Available online at www.sciencedirect.com
ScienceDirect
Computers and Mathematics with Applications 55 (2008) 1585–1593
An International Journal
computers & mathematics
with applications
www.elsevier.com/locate/camwa

Parallel simulation of particle suspensions with the lattice Boltzmann method☆

Kevin Stratford[a,*], Ignacio Pagonabarraga[b]

[a] *Edinburgh Parallel Computing Centre, James Clark Maxwell Building, The University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ, Scotland, United Kingdom*

[b] *Departament de Física Fonamental, Universitat de Barcelona, Carrer Martí i Franqués 1, 08028-Barcelona, Spain*

Abstract

A description of the steps taken to produce a massively parallel code for particle suspension problems using the lattice Boltzmann method is presented. A number of benchmarks based on a binary fluid lattice Boltzmann model are used to assess the performance of the code in terms of the computational overhead required for the particle problem compared with the fluid-only problem, and for the scaling of the code to large processor numbers. On the Blue Gene/L architecture, the additional computational cost of particle suspensions of up to 40% solid volume fraction (here over a million particles) is negligible compared with the fluid-only code.

Keywords: Massively parallel computing; Colloidal suspensions

1. Introduction

In its most simple form, the lattice Boltzmann (LB) algorithm provides an efficient and effective way to obtain a numerical solution of the Navier–Stokes equations for incompressible isothermal fluid flow. The method differs from standard computational fluid dynamics (CFD) techniques by introducing not only a discrete configuration space, but also a discrete velocity space (for a recent review see, e.g., Ref. [1]). The algorithm then centres on a discrete distribution function $f_i(\mathbf{r}; t)$ which has one component for each of the chosen set of discrete velocities $\{\mathbf{c}_i\}$. The distribution function can be thought of as representing the density of fictitious fluid particles at the lattice site $\mathbf{r}$ with velocity $\mathbf{c}_i$; the hydrodynamic quantities are then the moments of the distribution function. In three dimensions, there are two common choices for $\{\mathbf{c}_i\}$ which ensure the correct symmetry properties of the underlying equations are observed: D3Q15 and D3Q19, having 15 and 19 velocities respectively. This large number of velocities gives rise to a significant computational cost in terms of memory. Despite this, LB has significant computational advantages,

☆ This work is funded in the UK by EPSRC GR/R67699 (RealityGrid). KS is also grateful for support from the European Community Research Infrastructure Action under the FP6 Structuring the European Research Area Programme of the HPC-Europa project (contract number RII3-CT-2003-506079).

* Corresponding author.
E-mail address: kevin@epcc.ed.ac.uk (K. Stratford).

0898-1221/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2007.08.018

in particular the almost trivial local formulation of the fluid pressure (which is easy to compute compared with the solution of a Poisson equation required by most conventional CFD methods). The resulting code can then be very concise, and is easily adapted to parallel computation. Parallel codes are important as providing sufficient resolution to capture interesting three-dimensional structure while avoiding finite-size effects requires system sizes and run times which are out of the reach of single processor machines.

Another strength of the LB algorithm is in its utility in complex fluid problems which are characterised by complicated and/or changing geometry such as fluid mixtures and suspensions of particles in fluids. For the latter, the boundary conditions required to represent moving solid objects (often based on spheres) can be included via the standard procedure of bounce-back on links, or BBL [2,4]. This technique has been used successfully to investigate, for example, sedimentation of suspensions of spheres under gravity [3], the motion of more complex objects such as ellipsoids [5], and particles in binary fluid mixtures [6]. However, the price to pay for the inclusion of moving particles is considerably more complexity in code than is required for the basic fluid LB algorithm. With this added complexity, the parallelisation becomes more difficult to achieve efficiently. The purpose of this work is to describe the steps taken to produce a highly scalable parallel code for particle suspensions using LB. The basic steps presented in Section 2 are appropriate for a single fluid LB model where particles are represented by BBL. However, once this infrastructure is in place, it can easily be extended to more complex LB problems, e.g., the binary fluid model which is used for benchmarks used in Section 3 [6].

There are two standard ways to go about parallelisation for scientific problems: OpenMP [7] – specifically for use on shared memory machines – or message passing via the message passing interface MPI [8]. While OpenMP has advantages in terms of ease-of-use, it is domain decomposition and message passing which ultimately opens the way to efficient scaling to the largest problem sizes which can then be run on the largest distributed memory machines. The challenge is then to write appropriate domain decomposition code which allows efficient performance at large problem size and large processor count to be achieved.

The particle suspension problem in LB combines two types of calculation commonly found in numerical computation. First, there is a (regular) lattice based problem of the type typically encountered in finite-difference implementations of CFD. Here, it is relatively easy to produce a static domain decomposition based on the lattice which will provide good load balance and remain fixed for the duration of the calculation. Second, there is a freely moving particle component similar to that found in classical molecular dynamics (MD) or other methods such as smoothed particle hydrodynamics [9]. Domain decomposition here tends to favour unstructured and dynamic methods which can maintain load balance for moving, and inhomogeneous, distributions of particles. (The exact approach taken might depend upon the nature and range of inter-particle forces, an important consideration in MD-like calculations.) In modelling moving particles which interact with the lattice there is a degree of tension between these two approaches. However, the fact that the particles couple to the lattice through BBL favours a regular decomposition based on the lattice; this is the approach taken here.

The paper is organised as follows. Section 2.1 provides an overview of the lattice Boltzmann method pertinent to the present work. Section 2.2 introduces the cell list as a basis for the domain decomposition and Section 2.3 describes the domain decomposition itself. Section 2.4 gives a detailed description of the parallel update algorithm. To test the efficiency of the implemented algorithm, a number of benchmark problems are performed on two different machine architectures. A summary is presented in Section 4.

2. Method

In this section, discussion is restricted to a parallel implementation with a single fluid LB model.

2.1. LB and bounce-back on links

The Navier–Stokes equations describe the time evolution of the hydrodynamic quantities related to fluid flow: the density $\rho(\mathbf{r};t)$ and the velocity $\mathbf{u}(\mathbf{r};t)$. The LB approach introduces the distributions $f_i(\mathbf{r};t)$, the moments of which provide the hydrodynamic quantities. The time evolution of the distribution function is then described by a discrete analogue of the Boltzmann equation:

$$f_i(\mathbf{r}+\mathbf{c}_i\Delta t;t+\Delta t)-f_i(\mathbf{r};t)=\sum_j L_{ij}\left[f_j(\mathbf{r};t)-f_j^{\mathrm{eq}}(\mathbf{r};t)\right]. \tag{1}$$