

Parallelizing Solid Particles in Lattice-Boltzmann Fluid Dynamics

Gregory Wolffe[†], John Oleszkiewicz[†],
David Cherba[†],
[†] Dept. of Computer Science
Grand Valley State University
Allendale, MI 49401

Dewei Qi[§]
[§]Dept. of Paper and Printing Science and
Engineering
Western Michigan University
Kalamazoo, MI 49008

Abstract *Computational fluid dynamics (CFD) is a widely used numerical method for simulating complex systems, with applications ranging from aerodynamic science to hydrology. The lattice-Boltzmann method is a relatively recent technique that has been shown to be as accurate as traditional CFD methods, but with the ability to integrate arbitrarily complex geometries at a reduced computational cost.*

Although fluid flow in the lattice-Boltzmann method is easily parallelized, this is not true of the specific problem of modeling solid particles suspended in a fluid. The data structures representing solid particles and the interactions required to communicate information about them constitute time-consuming and serial portions of the simulation. In this paper, we describe a parallel implementation of the lattice Boltzmann method that incorporates new data structures and a modification to the basic algorithm. The goal of these modifications is to enable strictly local, hence parallel, computations on solid particles and reduced communication overhead.

Keywords: computational fluid dynamics, lattice-Boltzmann method, cellular automata

1 Introduction

Fluid dynamics is the study of the phenomena of fluid flow. Computational fluid dynamics (CFD) is the attempt to model or predict quantitatively what will happen to fluids under various conditions or in the presence of complications. Computational fluid dynamics is a fundamental application in the area of scientific computing and is an important field of study with emerging applications in:

- Aerodynamics, hydrodynamics, and the study of turbulence
- porous media flow and petroleum extraction methods
- meteorology and oceanography
- chemical reactions and phase changes
- biological membranes and cellular surfactant effects.

Because of the complexity of the physical and biological systems being modeled, problems in computational fluid dynamics are at the limits of computational power. For this reason, computational scientists have applied parallel programming techniques to CFD problems in order to solve systems of sufficient size and physical detail in a reasonable amount of time. This approach requires careful design of data structures, effective decomposition methods, efficient parallel algorithms, optimized communication, and effective use of multiprocessors and computing clusters [1].

This paper is concerned with the particular problem of modeling the behavior of solid particles suspended in fluid flows. Spherical and non-spherical particle suspensions with fluid flow occur in a variety of industries, such as petroleum and paper engineering. Typically, numerical simulations based on the Navier-Stokes equations have been used to model and study these complex particle-fluid systems [2]. Recently, cellular automata systems known as lattice methods have been applied to this problem [3, 4]. In this paper, we describe an enhancement to a technique known as the lattice-Boltzmann method; specifically, we present new data structures and a novel algorithm that enables parallel computation of

This research was partially supported by the PRF fund of the American Chemical Society (Grant #35180 B9).

the solid particles in addition to parallelized fluid flows.

The structure of this paper is as follows: the next section presents background information on the methods used to simulate complex particle-fluid systems. This is followed by a description of the “solids” problem and inefficiencies found in current implementations. We then present the details of our parallelized solids technique with a brief analysis focusing on scalability issues and speedup. A conclusion summarizes the contributions of our research and suggests the directions it is expected to take in the future.

2 Fluid Flow Theory and Models

In this section we describe some of the approaches used to solve fluid flow problems and discuss their relative advantages and disadvantages. These methods are used either to solve or to derive approximate solutions to the Navier-Stokes equations, a set of partial differential equations used to model fluid flow. In the general case, the Navier-Stokes equations do not have closed form solutions and are hence solved numerically.

2.1 Macroscopic methods

The first method can be called the macroscopic approach. The process involves modeling the real world as a continuum using the Navier-Stokes equations for incompressible fluid flow. The real world is then discretized in a process known as mesh generation, in which the continuum description is transformed into a discrete one. The result is a system of nonlinear equations, which are then solved using finite-element or finite-difference numerical methods.

Although this method requires that the physical properties of fluids be carefully maintained, it is well understood and has been used to model colloidal particles, sedimentation, and Couette flows [5, 6]. However, it is computationally expensive in systems employing fine resolution meshes and their correspondingly high number of unknowns to solve for. This approach is also not suitable for modeling complex geometries, such as those found in porous media.

2.2 Microscopic methods

At the other end of the scale are the so-called microscopic approaches. These methods model the atomic or molecular world and are known as molecular dynamics (MD) simulations. Each particle (atom or molecule) is tracked, using Newton's equations of motion to determine position and velocity and to solve for collisions.

Since each particle follows the exact same rules of motion, these methods are straightforward to implement, and easily parallelized. But even for massively parallel systems, solving the equations of motion for millions of particles at each instant of time is a computationally intensive process. Because of this, the size and duration of simulations is restricted to relatively small systems and relatively short runs.

2.3 Mesoscopic methods

At an intermediate scale are the mesoscopic approaches: the lattice-gas and lattice-Boltzmann methods [7]. These methods use cellular automata organized into a restricted lattice structure. Instead of discretizing the continuum Navier-Stokes equations, fluid “particles” are situated at each discrete point of the lattice. Simplified molecular dynamics – conservation of mass, momentum, and energy – is used to capture microscopic behavior. Via statistical averaging, the collective behavior of the fluid particles approximates the dynamics of the continuum equations.

The lattice-gas method is similar to molecular dynamics methods in that it tracks particle motion subject to Newton's laws and collision rules, with macroscopic quantities obtained from the particle distribution by ensemble averaging. An advantage of this approach is that due to the restrictive geometry of the lattice, computation is greatly simplified as compared to MD methods. Another advantage is that any lattice node can be marked as solid, allowing for the integration of arbitrarily complex geometries that would be difficult to model with continuum methods. The main disadvantages of the lattice-gas method are statistical noise and restrictions on Reynolds numbers and density variance.

The lattice-Boltzmann method (LBM) retains the advantages of the lattice-gas method – straightforward implementation of physical rules, strictly local computation, easy parallelization, and simple incorporation of complex geometries – while addressing the disadvantages. The basic idea is to represent fluid as a particle distribution function located at a lattice node. At discrete time steps, fluid particles move to neighboring nodes, colliding with other fluid particles. The Boltzmann equation is used to solve for the collision-induced evolution of the fluid particle velocity distribution function. This distribution function represents the expected number of particles in each of a discrete set of states or velocities defined by the lattice geometry:

$$\frac{\partial f_i}{\partial t} + \mathbf{e}_i \cdot \nabla f_i(\mathbf{x}, t) = \Omega_i(f_i)$$

where \mathbf{e}_i represents the allowable velocities and Ω is the collision operator.

In simple lattices, the velocities represent fluid particles at rest and those that flow along axial and diagonal directions (see Figure 1).

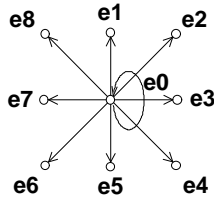


Figure 1 Discrete lattice flows

Particles arriving at the same site simulate “colliding” fluid. A simple collision rule is chosen that conserves mass and momentum and assumes that the fluid momentum “relaxes” towards an equilibrium distribution over time:

$$f_i(\mathbf{X}, t+) = f_i(\mathbf{X}, t) - \left(\frac{f_i(\mathbf{X}, t) - f_i^{eq}(\mathbf{X}, t)}{t} \right)$$

where f_i^{eq} is the equilibrium distribution and t is the relaxation time, used to define the kinematic viscosity of the fluid.

It has been proven that the incompressible Navier-Stokes equations are fully recovered at the macroscopic scale via a Chapman-Enskog-

like expansion [8]. Ladd showed how the hydrodynamic fields for mass density, momentum density and momentum flux can be described in terms of $f(X, t, v)$ and m , the molecular mass [9]. Using these equalities, macroscopic quantities at each lattice node can be obtained algebraically from moments of the distribution function:

$$\mathbf{r} = \sum_{i=0}^8 f_i(X, t) \quad \mathbf{u} = \frac{1}{p} \sum_{i=1}^8 f_i(X, t) \cdot \mathbf{e}_i$$

where \mathbf{r} is the fluid density and \mathbf{u} is the mean velocity vector of the fluid in a 2D lattice.

3 Modeling Solid Particles

The lattice-Boltzmann method is well-suited for the specific problem of modeling solid particle suspensions because of its ability to solve systems containing particles with arbitrary shapes and complex geometries.

The LBM simulation incorporates discretized solid particles that move across the nodes of the stationary lattice. The motion of the solid particles is determined at each time step from the hydrodynamic forces and torques; basically it is the summation of the momentum of all fluid flow hitting the solid particle boundaries:

$$\mathbf{f}_j = 2\mathbf{e}_i (f_i(\mathbf{X}, t+) - b_i(\mathbf{V}_b \cdot \mathbf{e}_i))$$

$$\mathbf{T}_j = \mathbf{X}_b \times \mathbf{f}_j$$

where \mathbf{f}_j is the hydrodynamic force, \mathbf{T}_j is the torque, \mathbf{V}_b is the velocity of the center of mass of the solid particle, and the b_i are coefficients chosen to conserve fluid mass. Figure 2 shows force calculations for a spherical solid particle.

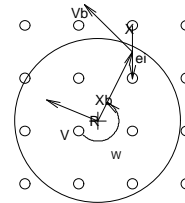


Figure 2 Velocity vectors at boundary crossing

Similarly, the motion of the solid particle affects the fluid surrounding it. For stationary solids, a no-slip boundary condition is implemented using the bounce-back method: fluid hitting a boundary reverses its velocity such that the average velocity at the boundary is zero. For moving solid boundaries the bounce-back rule is modified as suggested by Ladd to conserve fluid mass by allowing the exchange of fluid at the nodes adjacent to both sides of the solid surface.

Calculating both the translation and rotation of solid particles, specifically non-spherical particles, is important for the accuracy of the simulation. An additional factor that must be considered is the possibility of solid particle collisions. The effects of these factors can be determined in a straightforward way using Newtonian dynamics (see Figure 3).

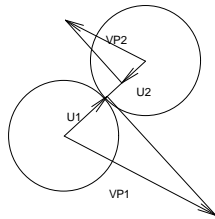


Figure 3 Collision vectors

As experience with this method broadens, it has been found that lattice-Boltzmann simulations are in excellent agreement with finite-element results calculated from the Navier-Stokes equations and with experimental observation of the behavior of solid particles (drafting, kissing, and tumbling) [10, 11].

4 Solid Modeling Inefficiencies

Several parallel models have been developed that incorporate solid particles suspended in fluid flow. As we examined these various implementations, we asked the following questions: how are solid particles represented? How is information about solid particles communicated among processors in a parallel system?

In a typical parallel implementation, a square or cubic lattice is subdivided into smaller squares or cubes; each partitioned subsquare or

subcube is assigned to a processor. Each processor steps through the lattice nodes it is responsible for, propagating fluid flow to neighboring nodes in a process called streaming, and calculating the new equilibrium distribution function at each node based on the flow received from its neighboring nodes. All computations are based on communication with nearest-neighbors, and each processor works in parallel on the nodes in its partition during a single time-step or iteration of the simulation. Each processor must then exchange information with processors who share adjacent edges or planes of the lattice (see Figure 4). This is a classic SPMD architecture for parallel programming – local parallel computation with periodic synchronization points.

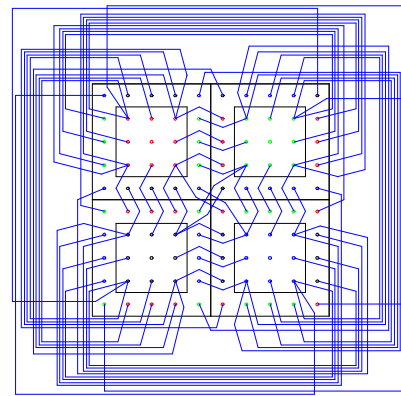


Figure 4 Communication pattern (4 processors)

However, there is no such partitioning implemented for solid particles. In answer to our question about solid particle representation, we found that in typical implementations each processor maintains a matrix of information about every solid in the simulation. At each time step, every processor calculates the cumulative momentum contribution from its fluid flows for every solid particle, regardless of where the particle is actually located. In answer to our question regarding communication, we found that in current implementations all local information concerning solid particles is broadcast to every other processor.

When a processor has received data from every other processor, it determines the relevant information (i.e. communicated forces involving solid particles in its domain) and updates its

matrix accordingly. This technique leads to inefficiencies as:

- a significant amount of superfluous information is communicated
- multiple traversals of the matrices are required for each timestep computation
- information is sent to processors that do not require it.

It also leads to scalability problems as:

- each processor added to the system causes an increase in the amount of messaging required, limiting speedup as communication costs rise
- each solid particle added to the simulation increases matrix size and subsequently the amount of system memory used for message buffering.

These are precisely the problems our approach is designed to address.

5 Parallelized Solids Technique

The basic idea behind our new model is the concept of "ownership" of solid particles. Instead of using a pure matrix representation, solid particles are considered to be objects. Any solid object whose center of mass lies within a lattice partition is owned by the processor responsible for maintaining that partition. Information regarding solid particles is communicated only to neighboring processors and only when the solid particle spans across a lattice partition boundary. Neighboring processors calculate forces for the portion of the solid particle that resides in their lattice partition and returns these results to the particle's owner.

Figure 5 depicts a spherical solid particle that spans multiple lattice spaces. There are four processors depicted in the system, arranged in a 2 x 2 matrix; each manages a partition. Lattice points are depicted as small dots; the partitioned lattice as squares with surrounding halos. The center of mass of the solid particle is located in the lower right processor – it is the “owner” of the particle. Since portions of the particle are located in other lattice partitions, the processors must communicate information about the effect of the particle on fluid flows and the forces and torque imposed by fluid on the particle.

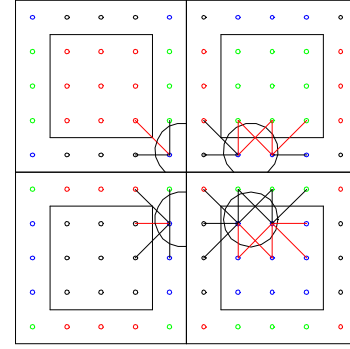


Figure 5 Solid particle spanning lattice partitions

There are four key areas where we have made significant performance improvements to the operation of the algorithm.

5.1 Streaming

Streaming is the process of communicating the calculated fluid flows from each lattice point to its nearest neighbors. Typical implementations represent fluid flows in matrix form; communicating the new flows then becomes a process of writing the data values into the appropriate locations in a matrix.

Our implementation utilizes a matrix of structures, rather than using multiple matrices for old and new flow data. We build upon the technique described above (writing to specific locations), but optimize it by precalculating the destination addresses for each flow stream. This improves performance by eliminating the complex and expensive array index calculations found in other implementations.

5.2 Flow crossings

As described in Section 3 above, boundary flow crossings occur when fluid collides with a stationary wall or a moving solid particle. These fluid-solid interactions need to be identified and incorporated into the calculations prior to the streaming step. In most implementations, the lattice points are “marked” as belonging to a solid; the entire lattice is then traversed to identify flow crossings.

Our implementation introduces a novel protocol for performing this step. The key new data structure is a list of paired nodes, where

each pair consists of adjacent lattice points, one of which is interior to a solid particle and one that is exterior. These pairs designate the boundary flow crossing points (i.e. areas in the model where fluid flow intersects a solid) and are precisely those lattice points that affect and are affected by the motion of solid particles. Specifically, each pair is used in the calculation of the hydrodynamic force and torque along a particular velocity vector.

Each processor that owns a solid particle constructs the list of pairs by examining the lattice points within a particle's bounding box. This leads to two significant improvements in performance. First, list creation and update of identified flow crossings occurs locally on the individual processors that own particles, rather than via the global traversal of matrices. Second, only those lattice points included in the list of pairs are used in subsequent computation of solid-fluid interactions, eliminating those calculations that do not contribute to the flow.

5.3 Particle collisions

Complex simulations involve numerous solid particles, increasing the probability that two particles may collide. The effect of collisions is determined using simple Newtonian dynamics; finding the component of velocity for each particle projected on the vector connecting their centers of mass. Collision detection is equally straightforward.

Our implementation optimizes collision detection by performing it during the boundary crossing calculation, when it is possible to determine if there is an object close enough to a solid particle that it needs to be checked for intersection. This allows us to efficiently perform the check on an as-needed basis only, and isolates it within the single "owning" processor's lattice space.

We also introduce the concept of an impulse force to communicate the effects of a collision between two solids. Full momentum transfer is achieved by adding the impulse force to the appropriate fluid flows to produce the desired change in velocity of a particle in a given time step. This effectively "bundles" collision data

communications within the packets used to send fluid flow data.

5.4 Communications

Fluid flows are calculated and communicated during each timestep. In addition, the new position, rotation and velocity data for each solid particle must also be communicated. Typical implementations use a "broadcast" style of communication for this operation.

We improve communication latencies in several important ways. The use of a paired-list data structure allows us to communicate fluid flow and solid particle information to only those processors that require the data. This is much more efficient than using repeated broadcasts.

In addition, by isolating calculations locally, it is possible to overlap communication with computation. Flow adjustments are made to facilitate streaming without waiting for communications. Solid particle position update packets built for adjacent lattice spaces are bundled for later transmission. The result is that calculations for fluid flow and for solid particle movement represent separate streams of execution that may proceed in a time-efficient sequence centered around two critical points.

5.5 The Complete Algorithm

A brief description of the flow of control in our implementation of the parallel LBM algorithm:

The main loop

- Update/record particle and fluid data
- Calculate equilibrium distribution function
- Determine boundary crossings
 - list of interior/exterior paired points
- Perform flow adjustment and force calculation
- Perform collision check and impulse calculation
- Communicate particle force data
- Stream flows
- Calculate new particle motion data
 - position and velocity
- Communicate particle motion data.

Figure 6 illustrates the relative ordering within the two streams of execution, and the critical points that must be observed for correct operation of the algorithm.

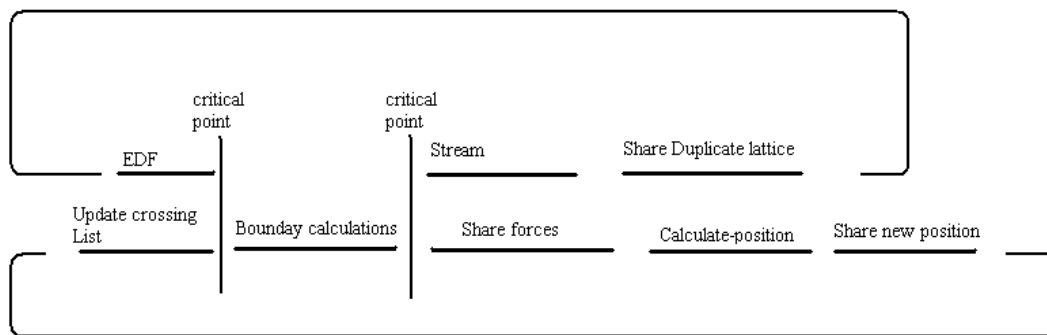


Figure 6 Algorithm critical points

6 Conclusion

Computational fluid dynamics is an area of fundamental interest for high-performance computing and is important because of the breadth of applications it encompasses. Efficient new methods are evolving, but so is the complexity of simulations, ensuring that computational demand remains high.

Modern research currently involves modeling complex fluid-solid systems, as is found in particulate suspensions. These simulations are very time-consuming due to the heavy communication requirements and the serial nature of solid particle computations. In this paper, we describe a new object-based approach that optimizes the solid particle computations, reduces the number and size of messages, and permits strictly local updating of particle information. These optimizations enable parallel computation of both solid and fluid particles. Future studies will concentrate on quantifying the effect of the optimizations.

References

[1] Wolffe, G.S., Hosseini, S.H., and Vairavan, K., Experimental study of workload indices for non-dedicated, heterogeneous systems, *Int. Conf. on Parallel and Distributed Processing Techniques and Applications*, p.470, 1997.

[2] Dias Da Cunha, R. and Luiz de Bortoli, A., A parallel Navier-Stokes solver for the rotating flow problem, *Concurrency and Computation: Practice and Experience*, 13:163-180, 2001.

[3] Wolfram, S., Cellular automaton fluids I: Basic theory, *Journal of Statistical Physics*, Vol. 45, p.471, 1986.

[4] Chopard, B. and Droz, M., *Cellular Automata Modeling of Physical Systems*, CUP, 1999.

[5] Brady, J.F. and Bossis, G., Stokesian Dynamics, *Annual Review of Fluid Mechanics*, Vol. 20, p.111, 1988.

[6] Hu, H., Joseph, D. and Crochet, M.J., Direct simulation of fluid particle motions, *Theoretical Computational Fluid Dynamics*, Vol. 3, p.285, 1992.

[7] Wolf-Gladrow, D., *Lattice-gas Cellular Automata and Lattice-Boltzmann Models*, Springer, 2000.

[8] Chen, H., Chen, S. and Matthaeus, W.H., Recovery of the Navier-Stokes equations using a lattice gas Boltzmann method. *Phys. Review*, Vol. 45, p.5339, 1992.

[9] Ladd, A.J.C., Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part I, *Journal of Fluid Mechanics*, Vol. 271, p.285, 1994.

[10] Aiden, C.K. and Qi, D., A new method for analysis of the fluid interaction with a deformable membrane, *Journal of Statistical Physics*, Vol. 90, p.145, 1998.

[11] Qi, D., Lattice-Boltzmann simulations of particles in non-zero-Reynolds-number flows, *Journal of Fluid Mechanics*, Vol. 384, p.41, 1999.