The current issue and full text archive of this journal is available at
www.emeraldinsight.com/0961-5539.htm

Received 23 January 2007
Revised 10 April 2007
Accepted 17 April 2007

Performance evaluation of the cell-based algorithms for domain decomposition in flow simulation

Junye Wang
Department of Aeronautical and Automotive Engineering, Loughborough University, Loughborough, UK
Xiaoxian Zhang
SIMBIOS Centre, University of Abertay Dundee, Dundee, UK
Anthony G. Bengough
Scottish Crop Research Institute, Dundee, UK, and
John W. Crawford
SIMBIOS Centre, University of Abertay Dundee, Dundee, UK

Abstract

Purpose – The cell-based method of domain decomposition was first introduced for complex 3D geometries. To further assess the method, the aim is to carry out flow simulation in rectangular ducts to compare the known analytical solutions.

Design/methodology/approach – The method is not based on equal subvolumes but on equal numbers of active cells. The variables of the simulation are stored in ordered 1D arrays to replace the conventional 3D arrays, and the domain decomposition of the complex 3D problems therefore becomes 1D. Finally, the 3D results can be recovered using a coordinate matrix. Through the flow simulation in the rectangular ducts how the algorithm of the domain decompositions works was illustrated clearly, and the numerical solution was compared with the exact solutions.

Findings – The cell-based method can find the subdomain interfaces successfully. The parallelization based on the algorithm does not cause additional errors. The numerical results agree well with the exact solutions. Furthermore, the results of the parallelization show again that domains of 3D geometries can be decomposed automatically without inducing load imbalances.

Practical implications – Although, the approach is illustrated with lattice Boltzmann method, it is also applicable to other numerical methods in fluid dynamics and molecular dynamics.

Originality/value – Unlike the existing methods, the cell-based method performs the load balance first based on the total number of fluid cells and then decomposes the domain into a number of groups (or subdomains). Thus, the task of the cell-based method is to recover the interface rather than to balance the load as in the traditional methods. This work has examined the celled-based method for the flow in rectangular ducts. The benchmark test confirms that the cell-based domain decomposition is reliable and convenient in comparison with the well-known exact solutions.

Keywords Simulation, Flow, Computational geometry

Paper type Research paper

International Journal of Numerical Methods for Heat & Fluid Flow
Vol. 18 No. 5, 2008
pp. 656-672

0961-5539
DOI 10.1108/09615530810879765

This work was supported by the Scottish Executive Environment and Rural Affairs Department and the Biotechnology and Biological Research Council. Junye Wang would like to thank Dr N. Stringfellow, Mr K Roy, Mrs P Adamson and Mrs Z Chaplin of UK national academic supercomputing service (CSAR) for their assistances and helpful discussions with the implementation of our codes.

1. Introduction

The lattice Boltzmann method (LBM) is a powerful technique for computational modeling of a wide variety of complex fluid flows in complex geometries. Unlike the classic computational fluid dynamics (CFD), which solves the conservation equations of macroscopic properties numerically, LBM is a discrete computational method based on the Boltzmann equation. It considers a typical volume element of fluid comprising a collection of particles that are represented by particle velocity distribution functions for each fluid component at each grid nodes. The particles perform consecutive propagation and collision on a discrete lattice grid. This feature gives LBM the advantage in studying non-equilibrium dynamics, especially in multiphase flows with micro- and meso-scale interfacial dynamics and complex boundaries. However, the LBM needs to store particle distribution function besides the macroscopic velocity components in the CFD, such as the finite volume and finite difference methods. Thus, it requires more variables or computer memory. Furthermore, solving real-world problems with computational methods often requires large a domain in order to correctly resolve the complex geometries, or to produce results that are accurate and free of unwanted finite size effects (Pan *et al.*, 2004; Shahpar and Lapworth, 2003; Wang *et al.*, 2006). With the advent of computer resources, it becomes possible to employ parallel machines for the complex and large-scale flow problems.

To use parallel processing, the solvers must be designed to use an appropriate domain decomposition of the computational grid. Traditionally, the domain is divided into approximately equal subvolumes of regular slices, boxes or cubes (Desplat *et al.*, 2001; Giovanni *et al.*, 1994; Amati *et al.*, 1997). In these strategies, the computational grid is geometrically decomposed into approximately equal volumes. Kandhai *et al.* (1998) and Pan *et al.* (2004) use the orthogonal recursive bisection method to partition the computational box in which the computational grid is decomposed into partitions in an orthogonal directions. Although, the approaches of the traditional equal subvolume are successful in many engineering and scientific problems, they do not extend naturally to the problems with solid objects or obstacles in fluid, such as porous media, carotid bifurcation and aircrafts, in which the nodes or elements are not organized regularly into explicit rows and columns (Kandhai *et al.*, 1998; Pan *et al.*, 2004). Consequently, load imbalances are introduced and the behaviour of the code is affected as its overall performance becomes restricted by that of the slowest process, especially, when many processors are used. Some advanced methods of domain decompositions have been developed, such as mesh graph partitioning scheme (Karypis and Kumar, 1998a, b), dynamic mesh partitioning (Walshaw and Cross, 1999; Basermann *et al.*, 2000). These advanced methods still use the recursive bisection techniques to decompose domains, and are essentially based on equal subvolume. Load balance and minimization of the edge-cut must be obtained by using re-partitioning modules, such as mesh-migration, graph re-partitioning. A concurrent optimization of the load balancing and minimizing edge-cut has to be performed. Thus, these methods are computationally expensive because they need searching and sorting computations to find the subdomain boundaries, the extremities of the graph, or the fielder vector (Kumar *et al.*, 1994), and some of their main disadvantages cannot be overcome using these traditional recursive bisection. Hence, there are three serious drawbacks involved in these advanced methods: waste memory, irregular communication pattern, expensive computations, and coding difficulties.

In conventional CFD or lattice Boltzmann models, the workload distribution is often proportional to the number of fluid cells (or nodes) because the solid cells do not take

part in computations. Should the solid objects be distributed heterogeneously across the system, it is necessary to develop a better method of domain decomposition to ensure an efficient parallelization (Desplat *et al.*, 2001; Martys and Hagedorn, 2002). Recently, a new method of domain decomposition has been introduced by the authors for complex geometries (Wang *et al.*, 2005), which has been used in the very complex geometries like porous media. However, this method was tested only in very limited flows in porous media. Since, experimental or analytical velocity profiles in porous media are not available, it is impossible to perform a comparison of the velocity profiles in a rigid control condition. The modelling, discretization, round-off and iteration errors need to be examined further because these errors may cancel each other for complex flow in porous media structure. It is desirable to perform a rigid comparison with exact solutions to validate the model and method. Alternatively, this can be carried out in a well-known benchmark test problem, in which there are exact solutions of velocity profiles. Furthermore, it can illustrate more clearly how the method works. The flow in rectangular ducts is such a case that satisfies the above objectives.

In this paper, the method of celled-based domain decomposition is used for flow in rectangular ducts as benchmark test. This paper is organized as follow. The first half is self-contained, devoted a brief introduction to the new approach of domain decomposition. The second half investigates first the parallel performance, and then compares with the analytical solution using the LBM, followed by an evaluation of the errors of the parallel algorithm.

2. Lattice Boltzmann methods for incompressible flows

The LBM (Chen *et al.*, 1992; Frisch *et al.*, 1986; Chen and Doolen, 1998) is a meso-scale approach for CFD in which the basic idea is to solve the discretized Boltzmann equation. These standard approaches in LBM hydrodynamics recover the incompressible Navier-Stokes (NS) equations when spatial gradients in the lattice density can be neglected. Since, it is practically impossible to maintain a constant density in LBM the effect of compressibility is at least problematic to LBM simulation and deters application of the standard LBM method for incompressible flow.

Zou *et al.* (1995) introduce a modified LBM method to recover the NS equations and reinterpret the usual LBM lattice velocity and density, but it essentially adjusts the equilibrium distribution function in a simple way to eliminate the compressibility error. The incompressible LBM was derived in the 2D space (Zou *et al.*, 1995) and we extend it to 3D problems. The particle velocities and the weighting factors can be expressed using a faced centered hypercubic lattice (D3Q19), as shown in Figure 1 (Wang *et al.*, 2005; Zhang *et al.*, 2002). The six nearest neighbours are connecting the centre to the centres of the six faces of the cubes. The 12 next-to-nearest neighbours are connecting the centre to the centres of the edges of the cubes. Thus, the lattice Boltzmann equation can be expressed as follows:

$$f_i(x + \xi_i \delta t, t + \delta t) - f_i(x, t) = -\frac{1}{\tau}\left[f_i(x, t) - f_i^{\mathrm{eq}}(x, t)\right] \tag{1}$$

where $f^{\mathrm{eq}}(x, t)$ is the equilibrium distribution at x and a time, t, δt is time step, ξ_i is the particle velocity, and τ is the single relaxation time which controls the rate of system approaching to equilibrium. The equilibrium distribution function is defined by:

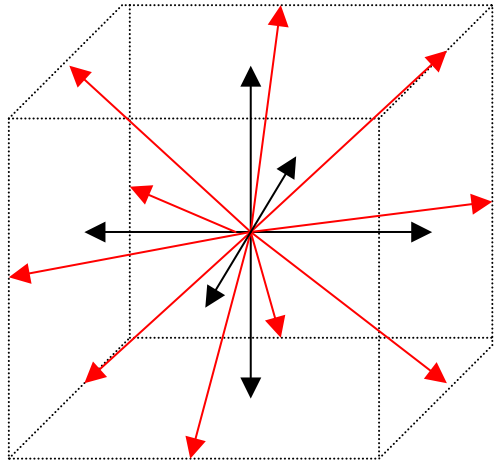

Figure 1.
The face centred hypercubic lattice (D3Q19)

$$f_i^{\mathrm{eq}} = w_i\left[\rho + 3(\xi_i \cdot u) + \frac{9}{2}(\xi_i \cdot u)^2 - \frac{3}{2}u^2\right] \tag{2}$$

where ρ is the density per node, and u is the macroscopic fluid velocity.

The weight factors used in the D3Q19 lattice, w_i, are defined as follows:

$$w_i = \begin{cases} \frac{1}{3} & i = 0, \\ \frac{1}{18} & i = 1, 2, \ldots, 6, \\ \frac{1}{36} & i = 7, 8, \ldots, 18. \end{cases} \tag{3}$$

and the fluid density and the fluid velocity are calculated from:

$$\rho = \sum_{i=0}^{19} f_i \tag{4}$$

$$u = \sum_{i=0}^{19} \xi_i f_i \tag{5}$$

Applying the Chapman-Enskog multi-scale expansion to equations (1)-(5) gives the continuity equation and incompressible NS equations in steady state:

$$\nabla \cdot u = 0 + \mathrm{O}(\delta^2) \tag{6}$$

$$u \cdot \nabla u = -\nabla(c^2\rho) + v\nabla^2 u + \mathrm{O}(\delta^2) \tag{7}$$

where c is the speed of sound ($c = \delta x/3\ \delta t$), $v = (2\tau\text{-}1)/6$ and δx is the size of cells in the lattice. Apart from the errors at higher orders, $\mathrm{O}(\delta^2)$, equations (6) and (7) are exactly the incompressible NS equations. The compressibility error induced by ignoring the change of density in the continuity equation has been eliminated.

3. Sparse systems

Sparse systems are the type of systems in which most of the matrix elements are zero, such as sparse matrices, and sparse graphs. Owing to the importance of the sparse systems, in scientific and engineering applications, the amount of literatures on parallel algorithms for sparse system is immense. Most of them are for the computation of sparse matrix equations. Since the locations of the nonzero elements in the matrix are known explicitly, unnecessary multiplications and additions with zero elements should be avoided. Hence, it is a common practice to store only the nonzero entries, and keep track of their locations in the matrix during the computation. Although, these efficient storage schemes for sparse matrices have been used extensively in matrix equation calculations (Martys and Hagedorn, 2002), there are few applications of them in the storage of spatially heterogeneous structure, such as complex geometries with solid objects, and porous media.

Previously, it has been customary to store and the compute full lattice of the geometries in a regular computational domain. This approach leads to a straightforward decomposition of parallel processing implementations. However, like sparse matrices, since the solid cells do not participate in calculations, there is no need to store the data for them. This full lattice approach is wasteful in floating point computation time and memory usage. In recent years, sparse approaches (Pan *et al.*, 2004; Martys and Hagedorn, 2002; Wang *et al.*, 2005) similar to sparse matrices have been developed for spatially heterogeneous structures. In these approaches, only the fluid cells (active sites) are stored and computed. At each active site, a pointer is used to reference the data structure. At inactive sites, the pointer is NULL; and no additional memory is allocated for the inactive sites.

4. Domain decomposition

In the proposed approach of domain decomposition, a 3D complex geometry is scanned in a Cartesian order, starting from the z-direction and then y- and x-directions. Active elements are counted and stored in ordered 1D arrays rather than in the conventional 3D arrays. Thus, all the variables of the simulation that are stored in the ordered 1D arrays are easy to be decomposed. The domain decomposition of the 3D data arrays becomes 1D. In fact, multi-dimensional arrays are just an abstraction, as we can obtain the same results with a simple array. Furthermore, through the scanning of the 3D complex geometry, the solid cells have been filtered out automatically since they do not participate in calculations. The sparse matrix approach can be integrated into the algorithm of the domain decompositions. Since the neighbouring cells are no longer immediately known in the 1D arrays, the definitions of their neighbours have to be imposed explicitly by using extra integer arrays that contain the neighbours for each of the cells. Finally, the 3D results stored the in 1D arrays can be recovered using a coordinate matrix.

For simplicity of explanation, the method of the domain decomposition is applied to a regular rectangular duct as a benchmark test. However, the method is general and can be easily extended to flows in complex geometries. The coordinates of the rectangular duct are shown in Figure 2. Firstly, the total number of fluid cells are counted and each of the variables corresponding to the fluid cells are stored orderly in 1D arrays. Then the number of the total fluid cells is divided by the number of processors. The data domain is split into several subdomains. Each of them is handled by a processor. The work is distributed across the processors in the domain.

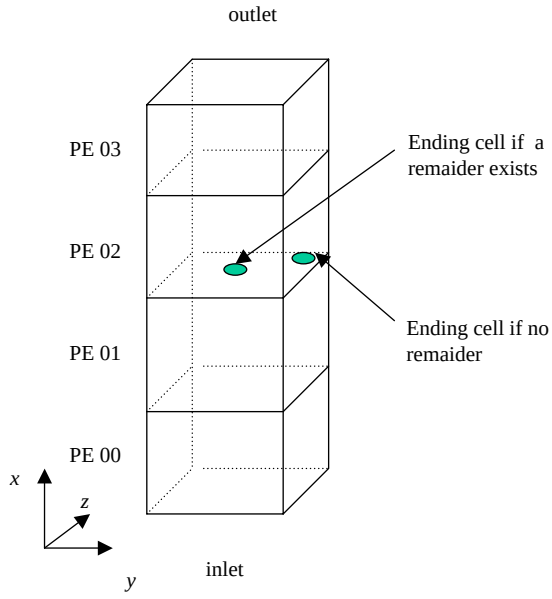

Figure 2.
A schematic diagram of the simulation domain and its coordinate axes

The starting and ending cells in each processor can be recovered using the coordinate arrays. Since all the cells on interfaces between the subdomains are ranked orderly, it is easy to obtain all the boundaries between the subdomains by using the starting and ending cell coordinates.

A consequence of this is that there are two possibilities: the total number of the fluid cells is either an exact multiple of the number of processors (no remainder) or not an exact multiple of the number of processors (remainder). Figure 3 shows two types of interfaces, in which the white cells belong to processor 1 and the grey cells to processor 2. If remainders do not exist, the size of each processor is identical, and the domain is decomposed uniformly. All the starting cells will be in the coordinate (x, 0, 0), and all the ending cells will be in the coordinates $(x, N_y - 1, N_z - 1)$, where N_y and N_z are the numbers of cells in the y- and z-directions, respectively. Plane interfaces between the subdomains are identical to those using conventional slice domain decomposition. Such a plane interface is shown in Figure 3(a). However, if a remainder exists, the remainder will be distributed further on some processors. Hence, the number of cells in some processors will be one item longer than that in others. The starting and ending cells in the processors are not generally on corners, but could be at any position. For this case, the starting and ending cells are still recovered using the coordinate arrays. The coordinates of the opposite cells of the starting and ending cells in the two neighbour surfaces are also positioned. Thus, it is easy to recover the boundaries between the subdomains through these special cell coordinates. Figure 3(b) shows an irregularly folded surface when a remainder exists, in which the interface cells are marked with B. The interface is not a plane surface but a folded surface. It should be pointed out that the boundary cells might be one extra column (or row) more than those shown in Figure 3(a) because the diagonal neighbour cells need to be communicated. For the present D3Q19 lattice, these extra diagonal boundary cells have been labelled with B^*.

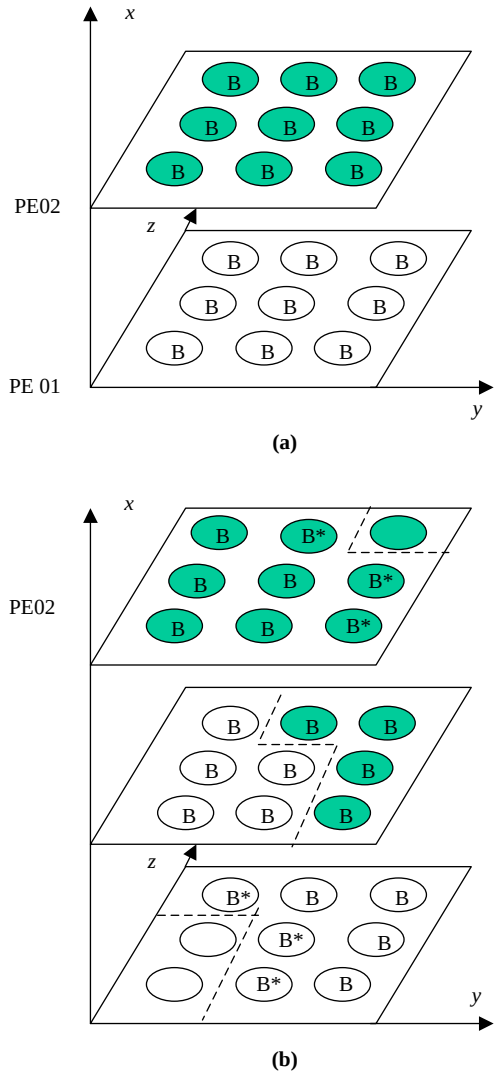

Figure 3.
Typical boundaries between subdomains; the white and grey cells belong to PE01 and PE02, respectively: (a) the interface when there is no remainder; (b) the interface when a remainder exists

5. Data structure and memory requirements

After the total number of fluid cells is counted and each variable and its coordinates associated with the fluid cells are stored in order in their 1D arrays. Figure 4 shows a simple example of a 2D 4 × 7 domain distributed on two processors. In Figure 4, the grey refers to internal cells, and the white to boundary and interface cells. The data array structures for all the variables(velocities, distribution functions, coordinates and neighbors) are the same. As an example, a corresponding 1D data array of variables is also shown in Figure 4. It can be found that although different from their natural ordering in 2D solution domain, the data in inner and interface cells are still ordering in

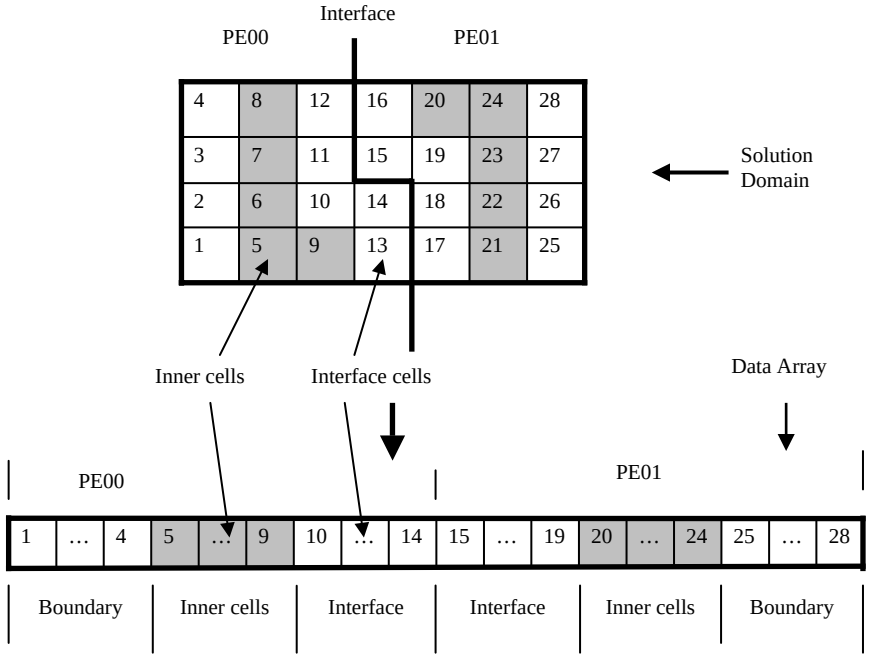

Figure 4.
Conversion from lattice cells to a sparse representation and the associated data structures

the 1D array. Particularly, their interface and boundary cells are continuous in the array. This gives regular communication patterns, and simple data structures. The starting and ending cells on the subdomain's boundary surfaces can be referred automatically by moving the pointers. Hence, the data are easy to transfer in this structure. Furthermore, subdomains that need communicate are in the nearest neighbour, for example, processor 1 is a neighbour of processor 2.

The basic advantage of the proposed storage scheme is its simplification in identifying each part of the domain. The physical domain can be decomposed directly and then mapped to the computational domain. In case of solid objects immersed in the fluid, only fluids cells are kept in memory. Owing to the consecutive and ordering storage of the boundary cells, the expensive sorting and searching operations can be avoided in locating the neighbour cells. The data structures do not need description of different types of neighbors and processor index because of the advantages of nearest communication connection, regular communication pattern, and simple data structures. More memory can be saved compared with other sparse approaches. The indices of neighbours are a 1-byte indicator to distinguish solid and fluid cells.

The current algorithms need more memory to store the coordinates and the neighbours of each cell. We can compare the total memory requirements between 1D and 3D arrays representations. For the 3D representation, the memory required by the LBM for single-phase flow can be estimated from:

$$[2 \times 19 \times \text{size of(float)} + 4 \times \text{size of(float)}]N$$

where N is the total number of the fluid cells in the lattice, the first term represents the distribution functions at the 19 directions for the current and next time steps, and the

second term represents the fluid density and the three velocity components. Assuming 8-byte for each float number, the required memory is therefore $336N$ bytes.

For the 1D representation, the memory required by the LBM for single-phase flow can be estimated from:

$$[2 \times 19 \times \text{size of(float)} + 4 \times \text{size of(float)} 3 \times \text{size of(int)} + 18 \times \text{size of(int)}]N$$

where the first two terms are the same as the 3D representation, and the third term is for the x-, y-, z-coordinates indices, and the fourth term is for 18 neighbour information. Assuming two-byte for each integer, the required memory is $378N$ bytes. Hence, the memory required in 1D representation increases approximately 12.5 per cent compared with the 3D representation. Despite the increase in storage, it is negligible in comparison with the advantages of the 1D representation.

6. Implementation details

Numerical experiments have been performed on a Silicon Graphics Origin 2000 with 128 CPUs running at 400 MHz, equipped with 128 GB of main memory, and the operating system IRIX 64 version 6.5.

The parallelization was accomplished within the simple single program multiple data model. The data volume is divided into spatially continuous blocks along the x-axis; multiple copies of the same program run simultaneously, each operating on its own block of data. At the end of each iteration, the data for the folded surfaces that lie on the boundaries between the blocks are passed between the appropriate processors before the iteration is completed. By using a ghost layer of lattice cells in the surrounding of the subdomain, the propagation step can be isolated from the data exchange step. The ghost lattices are lattices that the processor needs to compute all its stencils. However, the processor does not compute the ghost cells themselves. The ghost cells are computed during the previous step by its neighbouring processors. Before the propagation step, the processor must have received the data at the ghost layer from the neighbours at the conclusion of the previous time step, and the data in the interface are then sent to the ghost layer of its neighbouring processors.

Hence, the loop is first to calculate the equilibrium distribution function, $f_i^{\text{eq}}(x, t)$ in equation (1), and then the collision, the term on the right hand side of equation (1). Before the stream, the interface data between the sub-domain are exchanged. Finally, the first term on the left-hand side of equation (1) is computed, and the stream is performed. If convergent criteria have not been met, it returns to the first step. The loop is cyclically carried out until the convergent criteria are met.

For data transfer, the Message Passing Interface – M.P.I. Forum (1994) has been adopted, meaning that a group of processors, which can communicate with each other, can be ranked by MPI from 00 up to numProcs-1 (where numProcs is the number of processors). The MPI functions, MPI_Comm_size and MPI_Comm_rank, are used to determine the number of processors in a given group and the rank of each processor within the group. MPI_Send and MPI_Recv functions are used to transfer data values of the ghost cells between the processors. These are blocking functions, and do not return until the communication is complete. To prevent deadlocks, we authorize the odd-numbered processors to MPI_Send first and MPI_Recv second, while for the even-numbered processors it is MPI_Recv first and MPI_Send second. The periodic boundary condition is handled transparently for simplification; the processor handling the top outlet boundary of the data volume simply exchanges data with the processor

handling the bottom inlet boundary of the data volume. Finally, the global communication function, MPI_Allreduce, gathers the residuals from the processors and broadcasts the result of the convergence check.

7. Numerical results and discussions

7.1 Computational environments

To test the proposed domain decomposition method and the parallel codes, 3D square duct flows with the periodic boundary condition are simulated and compared with the analytical solutions. Although several improved boundary methods have been suggested to solve the non-slip boundary in LBM in the literatures (Zou and He, 1997; Noble *et al.*, 1995; Maier *et al.*, 1996; Inamuro *et al.*, 1995), they are not easy to extend to 3D complicated geometries. Hence, the simple bounce-back method was employed in this work. Here, we do not take into account the I/O and the serial initialization time, which consumes only a negligible fraction of a typical simulation in all the timing tests.

The analytical solution for the flow in an infinitely long rectangular duct, $-a \leq y \leq a, \quad -b \leq z \leq b$ with x being the flow direction, is given by (White, 1974):

$$u(y,z) = \frac{16a^2}{\mu\pi^3}\left(-\frac{\mathrm{d}p}{\mathrm{d}x}\right)\sum_{i=1,3,5,\ldots}^{\infty}(-1)^{(i-1)/2}\left[1-\frac{\cosh(i\pi z/2a)}{\cosh(i\pi b/2a)}\right]\frac{\cos(i\pi y/2a)}{i^3} \tag{8}$$

The maximum velocity is at $y = 0$ and $z = 0$ and given by:

$$u_{\max}(y,z) = \frac{16a^2}{\mu\pi^3}\left(-\frac{\mathrm{d}p}{\mathrm{d}x}\right)\sum_{i=1,3,5,\ldots}^{\infty}(-1)^{(i-1)/2}\left[1-\frac{1}{\cosh(i\pi b/2a)}\right]\frac{1}{i^3} \tag{9}$$

The velocity can be normalized by the maximum velocity, giving:

$$\frac{u(y,z)}{u_{\max}(y,z)} = \frac{\displaystyle\sum_{i=1,3,5,\ldots}^{\infty}(-1)^{(i-1)/2}\left[1-(\cosh(i\pi z/2a)/\cosh(i\pi b/2a))\right]\left[\cos(i\pi y/2a)/i^3\right]}{\displaystyle\sum_{i=1,3,5,\ldots}^{\infty}(-1)^{(i-1)/2}\left[1-(1/\cosh(i\pi b/2a))\right]\left[1/i^3\right]}, \tag{10}$$

where p represents the pressure, and a and b are the width and height of the cross section of the rectangular duct, respectively.

Thus, the non-dimensional velocity, $u/u_{\max}$, is independent of the viscosity and the pressure gradient. Three different values of τ, 0.9, 1.0, and 1.2, have been used in the simulations. Because the similarity of the results, only the results with $\tau = 1.0$ are presented here. The criterion to determine that the steady state has reached is:

$$\sum_{i=1}\frac{|u(x_i,t+1)-u(x_i,t)|}{|u(x_i,t+1)|} \leq 10^{-10} \tag{11}$$

where the summation is over all the fluid cells. The code usually takes a few thousand iterations to reach the steady state, depending on the viscosity and the boundary conditions.

7.2 Parallel performance

Two important factors are used to evaluate the efficiency of the parallelization: workload balance and communication workload balance. The former is defined as $\max\langle N\rangle/\min\langle N\rangle$, where $\max\langle N\rangle$ is for the subdomain with the maximum number of computational cells in all the subdomains and $\min\langle N\rangle$ for the subdomain with the minimum number of computational cells. In the proposed method of the domain decomposition, the maximum number of active cells for the $\max\langle N\rangle$ subdomain is one cell more than those for the $\min\langle N\rangle$ subdomain, so the workload balance factor is $(N+1)/N$, i.e. $1+1/N$. The latter is defined as $\max\langle N_c\rangle/\min\langle N_c\rangle$, where $\max\langle N_c\rangle$ is the subdomain with the maximum number of communication cells in all the subdomains, and $\min\langle N_c\rangle$ is the one with the minimum number of the communication cells. Likewise, we can evaluate the communication workload factor using cell numbers on an interface. It should be pointed out that the boundary cells might be one extra column (or row), as shown in Figure 3(b) more than those shown in Figure 3(a) because the diagonal neighbour cells need to be communicated. For the present D3Q19 lattice, these extra diagonal boundary cells have been labeled with B^* in Figure 3(b). Because there is the extra row which requires to communicate for the 3D folded interface, the maximum communication number therefore needs to add the cell number of the extra row, N_z. The communication workload factor is $(N_y \times N_z + N_z)/\ (N_y \times N_z) = 1 + (1/N_y)$. If the computational domain is large enough, or when N and N_y are larger, both the workload balance and the communication workload factors approach to one. Thus, the work and communication load imbalances are negligibly small.

The analysis of the parallel performance is measured using speedup S and the efficiency E (Kumar *et al.*, 1994). We define the efficiency as the ratio of the time taken for the program to run on a single processor to the product of the number of processors and the time for taken by the parallel program using p processors:

$$S = \frac{T_s}{T_n}, \quad E = \frac{T_s}{pT_n} \tag{12}$$

where p is the number of processors, T_n is the execution time for the parallelized algorithm using p processors, and T_s is the execution time for the serial algorithm using the a single processor.

Figure 5 shows the total execution time taken for the program to run using different number of processors for the ducts with three different sizes. For a given duct size, the total run time decreases nonlinearly as the number of processors increases. Figure 6 shows the speedup, in which the values of the speedups have been normalized using the value of the single processor. The speedup tends to become saturated, and the curve flattens. It is also apparent that the speedup increases as the size of computation increases. The speedup saturation delays as the size of the computation domain increases. These data agree well with Amdahl's law. Figure 7 shows that the efficiency, E, changes as the number of processors changes for three ducts with different sizes. For the largest duct, the efficiency drops as the number of processors increasing. However, for the smaller duct, $250 \times 30 \times 30$, the efficiency oscillates around one. This is because of the effect of the system architecture on parallel performance. In Origin architectures, two CPUs share one memory path, thus the memory bandwidth limits the parallel performance of LBM. A linear scaling is not expected when least number of processors are employed. Furthermore, a parallel efficiency larger than

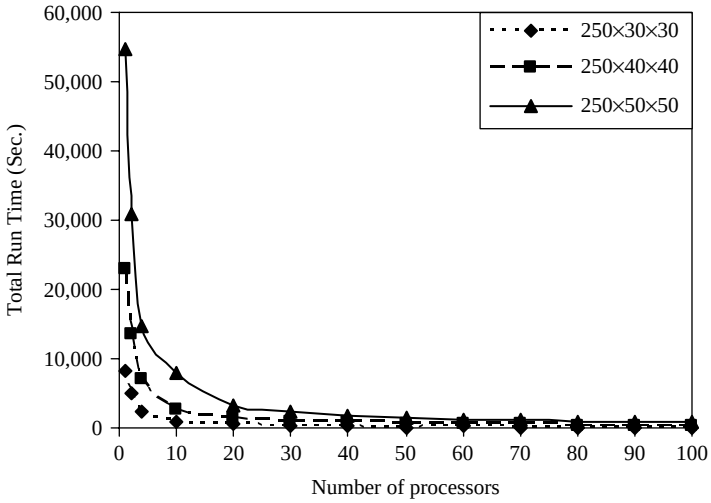

Figure 5.
Total run time versus the number of processors for different computational sizes

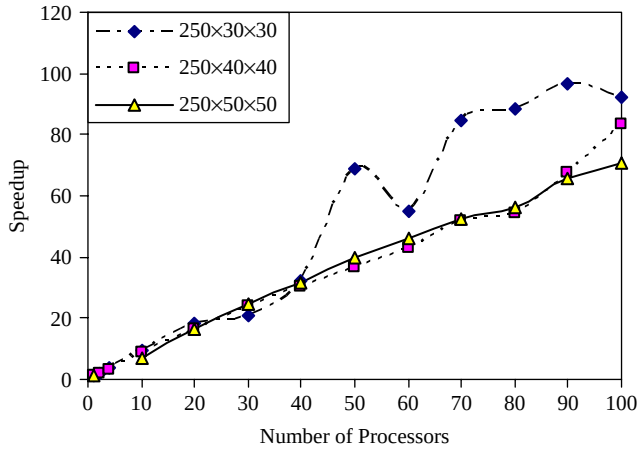

Figure 6.
Scale-up and speed-up measurements for different domain sizes on the origin 2000

one is possible because of the cache effect for the smallest size (250 × 30 × 30). The readers who are interested in the effect of system architectures can refer to Pohl *et al.* (2004).

7.3 Velocity profile comparison between the numerical and analytical results

Figure 8 shows a comparison between the numerical and analytical velocity profiles, $u/u_{\max}$. The measurements were taken at $x = 100$. It can be seen that the errors reduce as the mesh is made refiner. The errors are greater near the walls than in the centre. This is because of the effect of slip velocities at the wall on macro velocities when the bounce-back boundary condition is used (He *et al.*, 1997). Hence, the errors are sensitive to the number of the mesh cells near the walls. Further simulations have not shown differences between the results of serial code and those of parallel code for the same model and algorithm. The domain decomposition does not cause additional errors, and both the results of serial and parallel codes are within machine accuracy.

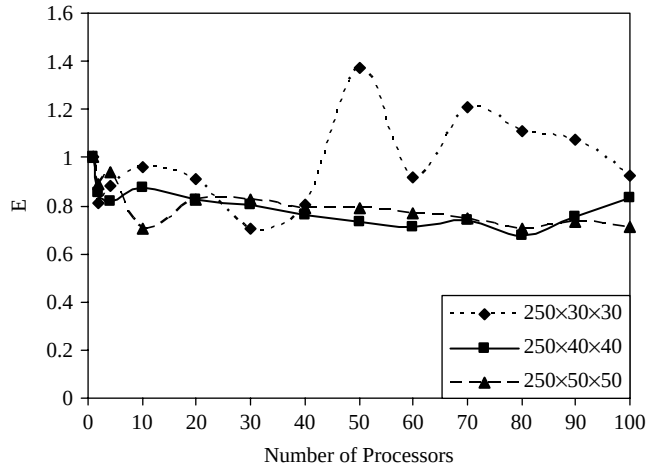

Figure 7.
The parallel efficiency, *E*, versus the number of processors for different computational grid sizes

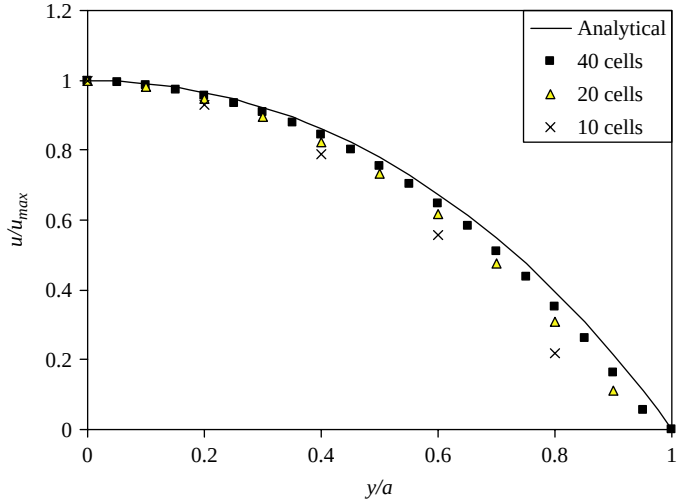

Figure 8.
Comparison of the normalized numerical and analytical velocity profiles (u/u_{max}) for the square duct flows

We then used the model to simulate flow in a column packed with glass beads as shown in Figure 9. Figure 10 shows a cross-section of the simulated 3D flow field in the porous medium. The fluid flows into the domain from the left-hand side and leave the column from the outlet on the right-hand side. The porosity of the column was 0.37. Prescribed pressures were applied to the left and the right sides of the column, and the other four sides were treated as periodic boundaries. The permeability was calculated from the simulation when the flow was deemed to have reached steady state. We also drive the fluid flowing in other two directions to calculate the permeability. The averaged permeability over three directions is $7.4 \times 10^{-9}\,m^2$. This agrees well with experimental result (Nakashima and Watanabe, 2002). The speedup is also tested for the porous medium, as shown in Figure 11. This is in good agreement with those shown in Figure 5.

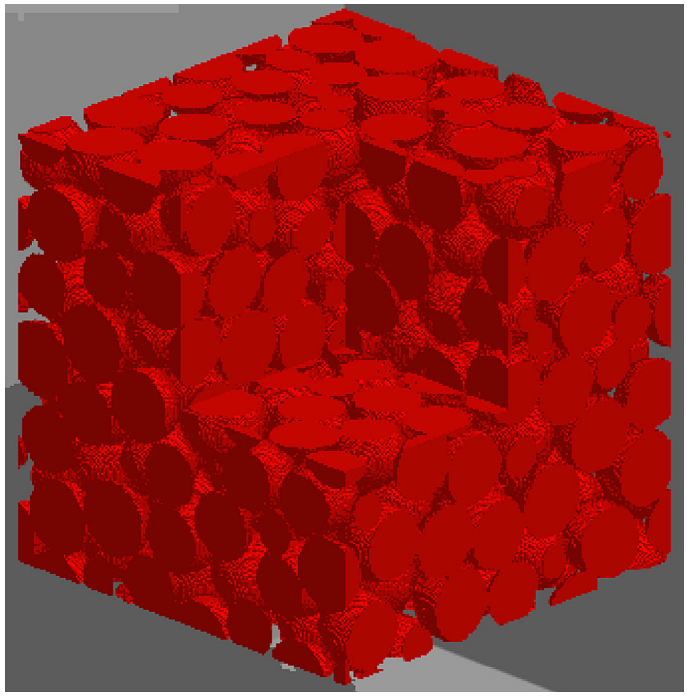

Figure 9.
Geometry of a porous medium packed with glass beads with the average bead diameter 2.11 mm

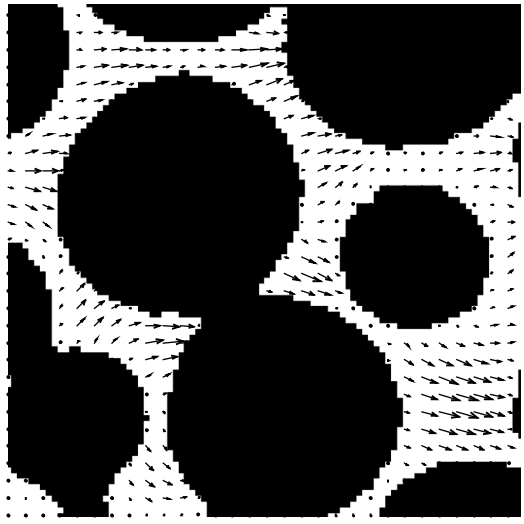

Figure 10.
A cross section of the simulated 3D flow field in the porous media

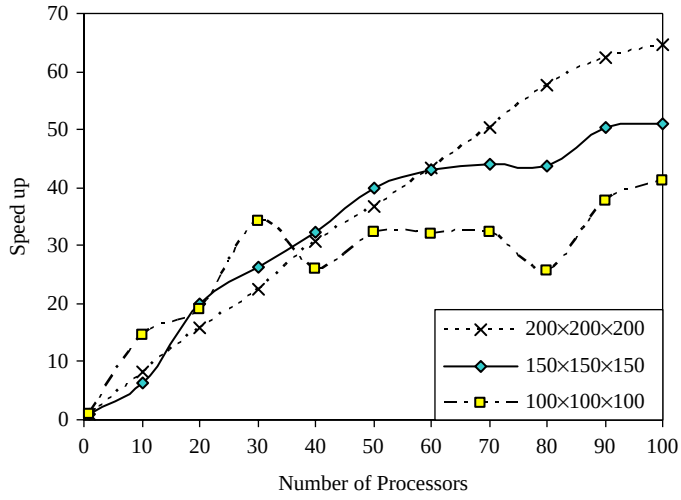

Figure 11.
Speedup measurements for different domain sizes of the porous medium on the origin 2000

The reader refers to reference Wang *et al.* (2005), for further details of the application of the proposed method to porous media.

8. Summary

We carried out a domain decomposition of simple rectangular ducts using a cell-based method. This method can easily recover the interfaces between subdomains. Furthermore, the numerical solution obtained using a combination of the proposed domain-decomposition method and LBM was compared with the exact solutions of the velocity profiles in the rectangular ducts. The simulation has shown that the parallelization of LBM codes does not cause additional errors. Moreover, the simulations do not show any difference between the results of parallel code and those of serial code for the same LBM models. Unlike the existing methods, the cell-based method performs the load balance first based on the total number of fluid cells and then decomposes the domain into a number of groups (or subdomains). Each subdomain has almost the same fluid cells in that the difference of fluid cells in each subdomain is either zero or one, depending on if the total number of fluid cells is a multiple of the processor numbers. The interfaces between the subdomains are recovered at last. Thus, the task of the cell-based method is to recover the interface rather than to balance the load as in the traditional methods. Once the total process of decomposition is completed, no further iteration is needed.

The proposed domain decomposition is not based on equal subvolumes but on equal numbers of active cells. The variables of the simulation are stored in ordered 1D arrays to replace the conventional 3D arrays. Thus, the domain decomposition of complex 3D problems becomes 1D. Finally, the 3D results can be recovered using a coordinate matrix. An important feature of the method is that there is no load imbalance because the number of active cells in each subdomain are equal and the domain decomposition is performed automatically. Furthermore, it keeps the advantage of the slice decomposition so that the dependencies between the processors are simple and the number of interfaces between subdomains is reduced to minimum due to the nearest

connectivity of the lattice on the interfaces to communicate. Another advantage is that the memory is optimum because only the active cells of the computational domain are stored in the memory.

The proposed approach is flexible for geometries as neighbours are defined in an arbitrary manner and the derivation of the algorithm is general for any complex geometries. Despite, it is illustrated with LBM, the method is also suitable for other numerical techniques in fluid dynamics. Considering the difficulties of the domain decomposition of complex geometries and the advantages of the 1D representation, the presented results are encouraging although some extra storage is required for the neighbour definitions.

References

Amati, G., Succi, S. and Piva, R. (1997), "Massively parallel lattice Boltzmann simulation of turbulent channel flow", *Int J. Mod. Phs. C*, Vol. 8, pp. 869-77.

Basermann, D.A., Clinckemaillie, J., Coupez, T., Fingberg, J., Digonnet, H., Ducloux, R., Gratien, J-M., Hartmann, U., Lonsdale, G., Maerten, B., Roose, D. and Walshaw, C. (2000), "Dynamic load-balancing of finite element applications with the DRAMA", *Applied Mathematical Modelling*, Vol. 25, pp. 83-98.

Chen, H., Chen, S. and Matthaeus, W.H. (1992), "Recovery of the Navier-Stokes equations using a lattice-gas Boltzmann method", *Phys. Rev. A*, Vol. 45, pp. R5339-42.

Chen, S. and Doolen, G.D. (1998), "Lattice Boltzmann method for fluid flows", *Annu. Rev. Fluid Mech.*, Vol. 30, pp. 329-64.

Desplat, J.C., Pagonabarraga, I. and Bladon, P. (2001), "LUDWIG: a parallel lattice-Boltzmann code for complex fluids", *Computer Physics Communications*, Vol. 134 No. 3, pp. 273-90.

Frisch, U., Hasslacher, B. and Pomeau, Y. (1986), "Lattice-gas automata for the Navier-Stokes equation", *Phys. Rev. Lett.*, Vol. 56 No. 14, pp. 1505-8.

Giovanni, P., Massaioli, F. and Succi, S. (1994), "High-resolution lattice Boltzmann computing on the IBM SP1 scalable parallel computer", *Computers in Physics*, Vol. 8 No. 6, pp. 705-11.

He, X., Zou, Q., Luo, L-S. and Dembo, M. (1997), "Analytic solutions of simple flows and analysis of non-slip boundary condition for the lattice Boltzmann BGK model", *Journal of Statistical Physics*, Vol. 87 Nos 1/2, pp. 115-36.

Inamuro, T., Yoshino, M. and Ogino, F. (1995), "A non-slip boundary condition for lattice Boltzmann simulations", *Physics of Fluids*, Vol. 7, pp. 2928-30.

Kandhai, D., Koponen, A., Hoekstra, A.G., Kataja, M., Timonen, J. and Sloot, P.M.A. (1998), "Lattice-Boltzmann hydrodynamics on parallel systems", *Computer Physics Communications*, Vol. 111 Nos 1/3, pp. 14-26.

Karypis, G. and Kumar, V. (1998a), "A fast and high quality multilevel scheme for partitioning irregular graphs", *SIAM Journal on Scientific Computing*, Vol. 20 No. 1, pp. 359-92.

Karypis, G. and Kumar, V. (1998b), "Parallel multilevel k-way partitioning scheme for irregular graphs", *Journal of Parallel and Distributed Computing*, Vol. 48, pp. 96-129.

Kumar, V., Grama, A., Gupta, A. and Karypis, G. (1994), *Introduction to Parallel Computing: Design and Analysis of Algorithms*, The Benjamin/Cummings Publishing Company, Inc., Redwood City.

Maier, R.S., Bernard, R.S. and Grunau, D.W. (1996), "Boundary conditions for the lattice Boltzmann method", *Physics of Fluids*, Vol. 8 No. 7, pp. 1788-801.

Martys, N.S. and Hagedorn, J.G. (2002), "Multiscale modeling of fluid transport in heterogeneous materials using discrete Boltzmann methods", *Materials and Structures*, Vol. 35 No. 254, pp. 650-8.

M.P.I. Forum (1994), "MPI: a message-passing interface standard", *Int. J. of supercomputing applications*, Vol. 8 Nos 3/4, available at: www.mpi-forum.org/docs/mpi-11-html/mpi-report.html

Nakashima, Y. and Watanabe, Y. (2002), "Estimate of transport properties of porous media by microfocus X-ray computed tomograpgy and random walk simulation", *Water Resources Research*, Vol. 38 No. 12, p. 1272.

Noble, D.R., Chen, S., Georgiadis, J.G. and Buckius, R.O. (1995), "A consistent hydrodynamic boundary condition for the lattice Boltzmann method", *Physics of Fluids*, Vol. 7 No. 1, pp. 203-9.

Pan, C., Prins, J.F. and Miller, C.T. (2004), "A high-performance lattice Boltzmann implementation to model flow in porous media", *Computer Physics Communications*, Vol. 158 No. 2, pp. 89-105.

Pohl, T., Deserno, F., Thurey, N., Rude, U., Lammers, P., Wellein, G. and Zeiser, T. (2004), *Performance Evaluation of Parallel Large-Scale Lattice Boltzmann Applications on Three Supercomputing Architectures*, SC2004 High Performance Computing, Networking and Storage, Pittsburgh, PA.

Shahpar, S. and Lapworth, L. (2003), "PADRAM: parametric design and rapid meshing system for turbomachinery optimisation, GT2003-38698", *Proceedings of ASME Turbo Expo 2003, Power for Land, Sea, and Air, Atlanta, USA*.

Walshaw, C. and Cross, M. (1999), "Dynamic mesh partitioning & load-balancing for parallel computational mechanics codes", *Invited Lecture, Proc. Parallel & Distributed Computing for Computational Mechanics*, *Weimar, Germany*.

Wang, J., Priestman, G.H. and Tippetts, J.R. (2006), "Modelling the strong swirl flow in a complex geometry using unstructured meshes", *Int. J. for Numerical Methods in Heat and Fluid Flow*, Vol. 16 No. 8, pp. 910-26.

Wang, J., Zhang, X., Bengough, A.G. and Crawford, J.W. (2005), "Domain-decomposition method for parallel lattice Boltzmann simulation of incompressible flow in porous media", *Physical Review E*, Vol. 72 No. 7, Art. No. 016706.

White, F.M. (1974), *Viscous Fluid Flow*, McGraw-Hill, New York, NY.

Zhang, X., Bengough, A.G., Deeks, L.K., Crawford, J.W. and Young, I.M. (2002), "A novel three-dimensional lattice Boltzmann model for solute transport in variably saturated porous media", *Water Resources Research*, Vol. 38 No. 9, p. 1167.

Zou, Q. and He, X. (1997), "On pressure and velocity boundary conditions for the lattice Boltzmann BGK model", *Physics of Fluids*, Vol. 9 No. 6, pp. 1591-8.

Zou, Q., Hou, S., Chen, S. and Doolen, G. (1995), "An improved incompressible lattice Boltzmann model for time-independent flows", *J. Stat. Phys.*, Vol. 81, pp. 35-48.

Corresponding author
Junye Wang can be contacted at: j.wang3@lboro.ac.uk