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:

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

Here, the left-hand side represents a streaming, or propagation, stage in which each element of the distribution is moved from a given lattice site to a neighbour in the appropriate direction $\{\mathbf{c}_i \Delta t\}$. The right-hand side represents a collision stage determined by the collision operation L_{ij}, which relaxes the distributions toward some local equilibrium $f_i^{\mathrm{eq}}(\mathbf{r}; t)$. The collision operator is generally replaced by a single relaxation time approximation [10, 11] or, increasingly, a multiple relaxation time approximation [12]. While the exact details of the collision stage are not important for this work, it should be noted that the algorithm usually proceeds by first computing a post-collision distribution $f_i^{\star}(\mathbf{r}; t)$ before performing a propagation stage.

Solid particles are represented as spheres which have a fixed radius a and have a centre of mass with position $\mathbf{r}_c$ and velocity $\mathbf{U}$ (here including both linear and angular velocity). The discrete shape of the particle is defined by a set of boundary links $\{\mathbf{c}_b \Delta t\}$ which connects fluid lattice sites to solid sites across the surface of the particle. The centre of mass moves freely across the lattice so that the discrete shape of the particle can vary. The BBL boundary condition then reflects incoming distributions at the surface with a correction that depends on the particle velocity $\mathbf{U}$ (here following Refs. [13,14]). For a given particle, the corresponding transfer of momenta can be accumulated across all links and used to compute the force on the particle.

In practice, a fully implicit velocity update scheme is used which splits the force $\mathbf{F}$ (here to include the torque) into two components

$$\mathbf{F} = \mathbf{F}_0 + \zeta.\mathbf{U}, \tag{2}$$

where $\mathbf{F}_0$ is velocity-independent (i.e., the force that would be exerted on the particle if it were not moving). The matrix ζ is interpreted as a (6×6) drag matrix, the elements of which depend upon the geometric distribution of links making up the discrete particle. BBL is then performed in a self-consistent fashion which has essentially three steps:

(1) computation of the velocity-independent force $\mathbf{F}_0$, along with the elements of the drag matrix;
(2) calculation of the new velocity and position for the particle;
(3) actual bounce-back on links, which requires the updated particle velocity.

These steps can be incorporated between the collision and propagation stages. For a single particle, the only remaining stage is to reconstruct the set of boundary links in response to the position update.

2.2. *Lubrication forces and the cell list*

One further important consideration in modelling particle suspensions in LB is the behaviour when two or more particles are near the contact. While long range hydrodynamic interactions between particles are well represented by the LB fluid, short-range lubrication interactions – those occurring at particle separations of less than one lattice spacing – are not fully resolved. For spheres, the missing part can be added back pairwise [14,16] via knowledge of the exact lubrication expression [15].

To perform the lubrication correction, pairs of particles closer than some critical separation h_c must be identified (where h is a surface-to-surface separation). Standard techniques from MD such as the cell list [17] can be employed to reduce this potentially $O(N^2)$ problem to complexity $O(N)$, which is crucial when the number of particles N becomes large. To implement the list approach, the domain is divided into cuboid cells whose minimum width is $2a + h_c$, ensuring that all pairwise interactions can be identified by examining particles in (at most) the adjacent cells. (If long range forces are required, different methods must be adopted.) The cell list will have an important role to play in the parallel implementation of the suspension problem.

If particles approach to the point of touching (where the separation h is very close to zero), this correction itself can lead to instabilities in the particle update owing to the $1/h$ divergence in the normal component of the lubrication [14]. The solution to the instability is to instigate a collective update procedure which includes the lubrication correction in the implicit particle velocity update. However, this gives rise to a linear algebra problem the cost of which scales adversely with the number of particles in mutual lubrication contact. To avoid the problem completely, one is limited to suspensions which are typically less than 40% solid volume fraction, or preventing approach to near touching via the inclusion of extra repulsive potentials at short range [19].

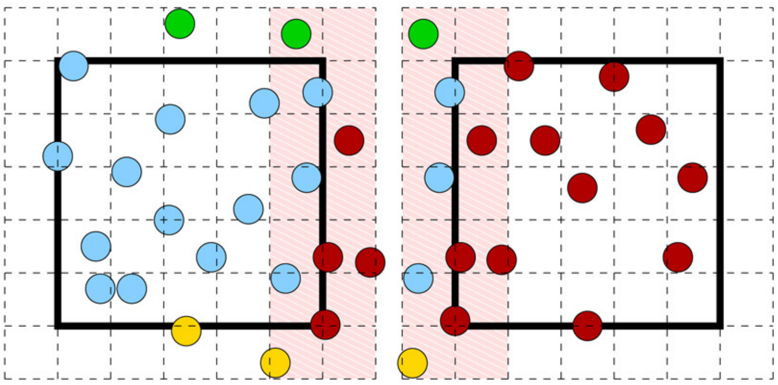

Fig. 1. A schematic diagram showing the cell list as the basis of a domain decomposition in two dimensions (the extension to three dimensions is straightforward). Each sub-domain (heavy lines) is split into a whole number of cells which constitute the cell list. Each cell may contain zero or more particles, and has a minimum width allowing all possible short-range particle–particle interactions to be identified by examining nearest-neighbour cells. One cell list's width in each sub-domain is included as a halo region which includes information on particles from neighbouring sub-domains. For example, the hatched region shows those cells involved in communication for exchanges in the left–right direction at the central boundary.

2.3. Domain decomposition

For fluid-only problem in LB, a domain decomposition approach can easily be adopted to introduce parallelism. The regular lattice is easily decomposed and equally sized sub-domains then provide a naturally load-balanced problem. A halo region, or copy of the neighbouring sub-domains, is used to receive the distribution functions at lattice sites beyond the edge of the local domain from neighbouring processors (see Fig. 2). This region is one lattice site in width to accommodate the propagation stage. This approach allows scalable codes to be developed for fluid problems [18].

This work retains this decomposition for the lattice, and extends the approach to encompass solid particles so that each sub-domain carries its own particles, and copies of those nearby about which information is required in order to compute BBL or particle–particle interactions. The natural device on which to base the decomposition is then the cell list (see Fig. 1).

The main problems involved in the domain decomposition for particle suspensions are then easily identified:

- details of particles moving from the interior of one sub-domain into a halo region must be passed to the appropriate neighbouring process or processes;
- particles with links involving lattice sites in more than one sub-domain must be constructed correctly — this may involve up to eight different processes in three dimensions;
- the sums involved in computing the velocity updates for such distributed particles must be computed with appropriate communication, and the results made available to all the relevant processes in order that BBL can take place.

At initialisation, each process is responsible for initialising the position and velocity of particles whose centres lie within the process sub-domain (denoted by the heavy lines in Fig. 1). Each particle receives a unique global identifier (an integer) which is used to distinguish the particle. State information (i.e., unique identifier, radius, position, and velocity) on particles in the cells at the perimeter of the sub-domain can then be exchanged with neighbouring processes. By limiting duplicated information to the halo regions for each process, large lattice sizes L can be achieved: while the computation will scale as the volume of the sub-domain (L^3) for uniform distributions of particles, communication scales only as the surface area of the sub-domains (L^2).

Once each process has the radius and position of all the particles in its sub-domain plus associated halo region, it can identify appropriate boundary links within its sub-domain (see Fig. 2). For the purposes of BBL, boundary links are required between fluid lattice sites in the sub-domain proper and solid lattice sites. No links are required within the lattice halo regions, as no updates to lattice properties are performed in the halo regions. This ensures all links are accounted for while avoiding duplication of links between processes.

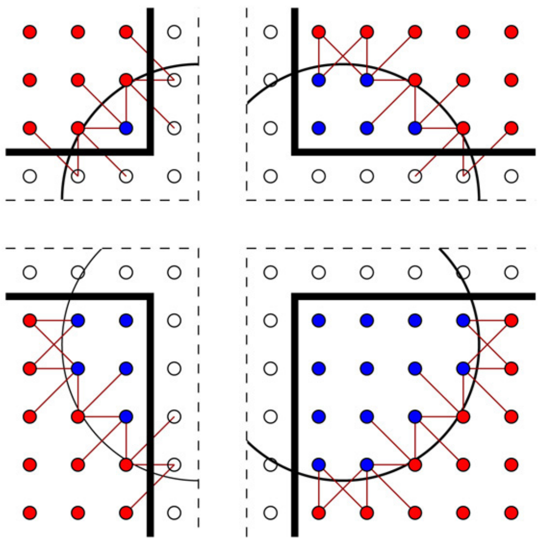

Fig. 2. A schematic figure representing the distribution of boundary links in the case where a particle is split between four processor sub-domains in two dimensions. Each sub-domain must know the position and radius of the particle, so that boundary links can be assigned between solid an fluid lattice sites. Links are included between fluid sites in the domain proper (shaded points) and solid sites in the lattice halo regions (unshaded points). There is no duplication of links between processors.

As boundary links for a given particle may be distributed between as many as 8 processes in three dimensions, communication is required to accumulate the resulting partial contributions to form a whole-particle sum. To perform BBL, all copies of the particle (i.e., all processes involved) must have the updated particle velocity. The whole-particle sum could be gathered to one processor which performs the update, the results of which are then broadcasted to the other processes involved. However, to avoid one round of communication, it is convenient to pass partial sums for all particles in the halo regions (shaded in Fig. 1) in both directions for a given dimension. In this case, all processes receive the full sum for all particles and so can compute the updated velocity.

Passing individual messages for each particle would clearly lead to a large number of messages, so particle information is aggregated into a single message to be passed at each process boundary. This single message is passed at each of the coordinate directions in turn to prevent the need for direct diagonal communication. To ensure particle information from within the aggregated message can be matched correctly upon receipt, information for particles is packed in a fixed cell list order. If there are two or more particles within a single cell, they are sorted according to unique identifier on all processes to ensure particle order within an aggregated message is maintained.

2.4. Parallel update algorithm

For completeness, the full algorithm for the parallel particle update is included here. It details five episodes of communication per lattice Boltzmann time step, two of which are associated with the lattice, and the remaining three with the particles.

(1) Communication for the lattice halo regions is undertaken.
(2) Particles for which state information must be communicated to neighbouring processes are identified. Information at each process boundary is aggregated and then sent in each coordinate direction in turn.
(3) Boundary links are constructed or reconstructed for all particles on all processes. Action arising from changes in discrete particle shape (e.g., removal or replacement of fluid at lattice sites) is performed.
(4) The cell list search is performed on each process to identify and compute short-range interactions between closely spaced particles.
(5) The collision stage for fluid sites within the sub-domain proper is computed.
(6) A second communication episode for the lattice halo regions to update the post-collision distributions $f_i^{\star}(\mathbf{r}; t)$.
(7) Communication of particle partial sums for interaction forces, and corrections arising from changes in shape are performed.

(8) A first sweep through the boundary links for each particle is performed to compute partial contributions to elements of the drag matrix ζ and the velocity-independent force $\mathbf{F}_0$.
(9) Communication of the partial sums for the drag matrix elements and velocity-independent force is performed.
(10) Particle velocity and position updates are performed for all particles on all processes.
(11) A second sweep through the boundary links for each particle is made to execute the actual bounce-back on links which updates the distributions at the appropriate fluid sites.
(12) The propagation stage is performed.

3. Results

There are a number of key issues which affect parallel performance for a given problem. One is load balance, i.e., achieving an equal distribution of work between processors. Another is the balance between communication and computation in the problem, which should favour computation. Rather than try to construct artificial problems to stress these aspects separately, we simply employ a problem of current interest to assess the performance of the method. In some ways the problem is favourable to parallel computation; nevertheless, it is a real application.

A binary fluid model is used which employs two distribution functions: one for the density and one for the composition [20]. Particles are included with no preference for either fluid so that it is energetically favourable that they straddle the fluid–fluid interface [6,21]. BBL occurs for both distributions. In comparison with a single fluid LB model, the impact of using a two-fluid model is broadly neutral — it has no affect on load balance, but may improve the computation to communication ratio. (There is considerable extra work associated with the collision stage in the binary fluid, but also twice the communication overhead for the distributions. In contrast, there is negligible extra computation or communication associated with the particles in the binary fluid problem compared with the single fluid.) In addition, the serial performance has not been specifically optimised, so the percentage of theoretical peak performance sustained is in the region of 5%.

The following sections describe benchmarks which examine the performance of two different aspects of the parallel code: first, the increase in computational cost as the number of particles is increased and second, the efficiency of the code in terms of scaling to a large number of processors for fixed problem size ("strong scaling"). So while not entirely optimal in serial, the balance of computation and communication (and hence scaling) reflects what can be achieved in a real calculation and the system sizes are representative of what might be required in real production runs for problems such as sedimentation of particles in a gravitational field or spinodal decomposition.

3.1. Platforms

The benchmarks are performed on two different architectures: a cluster of 32-way IBM p690+ Regatta systems (the UK national service HPCx) and an IBM Blue Gene/L system (at Edinburgh Parallel Computing Centre). These machines represent two different strands in modern parallel hardware. The p690+ Regatta system has relatively fast 1.8 GHz Power4 processors which can perform two floating point multiply/add operations per clock cycle. The Blue Gene/L architecture has deliberately moved to slower 700 MHz PowerPC 440 processors for reasons of power consumption and space requirement. However, there has been close attention to the communication system which means that message passing should incur minimum overhead. The machines use essentially the same compiler, and the options used are the same apart from architecture specific flags (`-qarch=pwr4` or `-qarch=440d` for the Power4 and PowerPC 440, respectively).

The Blue Gene/L system differs from the p690+ in that it has two processors per chip, or node. While the first is dedicated to computation, the second can be used to handle specifically communication or, alternatively, to augment the computational power. Thus "co-processor" (CO) mode uses only one processor per node for computation (i.e., the number of MPI processes is just the number of nodes) while "virtual node" (VN) mode uses both processors on the node for computation (the number of MPI processes is twice the number of nodes). Results for both modes will be reported.

3.2. Load-balanced problem

The benchmark undertaken in this section to assess the cost of increasing particle number is a naturally load-balanced problem of particles initialised in a random, roughly homogeneous, distribution within a system size

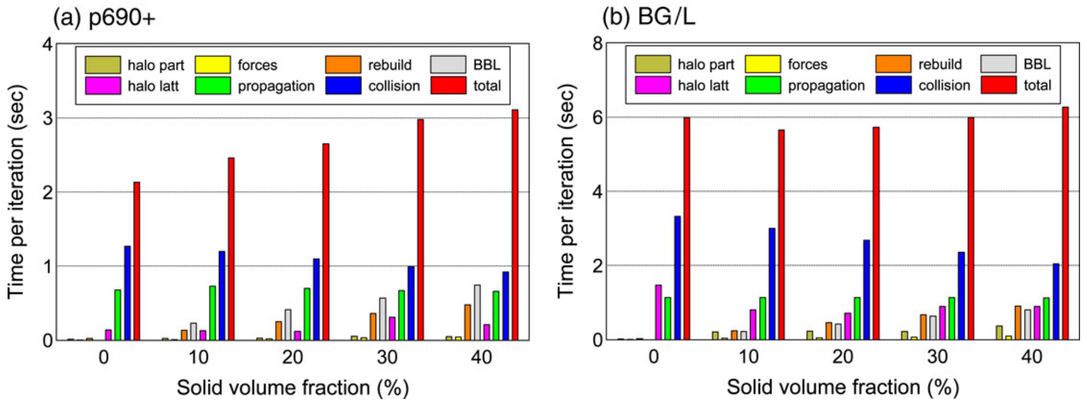

Fig. 3. The time per iteration as a function of solid volume fraction for a problem based on a 256^3 lattice and particles of radius $a = 2.3$. The problem is run on 32 processors on (a) IBM p690+ and (b) IBM Blue Gene/L systems. In (a) the increase in total time is dominated by the work done in the particle reconstruction and BBL. Note also that the time taken for the collision decreases with decreasing number of fluid sites, while that for the propagation is roughly constant. Communication time for the lattice halo swaps (which also captures load imbalance in the collision and propagation stages) remains a small part of the total. Communication for particle messages is negligible, even at the highest solid volume fraction. In (b) it can be seen that the increase in time associated with particles is almost completely offset by savings in the collision stage. The order of the bars is, from left to right: particle halo swaps, short-range force calculation between particles, particle reconstruction, BBL, lattice halo swaps, propagation, collision, and the total. Note the difference in vertical scale between the two plots.

L. The particle radius is chosen to be $a = 2.3$, and the number N is set to provide a given solid volume fraction ($4\pi a^3 N/3L^3$) between 0% and 40%. (Random initialisation takes place via a simple implementation of the Lubachevsky–Stillinger [22] algorithm.) For each problem, the model is run for 400 LB time steps and time statistics are recorded for the final 200. Each problem is repeated at least twice to ensure reproducibility.

Fig. 3 shows the breakdown of the total time for a problem size $L = 256$ on 32 processors on each of the architectures as a function of solid volume fraction. On the p690+, there is a clear increase in the overall cost moving from a fluid-only calculation to 40% solid volume fraction (here $N = 131,584$ particles). There is a small decrease in the time required in the collision stage as solid sites require no computation here. The time for the propagation remains steady (propagation is performed regardless of solid/fluid status). However, the increase in computational time for the particles (both in construction and bounce-back on links) more than offsets the saving in the collision stage, leading to the overall increase.

On the Blue Gene/L system, the situation is different. While slower than the p690+ overall, the balance of the breakdown has changed so that the saving in the collision stage now almost completely offsets the time required for particle computation. So while a factor of 3 slower for the fluid only problem, the Blue Gene/L is only a factor of roughly 2 slower at 40% solid fraction. For this architecture, the code deals with a considerably more complex problem for little extra computational cost.

For both architectures it can be seen that the additional communication costs involved with the particle halo swaps is negligible. (Communication costs for the p690+ on 32 processors are very favourable as MPI messages are exchanged via shared memory.) The total communication time, including the lattice halos (which here potentially includes load imbalance), remains at a reasonable fraction of the whole. Note that the propagation stage for the p690+ is relatively expensive compared to the collision owing to modest memory bandwidth.

It is worth noting that the cell list search and associated force calculation never occupies a significant portion of time in this implementation. If OpenMP implementation for shared memory is used (results for the p690+ not shown), this molecular dynamics-like stage, along with the particle rebuild stage can start to occupy a significant fraction of *total* time as the particle solid fraction is increased. This arises as a result of the number of synchronised updates required to shared memory in the interaction calculation and rebuild stages, and can result in very poor scaling at relatively modest processor count.

3.3. Scaling

The same load-balanced problem is now used to investigate the scaling of the code to large processor count. Two system sizes, $L = 256$ and $L = 512$ are considered along with the 0% and 40% solid volume fraction cases. Fig. 4

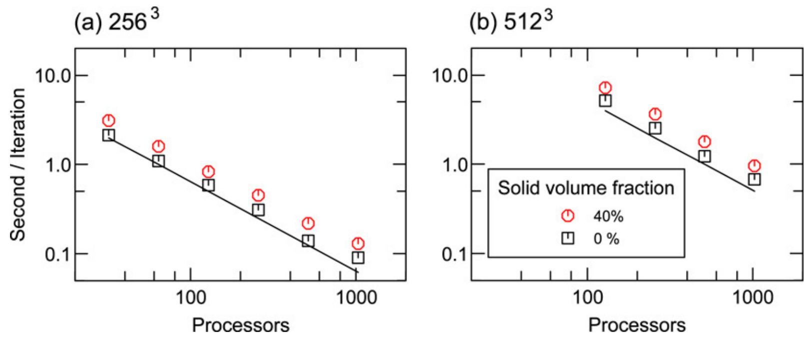

Fig. 4. The time per iteration for problem sizes of (a) 256^3 and (b) 512^3 lattice sites as a function of the number of processors on the IBM p690 system. Both problem sizes scale extremely well to the largest number of processors available (1024). The overhead for including solid particles at 40% volume fraction is roughly 50% of the fluid-only problem (40% solid volume fraction represents 131,584 and 1,052,672 particles for the smaller and larger lattice, respectively). The inclusion of particles has no impact on scalability. The line in each panel represents ideal linear scaling as a guide to the eye.

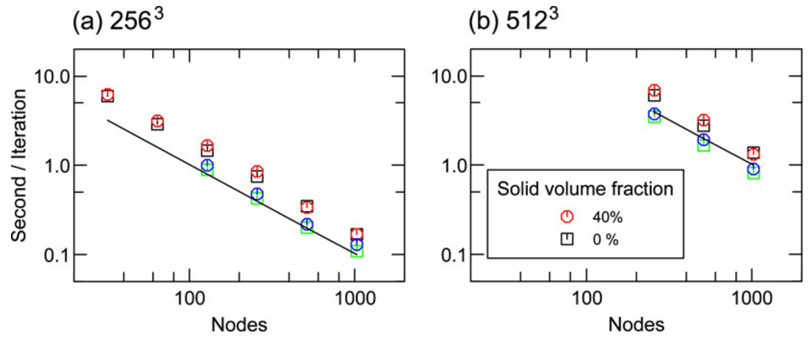

Fig. 5. As for Fig. 4 except for the BG/L architecture. The upper set of symbols in each panel represent CO mode, while the lower set are for VN mode. Comparison of the points with the linear guide line shows scaling for the larger 512^3 problem is superlinear.

shows the situation for the p690+ system. In general, the scaling is very good from 32 to 1024 processors, while the overhead for 40% solid volume fraction remains at around 50%. For the larger problem, which needs at least 128 processors owing to memory requirement the scaling is slightly more robust at the highest processor count.

Fig. 5 shows the same results for the Blue Gene/L system. Again, there is little or no overhead for 40% solid volume fraction compared with the fluid-only problem over the whole range of node count. The results for CO and VN modes are also shown: using the extra processor on each node gives a speed up of around 1.5–1.8.

3.4. Comments

In the above results, the problem is load balanced by nature. However, some problems, e.g., sedimentation of particles in a system with a bottom might give rise to significant load imbalance between regions of high particle concentration and regions with few particles. There are two possible solutions to this: first, the decomposition can be arranged to balance load by construction (possibly with some loss of scalability), or second, one could move to a dynamic domain decomposition. For example, for the sedimentation problem with bottom, the first would use a two-dimensional decomposition in the "vertical" direction, whereas the second could retain a three-dimensional decomposition but try to move work to processes nearer the top of the system. However, a number of factors mitigate against the latter for the LB problem. A dynamic decomposition which retained the same sub-domains for both particles and lattice not only involves more complex code, but might be limited in effectiveness by memory constraints. It would not be favourable to have significantly different domain sizes.

Furthermore, particle problems of different volume fractions tend to be naturally balanced; if the volume fraction is low, there is little work involved with particles and imbalance should not be a serious problem, while for high volume fractions there is little scope for imbalance owing to geometric constraints.

Note that the saving in computation in the collision stage associated with increasing number of solid sites cannot be used to offset increasing work associated with particles if the load is imbalanced; the communication and associated (weak) synchronisation required between stages (Section 2.4) preclude this.

A number of steps could be taken to improve the efficiency of the particle calculation in the current code. First, particle reconstruction in response to change in position, if required, is executed at every LB time step. As the particles move only slowly across the lattice, this cost could be taken less frequently with little impact. Second, memory for particle boundary links are allocated as required by changes in particle shape as the maximum number of links needed for a particle of given radius is not known *a priori*. This can result in links which are distributed in memory. If a maximum number of links required was available in advance, then memory could be allocated contiguously and therefore aid data locality at the BBL and reconstruction stages.

It is worth making a number of observations on software engineering. Codes of this complexity are extremely hard to debug at run-time or in post-mortem, so aggressive pro-active debugging (use of assertions) has been favoured, particularly in the case of matching particle messages, ensuring particle construction is correct, and ensuring particle copies on relevant processes are present but not duplicated. Timing information has also been gathered via hand instrumentation of the code. The use of MPI2 features, such as single-sided communication (which could, for example, be used to pass new particle information at step 2 of the parallel algorithm), were avoided as they were known to be absent on the Blue Gene/L system at the time of writing.

4. Summary

We have written a parallel lattice Boltzmann code which includes moving particles via domain decomposition and message passing using the message passing interface MPI. The benchmarks performed for naturally load-balanced problems show that despite the effort involved in terms of more complex code, the particle suspension problem can be undertaken with little or no extra computational cost depending on machine architecture. This confirms the view that LB is a remarkably effective computational tool for tackling complex fluid problems which can be extremely difficult via conventional CFD methods.

References

[1] S. Succi, The Lattice Boltzmann Equation for Fluid Dynamics and Beyond, Clarendon Press, Oxford, 2001.
[2] A.J.C. Ladd, J. Fluid Mech. 271 (1994) 285; J. Fluid Mech. 271 (1994) 311.
[3] A.J.C. Ladd, Phys. Fluids 9 (1996) 491; Phys. Rev. Lett. 76 (1996) 1392.
[4] A.J.C. Ladd, R. Verberg, J. Stat. Phys. 104 (2001) 1191.
[5] D. Qi, J. Fluid Mech. 385 (1999) 41;
D. Qi, L.-S. Luo, Phys. Fluids 14 (2002) 4440.
[6] K. Stratford, R. Adhikari, I. Pagonabarraga, J.-C. Desplat, J. Stat. Phys. 121 (2005) 163.
[7] The OpenMP standard is available from http://www.openmp.org/.
[8] The Message Passing Interface standard is available from http://www.mpi-forum.org/.
[9] L.B. Lucy, Astrophys. J. 82 (1977) 1013;
R.A. Gingold, J.J. Monaghan, Mon. Not. R. Astron. Soc. 181 (1977) 375.
[10] P. Bhatnagar, E.P. Gross, M.K. Krook, Phys. Rev. 94 (1954) 511.
[11] Y.H. Qian, D. d'Humières, P. Lallemand, Europhys. Lett. 17 (1992) 479.
[12] D. d'Humieres, Prog. Aeronaut. Astronaut. 159 (1992) 450;
D. d'Humières, I. Ginzburg, M. Krafczyk, P. Lallemand, L.-S. Luo, Philos. Trans. R. Soc. Lond. Ser. A 360 (2002) 437;
P. Lallemand, L.-S. Luo, Phys. Rev. E 61 (2000) 6546.
[13] M.W. Heemels, M.H.J. Hagen, C.P. Lowe, J. Comput. Phys. 164 (2000) 48.
[14] N.-Q. Nguyen, A.J.C. Ladd, Phys. Rev. E 66 (2002) 046708.
[15] D.J. Jeffrey, Y. Onishi, J. Fluid Mech. 139 (1984) 261.
[16] C.K. Aidun, Y. Lu, E.-J. Ding, J. Fluid Mech. 373 (1998) 287.
[17] M.P Allen, D.J. Tildesley, Computer Simulation of Fluids, Oxford University Press, 1987.
[18] J.-C. Desplat, I. Pagonabarraga, P. Bladon, Comput. Phys. Comm. 134 (2001) 273.
[19] M.E. Cates, K. Stratford, R. Adhikari, P. Stansell, J.-C. Desplat, I. Pagonabarraga, A.J. Wagner, J. Phys.: Condens. Matter 16 (2004) S3903.
[20] M.R. Swift, W.R. Osborn, J.M. Yeomans, Phys. Rev. Lett. 75 (1995) 830;
M.R. Swift, E. Orlandini, W.R. Osborn, J.M. Yeomans, Phys. Rev. E 54 (1996) 5041.
[21] K. Stratford, R. Adhikari, I. Pagonabarraga, J.-C. Desplat, M.E. Cates, Science 309 (2005) 2198.
[22] B.D. Lubachevsky, F.H. Stillinger, J. Stat. Phys. 60 (1990) 561.