Domain-decomposition method for parallel lattice Boltzmann simulation of incompressible flow in porous media

Junye Wang,[1,3] Xiaoxian Zhang,[2] Anthony G. Bengough,[1] and John W. Crawford[2]
[1]*Scottish Crop Research Institute, Dundee, DD2 5DA, United Kingdom*
[2]*SIMBIOS Centre, School of Computing and Advanced Technologies, The University of Abertay Dundee, Dundee, DD1 1GH, United Kingdom*
[3]*Department of Aeronautical and Automotive Engineering, Loughborough University, Loughborough, Leicestershire, LE11 3TU, United Kingdom*

(Received 26 October 2004; revised manuscript received 9 May 2005; published 13 July 2005)

The lattice Boltzmann method has proven to be a promising method to simulate flow in porous media. Its practical application often relies on parallel computation because of the demand for a large domain and fine grid resolution to adequately resolve pore heterogeneity. The existing domain-decomposition methods for parallel computation usually decompose a domain into a number of subdomains first and then recover the interfaces and perform the load balance. Normally, the interface recovery and the load balance have to be performed iteratively until an acceptable load balance is achieved; this costs time. In this paper we propose a cell-based domain-decomposition method for parallel lattice Boltzmann simulation of flow in porous media. Unlike the existing methods, the cell-based method performs the load balance first to divide the total number of fluid cells into a number of groups (or subdomains), in which the difference of fluid cells in each group is either 0 or 1, 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. The cell-based method is to recover the interfaces rather than the load balance; it does not need iteration and gives an exact load balance. The performance of the proposed method is analyzed and compared using different computer systems; the results indicate that it reaches the theoretical parallel efficiency. The method is then applied to simulate flow in a three-dimensional porous medium obtained with microfocus x-ray computed tomography to calculate the permeability, and the result shows good agreement with the experimental data.

DOI: 10.1103/PhysRevE.72.016706 PACS number(s): 02.70.−c, 46.15.−x, 83.85.Pt, 05.10.−a

I. INTRODUCTION

Modeling flow in porous media is of a growing interest in a wide range of fields including food processing [1,2], material drying [3], soil science [4,5], petroleum engineering [6,7], and chemical engineering [8]. The study of flow in porous media usually considers three scales [9,10]: pore scale, representative elementary volume (REV) scale, and regional scale. In the pore scale, a theoretical description of the mass and momentum balances is based on physical-chemical and thermodynamic understanding of the system, from which a mathematical description of the mass and momentum balances is carried out. The balance equations at the microscopic scale can be upscaled to macroscopic scale through a volume averaging technique. For heterogeneous porous media, averaging the results from the microscopic scale to the macroscopic scale is normally conducted over a REV. Therefore, modeling pore-scale flow in porous media is imperative in understanding the impact of microscopic heterogeneity on the averaging procedure [9].

Because of the difficulty of visualizing and quantifying the complicated opaque three-dimensional morphology of natural porous media, the earlier study of the pore-scale flow is largely based on network models [3,11,12] in which the pore structure is represented by the sites and bonds in a network. The size of each site and bond is determined from a given pore-size distribution, and the flow along each bond is assumed to be a one-dimensional Poiseuille flow. Since the network models highly idealize the complicated three-dimensional pore geometry, they may give results significantly different from real porous media.

The past decades have seen a rapid development in image analysis techniques. These techniques, ranging from magnetic resonance imaging [13] to microfocus x-ray computed tomography [14,15], have been increasingly applied to visualize and quantify the three-dimensional opaque porous media and flow process through them. For example, a high-resolution computed x-ray topography can provide a measurement of the complicated three-dimensional morphology at a length scale as small as a few microns. These, together with the development of computer power and the advancement of computational physics, have made direct simulation of microscopic flow in porous media feasible and a renewed interest in many fields. One method particularly suitable for modeling such flow is the lattice Boltzmann method (LBM) [16–21]. The LBM is a technique of simulating fluid flow based on kinetic theory. By appropriately choosing the equilibrium distribution function at the microscale, the LBM recovers the Navier-Stokes equations at the macroscale. The LBM simulates fluid flow by calculating the collisions and interactions between the particles moving on a regular lattice in phase space, rather than directly solving the Navier-Stokes (NS) equations. It is a direct simulation in which the macroscopic variables such as density and velocity are evaluated by tracking the dynamics of the particles. One of the simplest LBM is the lattice Bhatnagar-Gross-Krook method [22,23]. Because of its simplicity in handling complicated boundaries, the LBM has proven efficient to simulate pore-scale flow in porous media [24–30]. The standard LBM gives errors in modeling incompressible flow as it actually recovers a compressible flow where the pressure is proportional to fluid density. Effort has been made to reduce such errors [31–34].

Natural porous media are heterogeneous with a wide pore-size distribution. In studying their macroscopic properties such as permeability via pore-scale flow simulation, a large domain and fine grid resolution are required in order to adequately resolve the pore heterogeneity. This will require large computer resources and ultimately rely on parallel computation in which memory optimization and effective domain decomposition are essential.

A porous medium is characterized by its consisting of pore and solid. In modeling flow through it, the easy way is to store all the pores and solids. This is straightforward but wastes core memory. Recently, Martys and Hagedorn [27], Pan *et al.* [35], and Schulz *et al.* [36] used a sparse lattice representation to store only the fluid cells. It is known that uniformly decomposing a heterogeneous domain like a porous medium for parallel computation is difficult, particularly in the use of the sparse lattice representation where the cells lose their natural ordering, making neighboring cells unknown as a result. The existing domain-decomposition methods usually divide a domain into a number of approximately equal subdomains using regular partitioning [37–39] techniques. For example, Kandhai *et al.* [40] and Pan *et al.* [35] use the recursive coordinate bisection method to partition a box in which the grids are decomposed into a number of partitions in orthogonal directions; Schulz *et al.* [36] decompose a domain using the METIS package [41]. METIS is based on the multilevel *k*-way partitioning. The recursive bisection (RB) scheme and its variants are commonly used as the initial *k*-way partitioning in the multilevel *k*-way partitioning. In essence, it is difficult to decompose a highly heterogeneous domain like a porous medium into a number of same-size subdomains using RB [40–42]. Moreover, the multilevel partitioning scheme is computationally expensive due to the cost of computing the three-dimensional interfaces between subdomains to balance the load. Therefore, these methods have the following drawbacks to decompose a porous medium: waste of memory, load imbalance, complex communication pattern, lack of nearest subdomain communication, expensive implementation, and inflexibility in dealing with complicated geometry.

The work reported in this paper aims to propose a domain-decomposition method for parallel LBM simulation of fluid flow in porous media. Unlike the existing methods, the proposed method is cell based. Combined with the use of a sparse matrix to store only the fluid cells, the method divides the total number of fluid cells evenly into a number of groups and then assigns them to each processor. The difference of cell numbers in each processor is either 0 or 1, depending on if the total number fluid cells is a multiple of the number of processors. This not only saves core memory but also results in exact load balance, improving computational efficiency as a result. This paper is organized as follows: Section II presents a modified LBM model for incompressible flow; Sec. II gives the use of a sparse matrix to optimize memory for parallel computation of flow in porous media; Sec. IV reviews various domain-decomposition methods; Sec. V introduces the proposed cell-based method and its combination with the sparse matrix to solve flow in porous media; Sec. VI shows the performance of the proposed method using different computer systems, its comparison with existing methods, and application to simulate flow in a three-dimensional porous medium obtained using microfocus x-ray computed tomography.

II. LATTICE BOLTZMANN METHOD FOR INCOMPRESSIBLE FLOW

The standard LBM has been employed to simulate flow in porous media [24–29], and its error in approximating incompressible flow has been analyzed [31–34]. The following lattice Boltzmann method [31,33] is used in this paper:

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

where $f_i(x,t)$ is the particle distribution function at position x and time t in the direction of the velocity ξ_i, $f_i^{eq}(x,t)$ is the value of $f_i(x,t)$ in equilibrium—i.e., the equilibrium distribution function at position x and time t—and τ is a single relaxation time controlling the rate of $f_i(x,t)$ approaching $f_i^{eq}(x,t)$. The equilibrium distribution function is given by

$$f_i^{eq}(x,t) = w_i\left[\rho + 3(\boldsymbol{\xi}_i\cdot\boldsymbol{u}) + \frac{9}{2}(\boldsymbol{\xi}_i\cdot\boldsymbol{u})^2 - \frac{3}{2}u^2\right], \quad (2)$$

where ρ is fluid density, ξ_i is the ith particle velocity, and $\boldsymbol{u}$ is the macroscopic momentum. In this work, the nineteen speed model in three dimensions is used in which the 19 particle velocities are $(0,0,0)$, $(\pm\delta/\Delta t, \pm\delta/\Delta t, 0)$, $(\pm\delta/\Delta t, 0, \pm\delta/\Delta t)$, $(0, \pm\delta/\Delta t, \pm\delta/\Delta t)$, $(0,0,\pm\delta/\Delta t)$, $(0,\pm\delta/\Delta t,0)$, and $(0,0,\pm\delta/\Delta t,)$. The corresponding weighting factors are [5]

$$w_i = \begin{cases} \frac{1}{3}, & |\xi_i| = 0, \\ \frac{1}{18}, & |\xi_i| = \delta/\Delta t, \\ \frac{1}{36}, & |\xi_i| = \sqrt{2}\delta/\Delta t, \end{cases} \quad (3)$$

where δ is the side length of the cubic cells and Δt is time step. The fluid density and macroscopic momentum in Eq. (2) are calculated from

$$\rho = \sum_{i=0}^{18} f_i, \quad (4)$$

$$\boldsymbol{u} = \sum_{i=0}^{18} \xi_i f_i. \tag{5}$$

Applying the Chapman-Enskog multiscale expansion to Eqs. (1)–(5) gives the incompressible Navier-Stokes equations in the steady case as follows:

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

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

where $c=\delta/\Delta t$ is the sound speed and $\upsilon=(2\tau-1)/6$ is the kinematic viscosity. Excluding the high-order error $O(\delta^2)$, Eqs. (6) and (7) are identical to the stationary incompressible NS equations, in which the compressibility error is induced because of the use of the modified equilibrium function given in Eq. (2).

III. MEMORY OPTIMIZATION

The workload distribution in traditional computational fluid dynamics and the LBM increases as the number of fluid cells increases. It is easy to store and compute the whole lattice [37–39]. This is straightforward for parallel implementation but wastes core memory when applied to a porous domain because the solid cells are not involved in the calculation.

For a structure like a porous medium that consists of pore and solid obstacles, it is efficient to store only the fluid cells because the locations of the fluid cells are explicitly known. Martys and Hagedorn [27], Zhang *et al.* [4,5], Pan *et al.* [35], and Schulz *et al.* [36] have developed such an approach to simulate flow in porous media. In their approach, the computation is performed only for fluid cells. For a porous medium with low porosity, the core memory saved from this can be substantial. For example, for a three-dimensional domain consisting of N cells, the memory required, if storing the whole lattice, for single-phase flow is

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

where the first term represents the 19 distribution functions at the current and next time steps and the second term represents the fluid density and the three velocity components. Assuming that a float variable needs 8-byte memory, the total memory required by such storage is $336N$ bytes. While for the same problem, if only storing the fluid cells, the total memory required is

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

where ε is the porosity, the first two terms are the same as above, the third term represents the indices to identify the x, y, and z coordinates of a cell, and the fourth term stores the information to identify a cell's 18 neighbors. Assuming that each integer needs a 2-byte memory, the total required memory for $\varepsilon=0.3$ is $113.4N$ bytes; that is, the proposed method saves approximately $2/3$ memory compared to the conventional method.

IV. DOMAIN DECOMPOSITION: BACKGROUND

Two commonly used approaches to decompose a domain [40] are slice, box, and cube scheme and recursive bisection techniques.

A. Slice, box, and cube partitioning scheme

The traditional slice, box, and cube domain decompositions are based on equal-subdomain partitioning techniques [37–39,42]. The slice partitioning in one dimension divides the grids into slices corresponding to processor numbers, so that each slice has the same volume of meshes. The box partitioning in two dimensions uses two orthogonal one-dimensional slices to partition the domain: first, the volume is decomposed into vertical slices, then they are sliced horizontally, and finally the two partitions are overlapped and the resulting partitioned boxes (whose number is equal to vertical partitions $\times$ horizontal partitions) are assigned to the processors. Similarly, the three-dimensional cube partitioning uses three orthogonal one-dimensional slices to partition a domain. Although these traditional approaches can get a good load balance and communication locality for regular geometries, it is difficult for them to decompose a complex geometry with a good load balance.

B. Recursive bisection techniques

The recursive coordinate bisection (RCB) technique is the most intuitive recursive bisection technique. It recursively decomposes a domain into two parts of equal size along the orthogonal coordinate directions at each step until k partitions are generated. Because RCB does not use connectivity information, the partitions may be disconnected and have many grid-edges across the partition boundaries. This problem can be overcome by recursive bisections, such as recursive graph bisection (RGB) and recursive spectral bisection (RSB).

Domain decomposition techniques such as multilevel k-way partitioning [43,44], dynamic mesh partitioning schemes [45,46], and space-filling curves (SFC's) [47] have been developed and been used in packages such as METIS [41] and DRAMA [46]. However, it should be mentioned that both multilevel k-way and dynamic mesh partitioning are based on RCB, RGB, or RSB. For example, the multilevel graph partitioning consists of three phases: coarsening phase, initial partitioning phase, and refinement phase. The initial partitioning phase of the coarsest graph based on RB is crucial in the overall partitioning paradigm as the follow-up refinement has only limited capability to improve the quality of the initial partitioning. In the initial partitioning phase, a two-way partitioning is firstly obtained for a k-way partitioning, and then another two-way partitioning of each resulting partition is recursively obtained. After $\log k$ phases, the domain is partitioned into k partitions. Thus the problem of performing a k-way partitioning is reduced to performing a sequence of bisections. The initial partitioning phase is to minimize the edge cut and to make the size of the subdomains roughly the same. Since the RB techniques are unable to simultaneously obtain exact load balance, regular communication pattern, and the nearest communication connection in the coarsening and the initial k-way partitioning phases, a follow-up refinement has to be performed to balance the load after the initial partitioning. In the refinement phase, the number of cells in each subdomain is counted. Because the cell numbers in each subdomain are not the same, some cells have to be moved among the subdomains to balance the load. Generally, it is impossible to reach the exact load balance by moving the cells only once; hence, resorting and researching iterations are needed, costing more CPU time and giving rise to an irregular communication pattern and no-nearest communication connection. Furthermore, the multilevel partitioning may result in many different results in coarsening, initial partitioning, and refinement; it thus needs an optimization algorithm to find the best one. The advanced multilevel partitioning schemes can handle multiple-constraint and multiple-objective problems, but they are still difficult to use in problems such as highly heterogeneous porous media [40,41]. Methods including scattered decomposition (modular mapping) have been developed to handle highly irregular domains, but the load balance in them is achieved at the expense of increasing the overhead communication between the partitions that are not in the same processor.

V. CELL-BASED DECOMPOSITION METHOD

A. Decomposition

To overcome the inefficiency of the existing methods in decomposing heterogeneous porous media, we propose a cell-based domain-decomposition method. The existing methods usually decompose a domain first, then recover the interfaces, and load balance are performed. The interface is recovered to check load balance again. The recovery of the interfaces and the load balance is performed iteratively. 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 having almost the same fluid cells in that the difference of fluid cells in each subdomain is either 0 or 1, 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 interfaces rather than the load balance as in the traditional methods. An example of a three-dimensional porous medium is used to show how to decompose by using the cell-based method. In this example, the porous medium is scanned in first from the z direction, then from the y direction, and finally from the x direction. In the decomposition, the solid cells are skipped, and only the pore cells are counted and stored in an ordered one-dimensional array. The decomposition of the three-dimensional porous domains turns out to decompose the ordered one-dimensional array into a number of groups based on the number of processors. In the one-dimensional array, neighboring cells are no longer known and have to be directed explicitly using extra neighbor arrays. The spatial location of each fluid cell in the one-dimensional array is orderly stored in coordinate matrices. Thus, the coordinates of any fluid cell can be recovered based on the number of the cell using the coordinate matrices. Notice that the total number of the fluid cells may not be a multiple of the number of processors. In this case, the remainders are distributed further to former processors. Therefore, the difference of cell numbers in all the processors is either 0 or 1, an exact load balance.

B. Recovering the interfaces

The fluid domain is split into a number of subdomains, each being handled by a processor. Because the fluid cells are stored orderly and connected in a one-dimensional array, every cell has an ordered number and its coordinate is stored in coordinate arrays. Hence, every cell can be easily recovered through its ordered number and coordinate arrays.

The recovery of the interface between subdomains is to recover the coordinates of the first and last cells of the subdomain using the coordinate arrays. From the coordinate of the first cell in the subdomain we can recover the first boundary of the subdomain, and from the coordinate of the last cell we can recover the second boundary of the subdomain. The coordinates of the first and last cells in each subdomain have two situations: on the corners and not on the corners. For the former situation we take the first cell of the subdomain as an example. Here we assume that the domain is decomposed along the x axis. If the first cell is on the corner, it has coordinates $(x_1,0,0)$. All cells on the cross section, $x=x_1$, consist of a boundary of the subdomain. The last cell on the boundary has a coordinate (x,N_y-1,N_z-1), where N_y and N_z are the number of cells in the y and z directions, respectively, and all the fluid cells on this boundary have coordinates between $(x,0,0)$ and (x,N_y-1,N_z-1). It is easy to count the number of fluid cells on the boundary from the coordinate $(x,0,0)$ to (x,N_y-1,N_z-1). The number of fluid cells on the boundary is all the fluid cells to be communicated with their neighbor. Since these boundary cells are also stored orderly and connected in the memory, the memory positions of the boundary cells to be communicated can be traced through the ordered number of the first cell and the number of fluid cells on the boundary. Thus, the recovery real interface is a straight line in two dimensions as shown in Fig. 1(a), and it is a plane in three dimensions as shown in Fig. 2(a). In Figs. 1 and 2, the fluid cells are marked by white and solid cells by gray. The boundary cells labeled B_0 belong to subdomain PE00 and those by B_1 belong to subdomain PE01.

If the first and last cells on a interface are not in the corners but in an arbitrary position, they can still be recovered using the ordered numbers and the coordinate arrays. We still take the first cell in a subdomain distributed along the x axis as an example and assume that the subdomain is distributed to processor I. The ordered number of the first cell of the subdomain is known after decomposition, and its coordinate (x_1,y_1,z_1) can be recovered using the coordinate arrays. The coordinates of two cells opposite the first cell must be on its neighbor surfaces (x_1-1,y_1,z_1) and (x_1+1,y_1,z_1). Thus, all the fluid cells between (x_1,y_1,z_1) and (x_1+1,y_1,z_1) are on the boundary of processor I, and all the fluid cells between (x_1-1,y_1,z_1) and (x_1,y_1,z_1) are on the boundary of processor I-1. The number of the fluid cells on the two boundaries can be counted through the coordinates

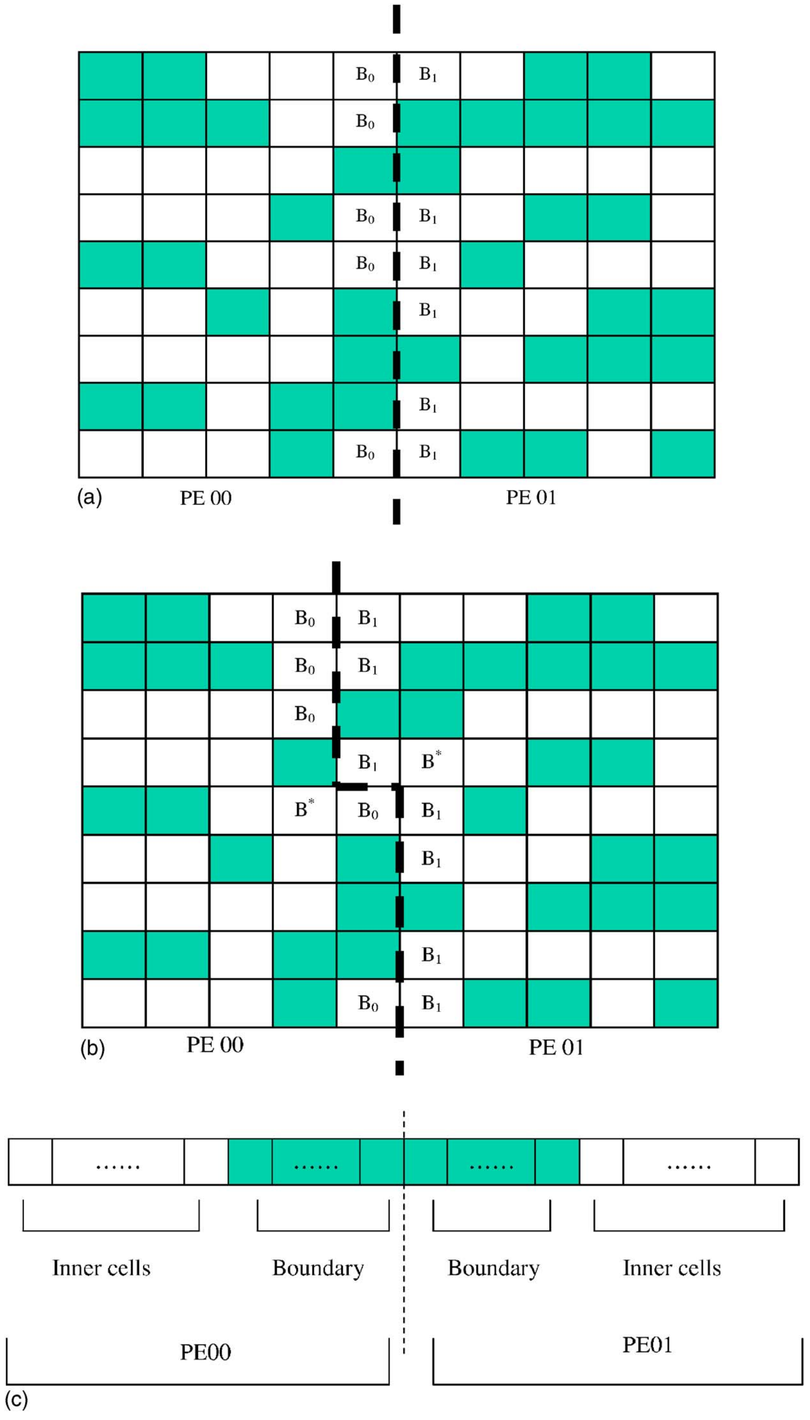


FIG. 1. (Color online) A typical boundary between the subdomains in two dimensions. The fluid cells are marked by white and the solid cells are marked by gray. The boundary cells labeled by B_0 belong to the subdomain PE00 and those by B_1 belong to subdomain PE01. (a) The interface when starting and ending cells are at the corners, (b) the interface when starting and ending cells are not at the corners, and (c) a typical pattern of a data array.

of the first cell and its two opposite cells, respectively. Figure 1(b) shows an irregularly folded line in two dimensions, and Fig. 2(b) shows an irregularly folded surface in three dimensions, in which the interface cells are marked with B, the starting cells with S, and the ending cells with E. It should be noted that the boundary cells might be one cell more than those shown in Fig. 1(a) for two dimensions and one column (or row) more than those shown in Fig. 2(a) for three dimensions because the diagonal neighboring cells need to be connected in the D3Q19 model. These extra diagonal boundary cells have been labeled with B^* in the figures. Figure 1(c) shows a typical pattern of a data array, in which the communication cells between the subdomains are ordered and closely connected. Likewise, we can recover the ending interface of the subdomain. Thus the interface of every subdomain can be recovered using the ordered number of the starting and ending cells and the coordinate arrays. It should be mentioned that because all fluid cells are stored orderly and connected in the memory and the data arrays are distributed orderly on the processors, the boundary cells to be communicated between two subdomains are also connected and nearest as shown in Fig. 1(c). Hence, data communication occurs only in the two nearest processors and nearest data memory. In other words, there is a nearest communication

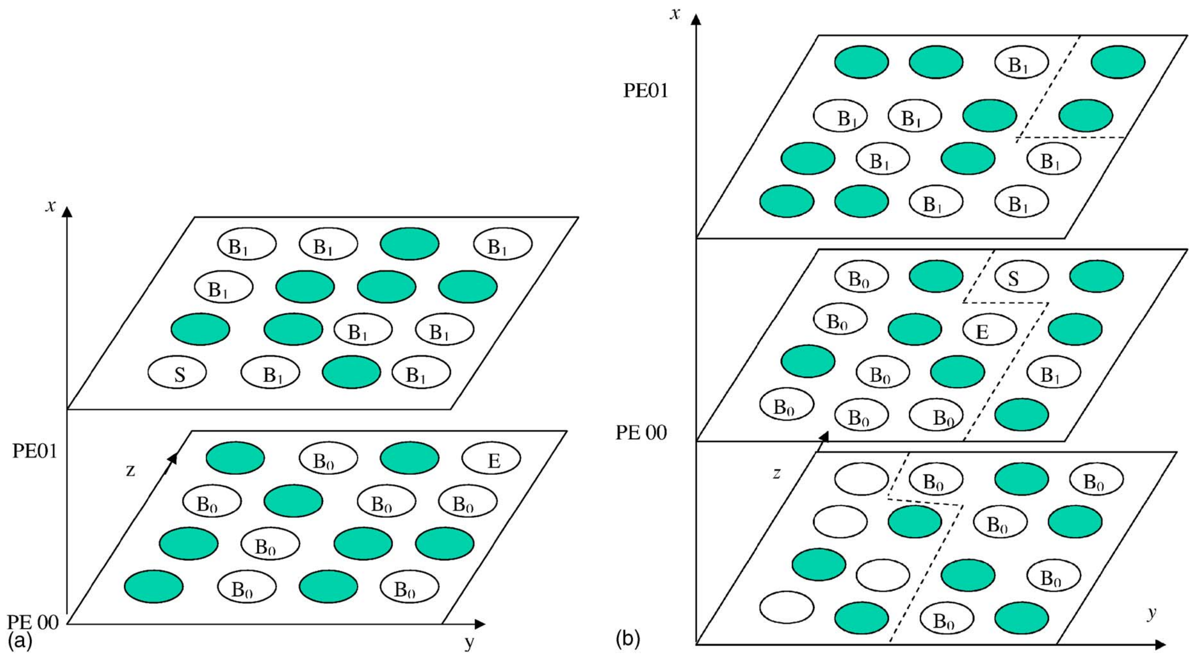


FIG. 2. (Color online) A typical boundary between the subdomains in three dimensions. The fluid cells are marked by white and the solid cells are marked by gray. The boundary cells labeled B_0 belong to subdomain PE00 and those by B_1 belong to subdomain PE01. (a) The interface when the starting and ending cells are at the corners and (b) the interface when the starting and ending cells are not at the corners.

connection. Furthermore, because the boundary cells from the first to the last cells on an interface are still stored orderly in the one-dimensional array and exchange data with one of the nearest neighbors, all fluid cells to be communicated are picked up once and no pointer jumps occur to pick up data in memory. Hence, the communication pattern is regular.

Another advantage of the proposed method is its suitability for arbitrary geometries as the neighbors of each cell are defined in an arbitrary manner and the domain decomposition is to decompose a one-dimensional array. Compared with the recursive bisection techniques, the proposed method has the advantages of exact load balance. Since all cells on the interface are ordered and closely connected in the memory, there is a regular communication pattern. Like the slice method—the communication connections occur only in the nearest processor and no other processors need to be communicated. In fact the proposed method keeps the main advantages of the slice method—the regular communication pattern and the nearest communication connection—while overcomes its disadvantage of load imbalance for complex geometries. Furthermore, the proposed method does not need resorting and researching operations. Therefore, it produces a decomposition that is comparable to and in most cases better than those produced by the multilevel partitioning algorithms. Finally, the method is simple and efficient as it does not require extra CPU time to optimize the decomposition of the domain. The core memory required by the proposed method is almost optimal as it only stores, calculates and communicates the fluid cells.

VI. RESULTS AND DISCUSSIONS

A. Computational environment

Numerical experiments were conducted on a Silicon Graphics Origin 2000 machine and a Silicon Graphics Altix 3700 machine. The specifications of the two systems are shown in Table I.

TABLE I. Specifications of the two parallel machines.

Machine	SGI Origin 2000	SGI Altix 3700
Number of processors	128 MIPS R12000	256 Intel Itanium 2
Cache hierarchy	Level 1 data 32 kbytes	Level 1 data 16 kbytes
	Level 1 instruction 32 kbytes	Level 2 256 kbytes
	Level 2 8 Mbytes	Level 3 3 Mbytes
Peak performance	800 megaflops	5.2 gigaflops
Clock speed	400 MHz	1.3 GHz
Global memory	$100=128\times 0.8$ gigaflops	$384=256\times 1.5$ gigaflops
Operating system	Irix	Linux

The parallelization is accomplished using a simple single-program multiple-data (SPMD) model. The data are divided into spatially continuous blocks along the x axis; multiple copies of the same program run simultaneously, each operating using its own block of data. All data depend on the results obtained from previous computations in other processors. At the end of each iteration, the data of the folded surfaces on the boundaries between the blocks are passed within the appropriate processors. By using a ghost layer of lattice cells surrounding the subdomain, the propagation step can be isolated from the data in the exchanging step. Ghost cells are cells that the processor needs to compute all its stencils, but not the ghost cells themselves; the ghost cells are computed in the previous step by the neighboring processors. Prior to the propagation step, the processor must receive the data for the ghost layer from its neighboring processors; the data in the interface are then sent to the ghost layer of the neighboring processors.

The message passing interface (MPI) [48] is used for data transfer. The MPI_Send and the MPI_Recv functions are used to transfer data at the ghost cells between the processors. These blocking functions do not return until the communication is completed. The blocking communications have two drawbacks: deadlocks and parallel inefficiency. To prevent deadlocks happening, we authorize the odd-numbered processors to MPI_Send first and MPI_Recv second, and authorize the even-numbered processors to MPI_Recv first and MPI_Send second when blocking communications are used. To overcome the two drawbacks, nonblocking communications MPI_Isend and MPI_Irecv are also used for comparison. In all the simulations, the periodic boundary is solved transparently in that the processor handling the top boundary simply exchanges data with the processor handling the bottom boundary. The flow is driven by a pressure difference applied to one direction and the prescribed pressure is solved using the method given in [16].

The results are obtained with $\tau=1.0$. The simulations are terminated when flow reaches steady state, and the criterion of checking the flow to have reached steady state is

$$\sum_{i=1} \frac{|\boldsymbol{u}(x_i,t+1)-\boldsymbol{u}(x_i,t)|}{|\boldsymbol{u}(x_i,t+1)|} \leqslant 10^{-10}, \tag{8}$$

where the summation is over all the fluid cells. In all simulations, it usually takes a few thousand iterations to reach the steady state.

B. Parallel performance

The parallel computation is analyzed by a speedup factor S and a efficiency E [42]:

$$S = T_1/T_n, \quad E_n = T_1/(nT_n), \tag{9}$$

where T_1 is the execution time for serial algorithm using a single processor and T_n is the execution time for parallelized algorithm using n processors.

Figures 3(a)–3(c) show the total execution time T_n, the speedup S, and the efficiency E, respectively, as a function of processor numbers using the Origin 2000 for different domain sizes. For a given domain size, the total execution time decreases nonlinearly while the speedup tends to saturate as the number of processors increases; the speedup saturates slowly as the domain size increases. Furthermore, a larger domain of the same problem yields a higher speedup and efficiency using the same number of processors, although both speedup and efficiency continue to decreases as the number of processors increases. This performance is in agreement with the Amdahl's law. The dependence of the efficiency E on processor number also varies with the domain size, increasing for larger domains and oscillating around one for smaller domains. In all the examples, the speedup is less than unity, while the efficiency could be higher than unity because of the better use of cache memory, particularly when using a few processors for a small domain.

Figure 4 shows the results obtained using Altix 3700. The performance of the Altix 3700 is also in a good agreement with the Amdahl's law, particularly for the large domain where speedup and efficiency are similar for the two systems. The difference is for the small domain, where the speedup in the Origin 2000 tends to saturate quicker and is therefore less efficient than the Altix 3700. This is probably due to the different computational and communicational speeds of the two systems. The computational and communicational speeds of the Altix 3700 are faster; the influence of cache memory is thus less substantial.

C. Comparison with existing methods

Parallel lattice Boltzmann simulation of flow in porous media has been reported [35,40,49] using different domain-decomposition methods. The proposed method is compared with them to show the improvement. Kandhai *et al.* [40] propose an improved decomposition approach based on orthogonal recursive bisection (ORB) and demonstrate its superiority. We therefore only show the comparison with ORB. We take the example studied by Kandhai *et al.* [40], which consists of $139\times380\times50\approx2.64\times10^6$ cells, to compare with two of ours examples, comprising 3.0×10^6 and 1.26×10^6 cells, respectively. Table II shows the comparison of the speedup factor, defined as T_4/T_{20}, of the proposed method and ORB. The improvement of the proposed method over ORG in terms of the speed factor is significant for both small and big problems.

Pohl *et al.* [49] used the SGI Altix 3700 to simulate turbulence and metal foams. Here we compare the performance of the proposed method with theirs. The comparison is for the variation of million lattice site updates per second (MLup/s), the (number of time steps)$\times$(number of lattice sites)/(execution time), with processor numbers. Figure 5(a) shows the result for an example consisting of 3.0×10^6 cells. The speedup factor of the proposed method using 64 processors for this problem is 50, slightly slower than the speedup factor of 58 reported in Pohl *et al.* [49] (Fig. 15) for problem with 4.2×10^6 cells. The less speedup of the proposed method for this example is due to the domain size difference rather than the method itself, as it has been shown in previous examples that the speedup factor of the proposed method increases as the domain size increases.

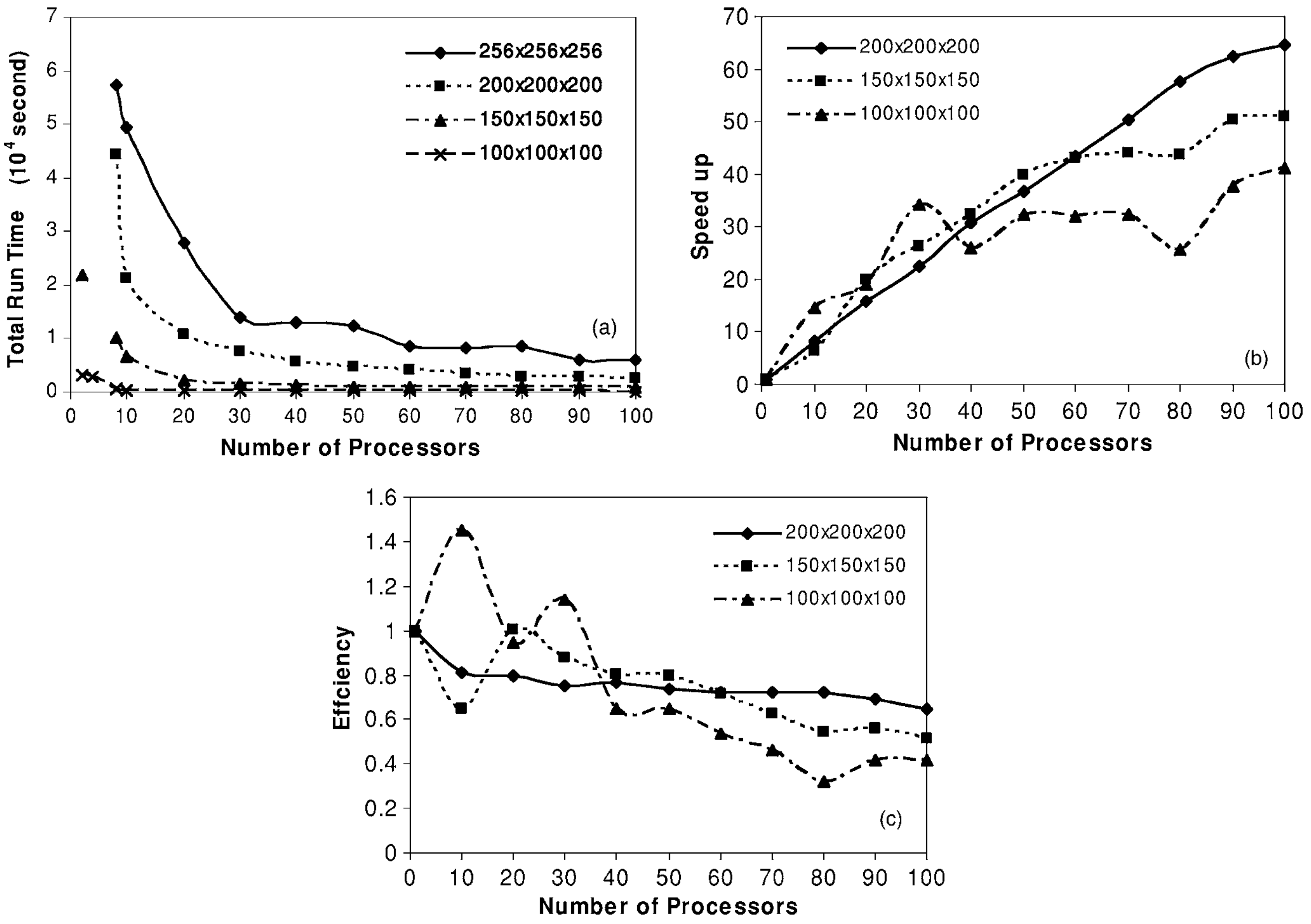


FIG. 3. The performance with SGI Origin 2000. (a) The total execution time, (b) the speedup, and (c) the efficiency.

The performance of the proposed method in SGI origin is compared with that reported in Pan *et al.* [35] in terms of execution time taken by per 300 iterations (time steps). Figure 5(b) shows that the execution time used by the proposed method for the problem with 2.25×10^5 cells decreases from 617 s when using one processor to 44 s when using 16 processors, giving a speedup factor 14. While the corresponding execution time taken by the RB method for the problem with 9.5×10^4 cells (Fig. 11 of Pan *et al.* [35]) is 400 s and 90 s, respectively, giving a speedup factor of 4.6. Furthermore, the proposed method needs less execution time when using 16 processors for the larger domain. Again, the proposed method has less overhead communication and/or better load balance and therefore has high speedup.

D. Comparison with theoretical efficiency

The theoretical efficiency can be estimated from

$$E=\frac{T_{cal}}{T_{cal}+T_{com}}=\left(1+\frac{T_{com}}{T_{cal}}\right)^{-1}, \qquad (10)$$

where T_{com} and T_{cal} are the communicational and computational times, respectively. T_{com} is expected to be proportional to the number of fluid cells in the subdomains, and T_{cal} is proportional to the number of communicating fluid cells. The following relations can be obtained:

$$T_{cal}=\alpha_1\frac{N^3}{nU}, \quad T_{com}=2\alpha_2N^2/V, \qquad (11)$$

where α_1 and α_2 are porosities in three dimensions and two dimensions respectively, U is the computational speed, and V is the communicational speed. Inserting the above equations into Eq. (10) gives

$$E=\left(1+\frac{2\alpha_2nU}{\alpha_1NV}\right)^{-1}. \qquad (12)$$

For a given example, α_1, α_2, U, and V are constant, and E is therefore only dependent on n and N. Figure 6 compares the theoretical E and the measured E from Fig. 3(c) assuming $\alpha_1=\alpha_2$ and $U=2V/3$. They are in good agreement. Equation (12) also indicates that the proposed method is more efficient when the ratio of the computational and communicational time is small. This may explain why the Altix 3700 has a higher efficiency than the Origin 2000 when using less few processors.

E. Flow field and permeability

One purpose of simulating pore-scale flow in porous medium is to calculate its flow field and permeability. The permeability of a porous medium is defined in the Darcy's law as [50]

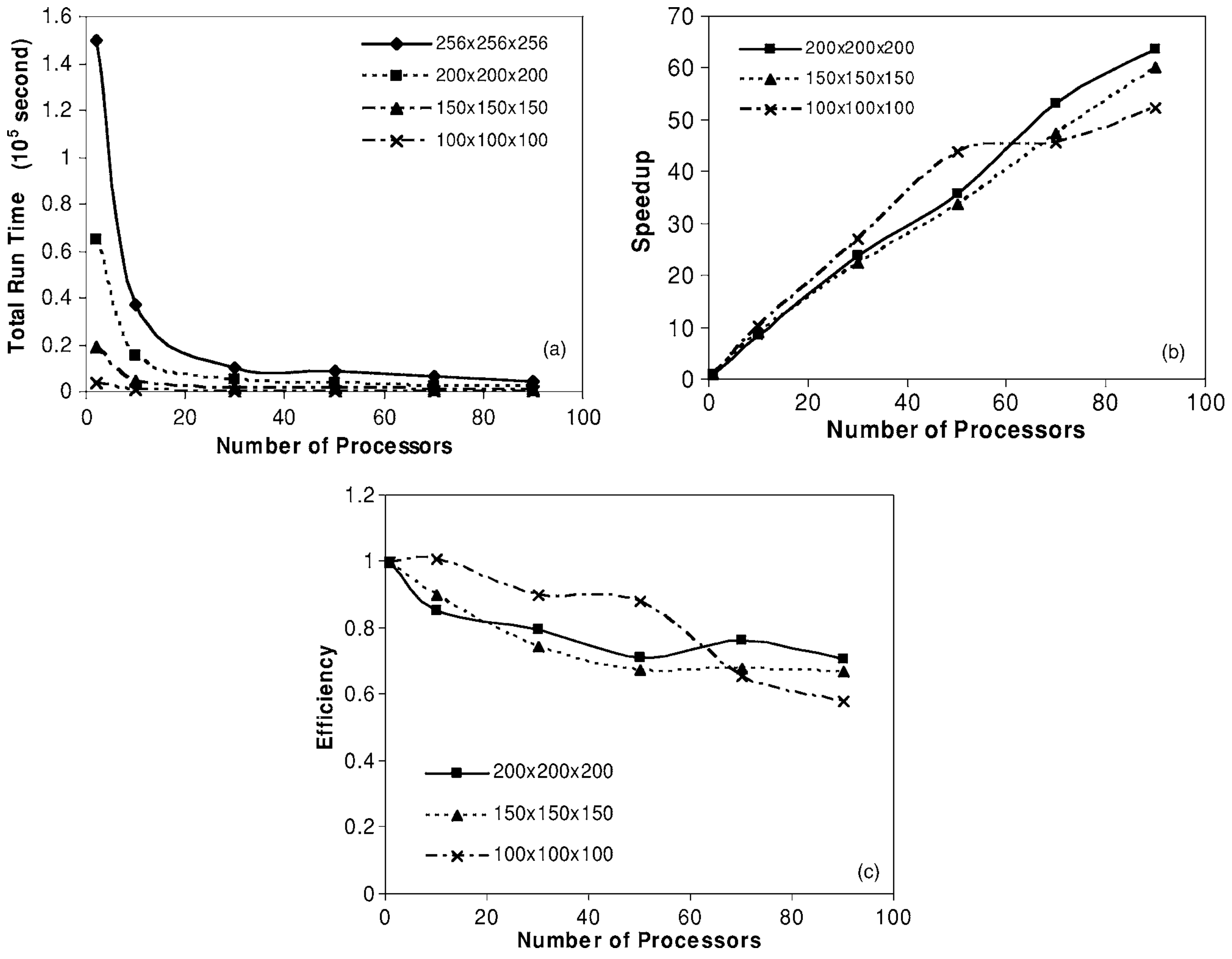


FIG. 4. Performance with SGI Altix 3700. (a) The total execution time, (b) the speedup, and (c) the efficiency.

$$\langle u \rangle = -\frac{K}{\nu}(\nabla P - \rho g), \tag{13}$$

where $\langle u \rangle$, ν, and ρ are the average velocity, kinetic viscosity, and fluid density, respectively, K is the permeability, P is pressure, and g is the gravitational vector.

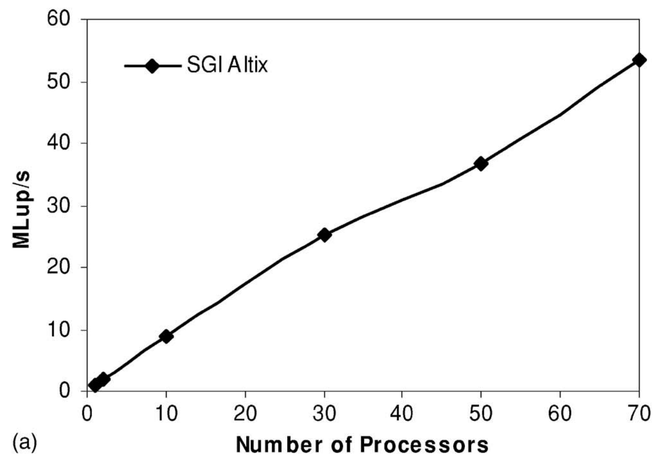
TABLE II. A comparison of the speedup between the ORB and the proposed methods.

Number of Processors	ORB (2.64×10^6 cells)	Cell-based (1.26×10^6 cells)	Cell-based (3×10^6 cells)
2		0.49	
4	1	1	1
8		1.38	4.41
9	1.95		
10		2.05	9.77
12	2.26		
16	2.85		
20	3.27	6.54	18.04
24	5		
30		8.74	20.47

Flow in a rectangular duct is performed first to test the model and the results are in good agreement with the theoretical solution. We then use the model to simulate flow in a column packed with glass beads. The porosity of the column was 0.37. Prescribed pressures are applied to the left and right sides of the column, and other sides are treated as periodic boundaries. The permeability is calculated from the simulation when flow reaches steady state. We also drive the fluid flowing in other two directions of the column to calculate the corresponding permeabilities. The averaged permeability over the three directions is $7.4 \times 10^{-9} m^2$ with a deviation of $\pm 0.2 \times 10^{-9}$ for the Origin 2000 and $\pm 0.15 \times 10^{-9}$ for the Altix 3700. This agrees well with experimental result [14].

VII. CONCLUSIONS

The work reported in this paper proposes a cell-based method to decompose a domain for parallel computation of fluid flow in porous media. The existing methods usually decompose a domain first and then recover the interfaces; the load balance is performed at last. The proposed method differs in that it performs the load balance first based on the total number of fluid cells and then decomposes the domain;



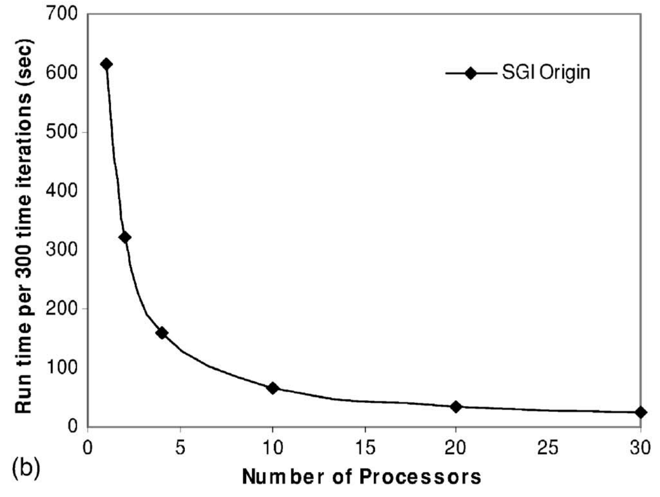


FIG. 5. (a) The million lattice site updates per second (MLup/s) vs number of processors using SGI Altix and (b) the total execution time taken per 300 time steps using SGI Origin.

the interfaces are recovered at last. As a result, the task of the proposed method is to recover the interfaces rather than the load balance as in the existing methods. Without special treatment, the proposed method can combine with the use of sparse matrix to optimize memory use. The proposed method is flexible and reliable for modeling flow in complex geometry. It has the following advantages: (i) automatically decomposes a complex flow domain, (ii) optimizes computer memory using sparse matrix that only store fluid cells, (iii)

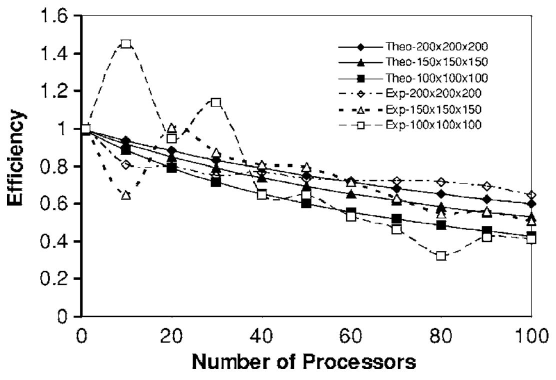


FIG. 6. Comparison of the theoretical efficiency and measured efficiency for SGI Origin 2000.

exact load balance in which the difference of fluid cells in each processor is 0 or 1, (iv) simple communication pattern and nearest communication connection among processors, and (v) high parallel efficiency in agreement with the theoretical efficiency.

The numerical experiments show that the decomposition produced by the proposed method is at least comparable to and in most cases better than that produced by the multilevel partitioning algorithms in terms of load balance, regular communication pattern, and nearest communication connection. Moreover, the proposed method is easy to code for flow in complex geometries. The examples shown in this paper are restricted to the LBM simulation of flow in porous media, but the method itself is applicable to other traditional numerical methods on either structured or unstructured grids. The inherent advantage of the proposed method makes searching, sorting, and optimizations no longer required as in the conventional domain-decomposition methods; the proposed method can evenly decompose a domain in just one scan.

ACKNOWLEDGMENTS

This work was funded by the Scottish Executive Environment and Rural Affairs Department and the Biotechnology and Biological Research Council of the U.K. The authors thank Dr. Nakashima for generously providing his three-dimensional image of glass beads. J.Y.W. would like to thank Dr. N. Stringfellow, Dr. J. B. Gibson, K. Roy, P. Adamson, and Z. Chaplin of the U.K. National Academic Supercomputing Service (CSAR) for their assistance and helpful discussions in implementing the codes.

[1] J. Wang, N. Christakis, M. K. Patel, M. C. Leaper, and M. Cross, Numer. Heat Transfer, Part A **45**, 751 (2004).
[2] K. Rastikian and R. Capart, J. Food. Eng. **35**, 419 (1998).
[3] M. Prat, Chem. Eng. J. **86**, 153 (2002).
[4] X. X. Zhang, A. G. Bengough, J. W. Crawford, and I. M. Young, Adv. Water Resour. **25**, 1 (2002).
[5] X. X. Zhang, A. G. Bengough, L. K. Deeks, J. W. Crawford, and I. M. Young, Water Resour. Res. **38**, 1167 (2002).
[6] K. Pruess and T. N. Narashimham, SPEJ **25**, 14 (1985).
[7] R. M. O'Connor and J. T. Fredrich, Phys. Chem. Earth, Part A Solid Earth Geod. **24**, 611 (1999).
[8] H. Freund, T. Zeiser, F. Huber, E. Klemm, G. Brenner, F. Durst, and G. Emig, Chem. Eng. Sci. **58**, 903 (2003).
[9] A. Bouddour, J.-L. Auriault, and M. Mhamdi-Alaoui, and J.-F. Bloch, Int. J. Heat Mass Transfer **41**, 2263 (1998).
[10] R. W. Zimmerman, T. Hadgu, and G. S. Bodvarsson, Adv. Water Resour. **19**, 317 (1996).
[11] J. Wood and L. F. Gladden, Chem. Eng. Sci. **57**, 3033 (2002).

[12] Y. Le Bray and M. Prat, Int. J. Heat Mass Transfer **42**, 4207 (1999).
[13] M. D. Mantle, A. J. Sederman, and L. F. Gladden, Chem. Eng. Sci. **56**, 523 (2001).
[14] Y. Nakashima and Y. Watanabe, Water Resour. Res. **38**, 1272 (2002).
[15] I. M. Young and J. Crawford, Science **304**, 1634 (2004).
[16] X. X. Zhang, L. K. Deeks, A. G. Bengough, J. W. Crawford, and I. M. Young, J. Hydrologic Eng. **306**, 50 (2005).
[17] F. Higuera and G. Jimenez, Europhys. Lett. **9**, 663 (1989).
[18] F. Higuera, S. Succi, and R. Benzi, Europhys. Lett. **9**, 4 (1989).
[19] H. Chen, S. Chen, and W. H. Matthaeus, Phys. Rev. A **45**, R5339 (1992).
[20] C. Shu, X. D. Niu, and Y. T. Chew, Phys. Rev. E **65**, 036708 (2002).
[21] S. Chen and G. D. Doolen, Annu. Rev. Fluid Mech. **30**, 329 (1998).
[22] P. L. Bhatnagar, E. P. Gross, and M. Krook, Phys. Rev. **94**, 511 (1954).
[23] Y. H. Qian, D. Dhumieres, and P. Lallemand, Europhys. Lett. **17**, 479 (1992).
[24] S. Succi, E. Foti, and F. Higuera, Europhys. Lett. **10**, 433 (1989).
[25] Z. L. Guo and T. S. Zhao, Phys. Rev. E **66**, 036304 (2002).
[26] C. Manwart and R. Hilfer, Physica A **314**, 706 (2002).
[27] N. S. Martys and J. G. Hagedorn, Mater. Struct. **35**, 650 (2002).
[28] A. Cancelliere, C. Chang, E. Foti, D. H. Rothman, and S. Succi, Phys. Fluids A **2**, 2085 (1990).
[29] A. Koponen, D. Kandhai, E. Hellen, M. Alava, A. Hoekstra, M. Kataja, K. Niskanen, P. Sloot, and J. Timonen, Phys. Rev. Lett. **80**, 716 (1998).
[30] J. Tolke, M. Krafczyk, M. Schulz, and E. Rank, Philos. Trans. R. Soc. London, Ser. A **360**, 535 (2002).
[31] Q. Zou, S. Hou, S. Chen, and G. Doolen, J. Stat. Phys. **81**, 35 (1995).
[32] Z. L. Guo and T. S. Zhao, Phys. Rev. E **66**(3), 036304 (2002).
[33] X. Y. He and L. S. Luo, J. Stat. Phys. **88**, 927 (1997).
[34] D. M. White, I. Halliday, C. M. Care, and A. Stevens, Physica D **129**, 68 (1999).
[35] C. Pan, J. F. Prins, and Cass T. Miller, Comput. Phys. Commun. **158**, 89 (2004).
[36] M. Schulz, M. Krafczyk, J. Tolke, and E. Rank, in *High-Performance Scientific and Engineering Computing*, Proceedings of the 3rd International FORTWIHR Conference on HPSEC, Erlangen, 2001, edited by M. Breuer, F. Durst, and C. Zenger (Springer-Verlag, Berlin, 2002), pp. 115–122.
[37] J. C. Desplat, I. Pagonabarraga, and P. Bladon, Comput. Phys. Commun. **134**, 273 (2001).
[38] P. Giovanni, F. Massaioli, and S. Succi, Comput. Phys. **8**, 705 (1994).
[39] G. Amati, S. Succi, and R. Piva, Int. J. Mod. Phys. C **8**, 869 (1997).
[40] D. Kandhai, A. Koponen, A. G. Hoekstra, M. Kataja, J. Timonen, and P. M. A. Sloot, Comput. Phys. Commun. **111**, 14 (1998).
[41] G. Karypis, K. Schloegel, and V. Kumar, Parallel graph partitioning and sparse matrix ordering library, 2003, http://www-users.cs.umn.edu/~karypis/metis/parmetis/files/manual.pdf
[42] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and analysis of algorithms* (Benjamin/Cummings, Redwood City, CA, 1994).
[43] G. Karypis and V. Kumar, SIAM J. Sci. Comput. (USA) **20**, 59 (1998).
[44] G. Karypis and V. Kumar, J. Parallel Distrib. Comput. **48**, 96 (1998).
[45] C. Walshaw and M. Cross (unpublished).
[46] D. A. Basermann, J. Clinckemaillie, T. Coupez, J. Fingberg, H. Digonnet, R. Ducloux, J.-M. Gratien, U. Hartmann, G. Lonsdale, B. Maerten, D. Roose, and C. Walshaw, Appl. Math. Model. **25**, 83 (2000).
[47] M. J. Aftosmis, M. J. Berger, and S. M. Murman (unpublished).
[48] M. P. I. Forum, Int. J. Supercomput. Appl. **8**, 159 (1994).
[49] T. Pohl, F. Deserno, N. Thurey, U. Rude, P. Lammers, G. Wellein, and T. Zeiser (unpublished).
[50] M. Sahimi, *Flow and Transport in Porous Media and Fractured Rock* (VCH Verlagsgesellschaft mbH, Weinheim, Germany, 1995).