem dependent. Using typical simulation variables summarized in Table I below, the cost of the MLG in vector floating point operations to execute a timestep, exclusive of the relatively inexpensive orbit calculations, is

$$\begin{aligned} \#\ \text{Flops for the Monotonic Logical Grid algorithm} &= F_{mlg} \\ &= N \times (N_{nn} \times N_{ot} \text{ for neighboring node interactions} \\ &\quad + 3 \times N_{si} \times N_{os}) \text{ for swapping iterations in } X, Y, Z \\ &= 2.25 \times 10^7 \text{ flops} \rightarrow \sim 0.5 \text{ sec/step at 50 megaflops.} \end{aligned} \tag{1.3}$$

Here $N_{si} = \sim 4$ is the number of iterations of vector swapping performed over the entire grid to restructure the MLG after the node positions change each timestep. $N_{os} = \sim 60$ is the number of floating point operations to execute a single swap of the data for two nodes in the MLG.

Typical values of manybody simulation variables are shown in Table I for a 5000-node 3D calculation where $N_x \sim N_y \sim N_z \sim 15$–20. About 0.4 sec/step are required on our Cray X-MP/12, rated at about 100 Megaflops.

TABLE I

Typical Values of Manybody Simulation Variables for a 5000-Node 3D Calculation where $N_x \sim N_y \sim N_z \sim 15$–$20$

$N = 5000$	= # of nodes interacting in space
$N_{os} = \sim 60$	= # operations per vector swap in MLG algorithms
$N_{si} = \leqslant 4$	= average # of vectorized swapping iterations to relocate object data in the MLG
$N_{nn} = \sim 60$	= # of near neighbors usually included in the interaction calculations
$N_{ot} = \sim 60$	= # of flops total compute an interaction
$N_{sc} = \sim 2.5$	= # of steps between recomputation of the nearest neighbors lists in scalar algorithms
$N_{cc} =$	# of objects in the average cell of PPPM chaining mesh

Section II contains a description of the MLG itself, an $O(N \log N)$ sort algorithm to find a starting MLG from arbitrary initial data, and simple algorithms which restructure the grid dynamically as the nodes move. Section II also presents a few simple tests of the method. Section III considers several extensions. Section IV contains a summary and conclusions.

II. THE MONOTONIC LOGICAL GRID ALGORITHM

A Monotonic Logical Grid is a simple, compact way of indexing and storing the data describing a number of nodes moving in space. For N particles in three dimensions, the three arrays of locations, $X(i, j, k)$, $Y(i, j, k)$, and $Z(i, j, k)$, constitute an MLG if and only if

$$\begin{aligned} X(i, j, k) &\leqslant X(i+1, j, k) \quad \text{for} \quad 1 \leqslant i \leqslant NX-1, \\ Y(i, j, k) &\leqslant Y(i, j+1, k) \quad \text{for} \quad 1 \leqslant j \leqslant NY-1, \\ Z(i, j, k) &\leqslant Z(i, j, k+1) \quad \text{for} \quad 1 \leqslant k \leqslant NZ-1. \end{aligned} \tag{2.1}$$

Given $N = NX \times NY \times NZ$ random locations, the spatial lattice defined by an MLG is irregular. However, the cells defined by logically neighboring locations are distorted cubes and thus form a useful consistent partitioning of the spatial volume. When the N node locations satisfy Eqs. (2.1) and any additional constraints or relations specifying other than infinite-space boundary conditions, they are in "MLG order." This ordering is useful because the direction for going from one node to another in space and in the MLG is the same. Further, nodes which lie between two nodes in space will also be between them in the MLG. Thus neighbors in real space have neighboring address indices in the MLG as well.

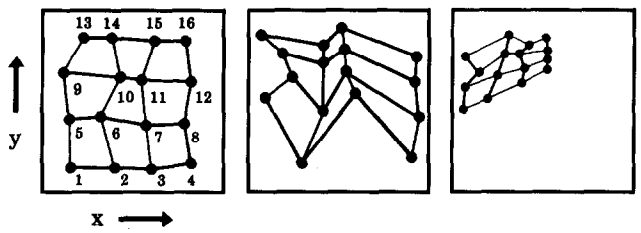

FIG. 2.1. Three 4 × 4 Monotonic Logical Grids are shown. Four rows of four nodes each represent near neighbors rather well for many spatial configurations. The cells in the MLG cluster automatically where the nodes are.

Figure 2.1 shows three different spatial configurations of 16 nodes. The nodes are ordered into four rows and four columns in each of these configurations corresponding to regular storage of the data in the two-dimensional 4 × 4 MLG. The cells of the MLG move with the nodes and thus always have exactly one node in them at any time. When all the nodes move to the upper left of the region, as in the panel on the right, the MLG is just as regular as when the nodes are uniformly spaced. This mapping of irregular locations onto a very regular data structure is what permits optimal use of vector and multiprocessor hardware.

Figure 2.2 illustrates several different MLG mappings of the same 16 node locations. The upper panel shows the 16 locations in a regular spatial lattice. The obvious numbering of the locations into four rows of four nodes each is an MLG

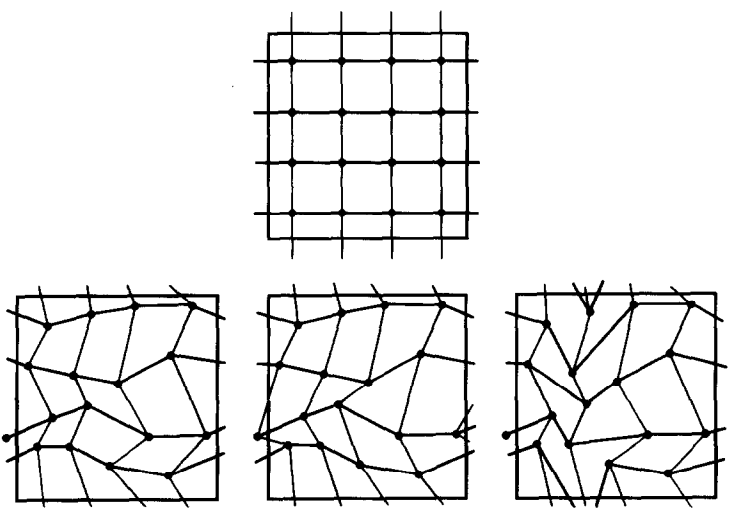

FIG. 2.2. Three Monotonic Logical Grids from identical data. The regular, doubly periodic 4 × 4 grid in the upper panel is a trivial MLG. The MLG in the lower left is a moderate distortion of the regular lattice above. The center and right-hand panels are other, somewhat more distorted MLGs indexing the same node locations as the optimal MLG on the lower left.

because the X and Y components of all the node locations (dots) increase monotonically with the X and Y indices, i and j. In the three lower panels the locations have been displaced from the regular spacing. Each of these panels contains the same data, but the MLGs for storing these data, as indicated by the logical mesh lines, all differ.

The lower left panel is a recognizable distortion of the regular grid above. The indexing of the nodes would be identical in both cases although the actual data stored would differ somewhat since the nodes have moved away from their regular locations. The lower center and lower right panels show different logical indexing for the same physical data giving two other acceptable Monotonic Logical Grids. In the center the connections to points in the second row from above and from below have all been displaced to the left. On the right, the connections to the second column from nodes located logically in columns one and three have been displaced downward.

These figures show that there can be a number of MLGs with the same Lagrangian data, all satisfying the required monotonicity conditions from Eqs. (2.1). These spatial monotonicity conditions constitute $3N - NX \times NY - NY \times NZ - NZ \times NX$ numerical comparisons which can be performed to determine if a particular organization of the node locations is in MLG order. For each dimension of the desired data structure such a monotonicity condition can be defined.

In space the coordinates can be rotated or redefined and this corresponds to a different family of MLGs for the same node positions. The monotonicity conditions may not change in the new coordinate system but the node location coordinates will. Even if the coordinate system is held fixed, node motion will quickly invalidate at least some of the relations (2.1) requiring a reorganization of the node data in the arrays to restore a completely monotone mapping. Using the monotonicity conditions, a given data structure can be checked to see if the locations are in MLG order more efficiently than distances can be compared in space. However, additional algorithms are needed when MLG order is violated.

If the nodes are not in MLG order, the following algorithm using a vector sort routine $O(N \log N)$ can be used to rearrange them. First sort all N locations into the order of increasing Z. The first $NX \times NY$ of them should be indexed $k = 1$, and sorted into the order of increasing Y. The first NX of these should be indexed $j = 1$ and then sorted into the order of increasing X. These nodes are indexed from $i = 1$ to $i = NX$. The next NX locations, indexed $j = 2$ but still $k = 1$, are again ordered and indexed from $i = 1$ to $i = NX$. This procedure is continued until the first $NX \times NY$ plane of locations has been arranged. Since the node locations were initially ordered in Z, the subsequent reorderings within the $k = 1$ plane cannot disturb the monotonicity conditions relating the first plane to any subsequent reordering of the second and subsequent planes. Similarly, all the locations will satisfy the monotonicity conditions in Y and X as well.

Once the first plane is ordered, the next $NX \times NY$ locations are indexed $k = 2$, and the MLG ordering within this plane is constructed just as for the first plane. All NZ planes are organized this way. The process requires of order

$$\begin{aligned} &NZ \times NY \times NX \times (\log NZ + \log NY + \log NX) \\ &\quad + NZ \times (NY \times NX \times (\log NY + \log NX) + NY \times NX \times \log NX) \\ &\quad = NZ \times NY \times NX \times (\log NZ + 2 \log NY + 3 \log NX) \end{aligned} \tag{2.2}$$

operations to construct the MLG. This sort algorithm could be repeated every timestep as necessary to restructure the MLG when node motions in one of the three coordinate directions cause some of the conditions (2.1) to be violated.

The existence of this constructive algorithm proves that at least one MLG for even random locations always exists and that it is not hard to find. As a consequence, data manipulation and summation algorithms in the MLG can always assume the rigorous spatial monotonicity of the MLG. When several node locations are identical, any ordering the sort procedure comes up with is correct as the conditions (2.1) are satisfied. Locally degenerate grids are possible when several locations overlap.

Although this algorithm is fast, it is of order $N \log N$, all data must be manipulated whether it is in MLG order or not, and the algorithm may move data a long distance in index space to correct even small changes in position. To counter these objections, an order $N \log N$ algorithm is described which executes local but vectorizable exchange or "swapping" operations on the MLG data to restore monotonicity everywhere. The extra factor $\log N$ now has a very small coefficient because small monotonicity upsets from the previous set of locations generally do not require information from the other side of the grid for their correction.

If two nodes move less than a typical separation distance per timestep, a condition generally required for accurate integration of the equations of motion, a few iterations are usually enough to restore MLG order. A "swap" is executed by testing the conditions in Eqs. (2.1), and then, when the corresponding monotonicity condition is violated, exchanging the locations in the logical grid of all data pertaining to the two nodes involved. Each direction is checked separately. A red–black algorithm [4] would allow at least half the tests in a given direction to be performed simultaneously and thus vectorized while converging as fast as a scalar iteration.

Five fully parallel arithmetic operations are sufficient to test for monotonicity and prepare to swap any amount of data by the usual data-splitting technique of multiplying by a one or a zero. Appropriately chosen ones and zeros can be used to exchange the different data at the node locations in six additional parallel operations per data item. When the locations are in MLG order, the swapping formulae change nothing. When two locations are out of order, these formulae interchange the node data in the MLG so they will be in order for the next iteration. The algorithm vectorizes easily and all node data at every MLG point can be treated identically.

These six operations must be repeated to swap each data variable stored in the MLG. As a minimum these variables include the three components of the node locations and an identification number, ID $\#(i, j, k)$, to mark which of the N par-

ticles currently is at i, j, k in the MLG. To vectorize the complete algorithm, the velocity components $VX(i, j, k)$, $VY(i, j, k)$, $VZ(i, j, k)$, the mass $M(i, j, k)$, and another force law constant $FC(i, j, k)$ must also be moved about dynamically. These nine variables require 54 operations to be moved between adjacent cells for each swapping iteration. Thus $N_{os} = \sim 60$ operations are required for each iteration in each direction for each node. This is about as much work as calculating three components of the force acting between two nodes which are near neighbors in the MLG. With $N_{si} = \sim 4$ swapping iterations being performed in each direction, the total cost of restoring the MLG every timestep is about the same as calculating forces from 12 neighbors. When timesteps are short, this cost reduces further.

When the MLG algorithm is used, the cost in vector floating point operations to execute the geometric and force summing in a timestep is given by Eq. (1.3). The

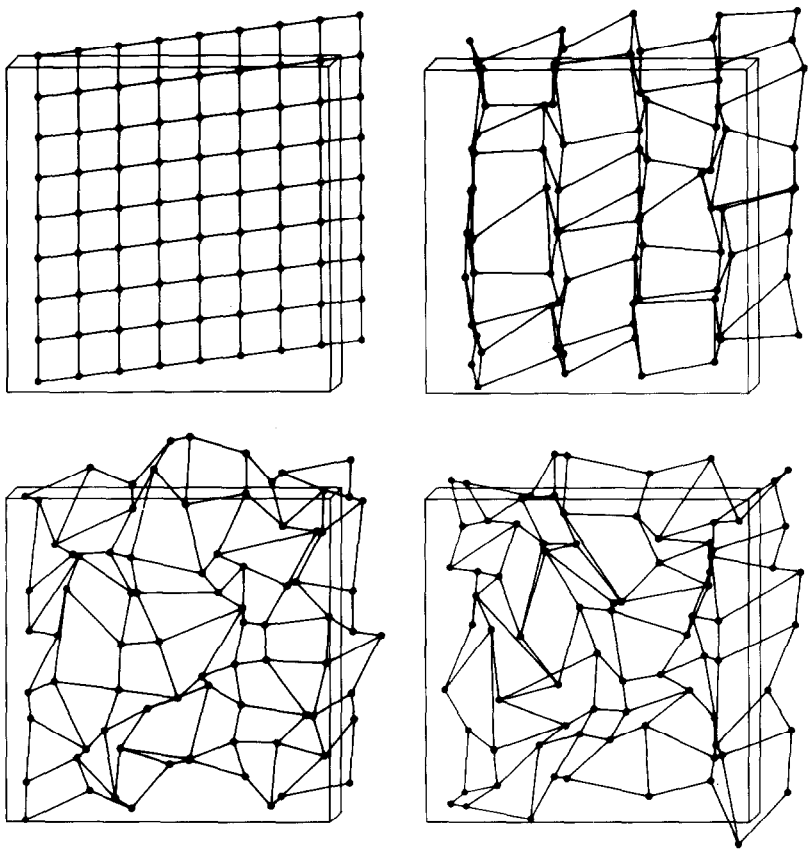

FIG. 2.3 One plane of the MLG for the 512-particle test problem is shown at four successive times as random motions distort the initially regular skew-periodic lattice. The panel in the lower right is plotted after several transits of the system by the fast particles. The average grid line length does not increase above about $\sqrt{2}$ times the initial, regular, and nearly minimal spacing.

speedup expected using this algorithm is large, a factor greater than thirty for 5000 nodes. Not only is the N^2 dependence removed but the actual near neighbor interactions can be computed with very high efficiency, comparable to the best order-N scalar algorithms. At most a fifth of the computation is expended on maintaining the MLG data structure. The rest is used in computing pairs of interactions at full vector efficiency.

The random motion of points in a cubical domain is taken as a test problem to illustrate the concepts. A topologically regular $8 \times 8 \times 8$ 3D grid is defined for storing the position and velocity components of 512 randomly located nodes. The domain is doubly periodic in X and Y and is bounded in Z by two reflecting end walls at $Z = 0$ cm and $Z = Z_{\max} = 8\,dZ$. A number of short calculations have been performed using this system to test and develop various aspects of the model. Figure 2.3 shows the first of eight planes of this 3D MLG, plotting the X and Y locations of the 64 nodes currently on that plane. The initial conditions for the calculation are shown in the upper left, regularly spaced locations with random velocities uniformly distributed in each coordinate from -10^7 to $+10^7$ cm/sec. The three remaining panels show plots of the 64 locations in the same MLG data plane at three subsequent times. As the nodes move in the plane and between planes, a complicated but clearly structured MLG is always maintained.

Under a number of different physical circumstances and numerous different initial conditions the model has been able to find an MLG after only a few swapping iterations. The average near neighbor separations increase somewhat at first over their almost minimal initial values. Rather quickly, however, random swapping halts the increase of this average distance to the nearest neighbors. The average length of lines connecting the Lagrangian nodes is only about $\sqrt{2}$ larger than the minimal lengths of the initial regular lattice.

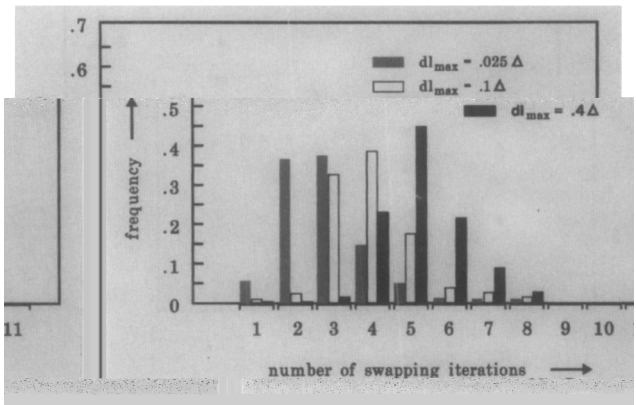

FIG. 2.4. Frequency of requiring N swapping iterations to construct a monotonic logical Grid. $dl_{\max}$ is the maximum fraction of the average spacing Δ traversed by any particle per timestep. The velocity distribution was square in the tests. As the timestep increases by a factor of 16 the number of iterations of swapping needed to restore the MLG increases only by a factor of 2, from about 2.5 iterations to about 5.

Figure 2.4 displays the frequency distribution for the number of swapping iterations required to restore the MLG after relative motion of the nodes has disrupted it. Three cases were run for the same initial distribution of zero-sized noncolliding particles, with timesteps $\delta t = 2.5 \times 10^{-16}$ sec, 1.0×10^{-15} sec, and 4.0×10^{-15} sec. The clear, outlined bars in the middle of each bin in Fig. 2.4 correspond to the intermediate case with 10^{-15} sec as the timestep. For this case $d1_{max} = 0.1\ \Delta$ meaning that the fastest particles traverse 1/10 of the regular initial spacing of $\Delta = 10^{-7}$ cm per timestep. The data with lightly shaded bars (on the left of each bin), $d1_{max} = 0.025\ \Delta$, show the results when δt is smaller by a factor of four. The data depicted with dark bars on the right show results when δt is a factor of four larger, i.e., $d1_{max} = 0.4\ \Delta$.

To interpret the figure consider $d1_{max} = 0.1\ \Delta$. About 40% of the timesteps required four iterations of swapping to restore the MLG. Less than 10% of the timesteps required six or more iterations. The average number of iterations required is 4.0 for $d1_{max} = 0.1\ \Delta$. When $d1_{max} = 0.025\ \Delta$, the average number of swapping iterations is 2.85, about $2\sqrt{2}$. When $d1_{max} = 0.4\ \Delta$, the average is 5.0 swapping iterations per timestep. Thus the actual computational work decreases per unit integration time with longer timesteps because the number of swapping iterations increases much more slowly than the timestep increases.

A great deal of swapping goes on in the first few iterations out to the average number for the particular timestep chosen. For timesteps with relatively large numbers of iterations, the likelihood of this extra work being required decreases by a factor of two or three for each extra iteration. These timesteps requiring a relatively large amount of work contribute very little to the average computation load needed to restore the MLG because they occur so infrequently.

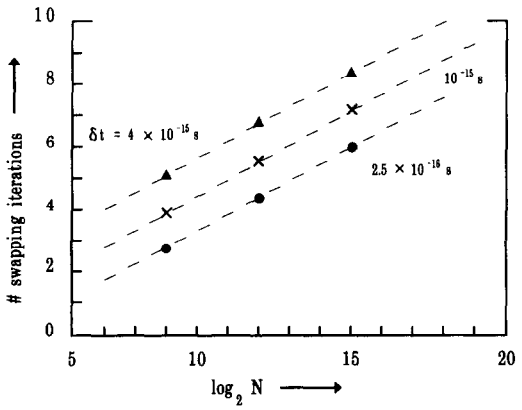

FIG. 2.5. The $N \log N$ cost of the swapping algorithm is illustrated. The average number of swapping iterations required to restore MLG order in each of three different system and timestep sizes is shown. The scaling with $N \log N$ is clearly established by the straight lines through the data points for different system size and timestep. In timings on the NRL Cray, swapping requires only about 5% of the total run time—allowing systems in excess of $1000 \times 1000 \times 1000$ before swapping becomes a significant fraction of the overall execution time.

Figure 2.5 shows the average number of swapping iterations required as a function of the base two logarithm of the system size N for the three timesteps considered above. The $N \log N$ dependence is clearly evident. For the $32 \times 32 \times 32$ node system running a molecular dynamics problem on the Cray X – MP/12 at NRL, the swapping constitutes only about 5% of the cost of the entire calculation so additional optimization is not required.

In test molecular dynamics calculations, with non-zero particle size, forces were considered between a given particle and each of the $5 \times 5 \times 5 = 125$ neighboring particles centered on it in the MLG. Since the interaction has to be computed only once for each pair of nodes in the Near Neighbors Template and can be ignored for self-interactions, these tests had

$$N_{nn} = (5 \times 5 \times 5 - 1)/2 = 62 = \sim 60 \tag{2.3}$$

near neighbors. When many nodes are within the cutoff distance R_c, the Near Neighbors Template should be extended to three planes in each direction, i.e., to $7 \times 7 \times 7$. An appreciable fraction of the forces calculated will be beyond the cutoff distance but this extra work can be performed by vector operations working from contiguous locations in the computer storage. This gain in speed is typically several orders of magnitude on highly parallel systems and is still worthwhile even if a factor of two or three is wasted calculating unnecessary interactions.

When nodes are far apart compared to the cutoff radius R_c, only the 13 neighbor interactions from the $3 \times 3 \times 3$ interaction template need be considered. This number 13 is the same as the number of chaining cells which have to be considered in Hockney's PPPM data structure to find all nodes within the cutoff radius R_c. Figure 2.6 shows a schematic rendition of three nested Near Neighbors Templates. Only the half of the template with index offset larger than zero has to be considered

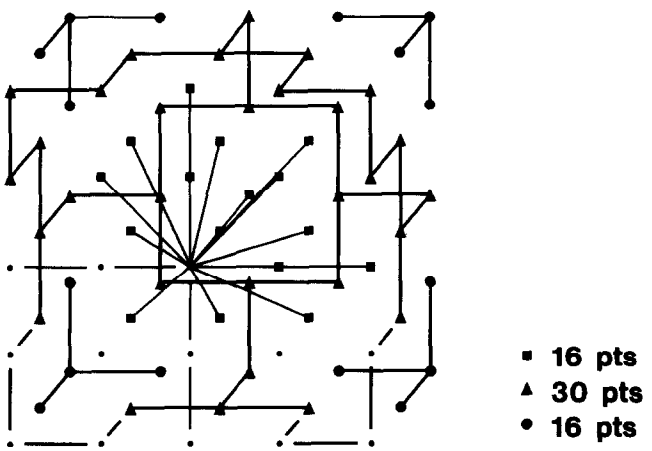

FIG. 2.6. Near Neighbors Templates for $5 \times 5 \times 5$ neighborhood. Three shells of neighbors, each on average farther from the target cell, are shown as the square, triangle, and circle points, respectively.

since all pair interactions with nodes having a lower storage address index will have been calculated previously. As shown, shells of interaction can be defined which correspond approximately to neighbors at different physical distances. The 16 neighboring nodes indicated with squares form the closest shell. The 30 triangle nodes are a bit further away, on average, and the 16 circle nodes form the furthest shell of the NNT.

III. Additional Aspects of Monotonic Logical Grids

The MLG replaces a regular grid in space having a variable number of nodes in each cell for an irregular spatial grid which has exactly one node per cell by construction. This logical and computational simplification brought about by the MLG mapping permits extensive optimization under current and planned supercomputer architectures (e.g., [3, 5]) without sacrificing the generality needed to make it useful. Thus the MLG algorithms can be applied to gas, solid, and liquid systems using the same logical structure for problems of interesting size, i.e., 1000–10,000 particles.

Optimization of nearest neighbor algorithms for particle dynamics is both machine and problem dependent. Vectorization techniques to achieve very high rates of computation require that all logical and arithmetic operations be performed on organized arrays of independent data. Distributed processing approaches to massive parallelism rely on a number of self-controlled processing centers operating asynchronously, but according to fixed rules of cooperation, on an evolving data base. To take advantage of both approaches simultaneously requires being able to define a number of vectorizable segments of the problem which can be calculated independently. Furthermore, the vectors must be long enough to be computationally efficient but short enough that the memory needed in each asynchronous processing center is not prohibitively expensive. The MLG algorithms can be partitioned for multi-tasking across a number of independent vector processors.

To maximize the length of vectors within each partition when the typical MLG dimension, $NX \approx NY \approx NZ \approx N^{1/3}$, is only about 20 (8000 objects) requires treating a substantial fraction of a plane as a single vector. In the $8 \times 8 \times 8$ test problem, vectors of length 64 can be used throughout except for the X-direction monotonicity tests where vectors half as long would result. This is accomplished by collapsing several indices into one index and by paying careful attention to the boundary conditions.

Several different boundary conditions have been tested and used with Monotonic Logical Grids; reflecting conditions, 3D free space conditions, doubly periodic conditions, and doubly skew-periodic boundary conditions. The boundary conditions determine the configuration of created data placed in "guard" or "ghost" cells around the MLG data arrays. There are as many planes of guard cells beyond the core MLG as are necessary to allow the pair-interaction calculations to proceed for

nodes on the edge of the MLG without reaching for unavailable information. This common technique to simplify the program leaves hiccups in the memory storage to skip over the ghost cells.

These hiccups in usual implementations of doubly periodic boundary conditions can be avoided by connecting the end of one row $(I-NX, J)$ with the beginning of the next row $(I=1, J+1)$ logically as neighbors. They are physically close, only displaced vertically by dY in a regular lattice. This "skew-periodic" MLG structure is shown in the upper right panel of Fig. 2.3. There the ninth row and column of image points are depicted above and to the right respectively of the fiducial space outlined in fine lines. This indexing scheme allows $NX \times NY$ nodes to be treated as interior points without indexing hiccups. In this effectively one-dimensional indexing scheme, guard cells are only used at the ends of the 2D planes in memory. The dY of skewing is distributed over the whole grid and hence becomes negligible once $NX \times NY$ reaches 64 or so.

Each of these different conditions requires different, straightforward treatments for the image nodes when swapping occurs. The only slightly tricky part is optimizing the swapping in the skew-periodic grid. [Note that one has to subtract (or add) one system length in the X-direction (i.e., $NX \times Z$) for each row down (or up) in the MLG the node data are swapped.] The overall skew-periodic MLG algorithm is as simple as the regular periodic boundary condition and replicates hypercube addressing in a regular linear memory.

Further gains are possible in principle. Optimum computational efficiency results when the last few swapping operations are performed only for the few grid points which might have become non-monotone due to adjacent swaps taking place during the previous iteration. The scalar program to perform the few remaining swaps and keep track of which few nodes might have had their monotonicity conditions affected by the previous swaps is complicated. To date, convergence relative to the force calculations has been so fast that this extra work has not been indicated. In the future it may be worth the effort for production calculations.

The same kind of gain can be obtained by trimming the Near Neighbors Template defining which logical neighbors are likely enough to be close spatial neighbors that they should be included in the vector interaction calculations automatically. When a scalar "cleanup" portion is added to the vector force summing algorithm, the number of logically neighboring nodes which are always considered can be reduced significantly below that required to ensure no near misses. Figure 2.6 shows three shells of logical interactions in the Nearest Neighbors Template (NNT), each succeeding shell taking neighbors which are logically, and usually physically, farther away.

For the few nodes which may have spatially close neighbors which are removed more than two or three locations logically, a scalar calculation can be performed. Once a node requires extra work, the scalar search can be extended to whatever logical distance is necessary to ensure that physically nearby nodes do not go uncounted.

A $5 \times 5 \times 5$ cubical NNT has 62 interactions which will be considered for each

TABLE II

The Percentage of MLG Near Misses[a]

4Å		.0039				
5Å	.0004	.0058		.0008		
6Å	.0027	.0121	.0016	.0019		
	.0055	.0585	.0678	.0499	.0051	
.0012	.0273	.1540	.1587	.1286	.0160	
.0016	.0448	.2928	.3396	.2090	.0238	
.0008	.0511	.7610	2.1150	.5980	.0226	
.0035	.1380	1.8610	4.4490	1.3080	.0756	
.0094	.2339	3.0650	7.1080	2.1180	.1458	.0008
Average node separation = 10Å in 8 x 8 x 8 MLG			Target Cell	2.2770	.0316	
				4.7060	.1415	
				7.1910	.2858	

[a] In each cell the percentage probability of a node in that MLG cell being within 4, 5, and 6 Å is given by the first, second, and third numbers, respectively. For example, the nodes that are displaced by one in both i and j from the target cell (i.e., $i+1, j+1$ or up one and one to the right) come within 5 Å of the target cell (0.5 times the average separation) only 1.3080% of the time.

node. From empirical evidence to date this is adequate provided the critical radius of consideration is somewhat less that the average separation, here taken to be the original node spacing. Relatively few near misses (1 in 10^4) can occur because neglected nodes are logically at least 3 and generally 4, 5, or 6 nodes away. When the shell is extended by about 20% (see Table II) to include nodes three steps away, fewer than 1 in 10^6 or 10^7 near misses occur.

Holes can be added to the MLG, locations which move or stay fixed in space but which don't contain a node. Any node neighboring a hole has one fewer real node in its interaction NNT. This obvious disadvantage is balanced by the fact that hole locations can be updated any way necessary to improve the locality and structure of the MLG. By adding or shifting holes about judiciously it may be possible to avoid highly distorted MLGs. The holes would be subject to swapping with nodes just as if they were nodes but their equations of motion can be different and their interactions with real nodes are zero.

Table II was computed using the 512-particle model with point non-interacting particles and the complete $7 \times 7 \times 7$ NNT. The volume around each particle was divided into shells of thickness 1 Å and the number of particles in each radial shell was counted for particles logically outside the $5 \times 5 \times 5$ template to determine how often "near misses" occur. A near miss occurs when a node gets close physically to another node without coming within the MLG Near Neighbors Template which ensures that the interaction is automatically counted in the vector sum. Concentrating first on the common features of the two physically identical calculations shown in the figure, we see that the probability of an uncounted particle penetrating the interaction volume drops off very rapidly as the distance becomes small and hence the physical interaction would be important. It is 100 times less

likely to find an undetected particle coming within 10 Å than to find one coming within 20 Å. It is another 100 times less likey to find one coming within 5 Å. No near misses were ever found less than about 3 Å. The table also shows quite clearly that nodes three locations away can be important.

A few words about the application of the MLG to Lagrangian fluid dynamics are appropriate here. Each node of the grid can be identified with a fluid or vortex element. The advantage is having a regular grid available to solve the physical evolution equations. Elliptic equations, for example, become amenable to highly efficient, vectorized multigrid methods on regular $NX \times NY \times NZ$ grids even though the fluid elements themselves move randomly. Two-dimensional and four-dimensional problems can be handled as easily by the same methods.

Work is needed, however, on telling how to evaluate spatial derivatives accurately on the distorted MLG. When nodes are far apart, the fluid cannot be as accurately represented in between as when they are close. To keep the resolution more uniform than the specific fluid flow may be capable of, it can become necessary to remove nodes where they are crowded and to inject them elsewhere to better resolve some regions. To do this in the MLG involves finding a fluid element which can be merged with a larger one nearby in a manner which conserves mass, momentum, and energy. This frees up an MLG node which can be "shifted" to the correct row, column, and plane to improve a deteriorating local resolution.

For comparison consider another free Lagrangian approach, the Lagrangian Triangular (Tetrahedronal) Grid [6–7]. In this approach the logical grid structure varies in time as the nodes move. The number of nearest neighbors can vary from node to node and the number and identity of these neighbors can vary at a given node as the Lagrangian configurations change. This extra freedom, not allowed in the MLG, is used to maintain a local grid structure optimized to guarantee diagonal dominance of the simplest conservative finite-difference elliptic operator. Scalar-linked lists become necessary to keep track of near neighbors though the resulting algorithms are still of order N. The price is the loss of local order in the grid and hence vectorization. Generalizing this Lagrangian Triangular Grid (LTG) to 3D is straightforward but practically very complicated. The grid must be composed of adaptively restructuring arrangements of tetrahedra.

Clearly the local spatial structure of the MLG is not as "good" as in the generally structured LTG but the global structure compensates for this. The monotonicity conditions specify a meaningful and useful relationship between spatial derivatives and grid differences. As a result, fluid flows with long-range correlations, unlike the random particle motions used in earlier tests, may lead to an additional computational expense at specific times. In the smooth flow of large rotating and translating vortices, an initially rectilinear grid might survive many timesteps before any of the local monotonicity constraints are violated. Nearby points would move in almost the same way. Once the fluid rotates far enough, however, monotonicity violations would have to occur. Because of the long-range correlation of the motions, a number of swapping iterations may be necessary to reset the MLG.

The MLG also suggests itself for use in multi-phase fluid problems. Each MLG

node could be used to represent a droplet in a spray or a grain of sand in a sandstorm. Droplets could have varying sizes which increase or decrease in time due to local surface effects like condensation, evaporation, or abrasion and all the droplets would not have to be simulated. The accumulation of temporal averages over times and distances short compared to changes in the background flow means that only a small fraction of all the droplets or particles would have to be followed in the MLG to get a good estimate of the interactions of the whole distribution of particles with the background gas.

The gaseous background could be represented on a Eulerian mesh to facilitate swapping of mass, momentum, and energy back and forth between particles and gas. The volume overlap of MLG cells with cells of the Eulerian grid can be used to circumvent a major complication of Monte Carlo methods, choosing the values of continuum functions at places where there are no particles or Lagrangian nodes. The MLG cells provide a natural way to interpolate back and forth between the two representations.

When insolation of dust or droplet clouds is important, the MLG provides a simple way to assess the radiation opacity along any particular direction. The grid axes can be rotated in a given direction and the swapping algorithm used to resort the nodes along that direction. There are a number of line-of-sight obscuration problems where this flexibility will be useful.

The Monotonic Logical Grid has been given only a geometric context so far. In the applications and examples above, the moving objects are being arranged relative to each other in 3D Cartesian space. Other more abstract applications suggest themselves. The MLG can just as easily represent multi-dimensional phase spaces for Boltzmann and Vlasov equations. The grid may also be useful for some classes of problems involving more abstract data organization.

IV. Summary and Conclusions

This report introduces a vector algorithm to determine near neighbors whose cost scales essentially as the number N of independent nodes. This is accomplished by defining a Monotonic Logical Grid for storing the node data dynamically so that nodes which are adjacent in real space are automatically close neighbors in the logical grid as well. As a simple geometric test problem, a regular $8 \times 8 \times 8$ 3D grid [redacted] particles in a cubical domain. For this idealized system the nodes were given random velocities and the MLG was evolved for many transits of the system by the faster particles. Statistics on near misses by logically far away nodes and on the number of swapping iterations required to restore the MLG were presented.

It was found that the restructuring of the dynamically changing MLG can generally be computed locally in a very few vectorized iterations without using inefficient scalar memory references or discontiguous memory operations. Almost all of the grid restructuring occasioned by nodes passing each other occurs in the

first two or three vectorized iterations. Further optimization is possible but the $N \log N$ scaling of MLG restoration has such a small coefficient that it is not necessary. It is also found that the spatially close nodes are nearby in the MLG as well. Two or three MLG index displacements effectively define the spatial near neighborhood except for a vanishing small number of cases which can be detected and corrected inexpensively.

The MLG differs from previous near neighbor algorithms. It effectively removes the constraint of having to associate a cell of the logical grid with a fixed region of real space. It introduces the simplifying "constraint" of only one node per computational cell. When many of the nodes cluster somewhere, a corresponding fraction of the storage locations in the MLG is automatically associated with that region. This means that substantial variations in node density are adaptively gridded by the MLG and large regions of space, as well as computer memory, are not occupied by empty cells.

This algorithm gives regular global orderings of the node data and so allows efficient, contiguous, concurrent vector operations which are longer than the relatively small number of neighbors considered for each node but which can be much shorter than the total number N. The algorithm will execute efficiently in vectors of small array processors and permits direct partitioning to take advantage of massive asynchronous parallelism in VLSI/VHSIC-based distributed processing systems. The cost to execute the current version of the model is one hour on a DEC VAX 11/780 for 1000 particles for 1000 timesteps when a simple force law for the 124 nearest neighbors is used. With commercially available highly parallel systems, tens of thousands of interacting objects could be monitored and the data base restructured thousands of times in about 15 minutes, fast enough for important realtime applications.

A number of potential applications were discussed briefly. Other uses will suggest themselves as the good properties and restrictions of the MLG mapping between real space and relative (logical or computer storage) space become better understood. Practical experience with the MLG is still limited so major surprises may yet be uncovered in some applications.

Many MLG configurations may be possible for the same physical node arrangements and simple examples suggest that the best configurations are much better than the worst. Thus efficient methods of optimizing local and global structure within the monotonicity constraints will eventually be imperative. Additional work is needed on the following questions :

1. What is the mathematical nature of the simple representations for spatial derivative operators and integral conservation operators and how can they be optimized computationally?

2. Is there an algorithm to optimize the grid structure using holes and/or adaptively varied local modifications of the monotonicity functions?

3. What is the cost of not reaching monotonicity every cycle?

4. What is the geometric or information theoretic meaning behind the ambiguity of possible representations, i.e., what kind of an uncertainty principle does this represent?

5. Is there a proof of convergence for the swapping iteration procedure?

ACKNOWLEDGMENTS

I would like to acknowledge years of informative and rewarding discussions with Dr. Martin Fritts on topics ranging over all aspects of this subject. His diligent and creative efforts on the LTG approach have provided an information base for the development of the MLG algorithm. I would also like to thank Dr. Sam Lambrakos for providing the information in Fig. 2.4 and 2.6 and Table II. This work was supported by the Office of Naval Research Projects in Large-Scale Scientific Computing (44-1909), RR014-03-05), Computational Hydrodynamics (44-0573, RRO 1403-02), and Molecular Dynamics (44-1950-0-5, 61153N), and by the Naval Research Laboratory.

REFERENCES

1. R. W. HOCKNEY AND J. W. EASTWOOD, Computer simulation using particles, (Mc. Grax–Hill, New York, 1981), Chap. 8, p. 267.
2. W. F. GUNSTEREN, H. J. C. BERENDSEN, F. COLONNA, D. PERAHIA, J. P. HOLLENBERG, AND D. LELLOUCH, *J. Comput. Chem.* **5** (1984), No. 3, p. 272.
3. G. C. FOX AND S. W. OTTO, *Phys. Today*, p. 50 (May 1984).
4. L. M. ADAMS AND H. F. JORDAN, "Is SOR Color-Blind?", ICASE Reprot No. 84-14, NASA Langley Research Center, May 1984 (unpublished).
5. E. J. LERNER, *High Technology*, 20 (July 1985).
6. W. P. CROWLEY, "FLAG: A Free-Lagrange Method for Numerically Simulating Hydrodynamic Flows in Two Dimensions," pp. 37–43, *in* "Proceedings of the Second International Conference on Numerical Methods in Fluid Dynamics," Lecture Notes in Physics **8** (M. Holt, Ed.), (Springer-Verlag, New York, 1971).
7. M. J. FRITTS AND J. P. BORIS, "The Lagrangian Solution of Transient Problems in Hydrodynamics Using a Triangular Mesh," *J. Comput. Phys.* **31** (1979), pp. 173–215.