Available online at www.sciencedirect.com
ScienceDirect
Journal of Computational Physics 227 (2008) 4895–4911
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

Performance evaluation of a parallel sparse lattice Boltzmann solver

L. Axner [a,*], J. Bernsdorf [b], T. Zeiser [c], P. Lammers [d], J. Linxweiler [e], A.G. Hoekstra [a]

[a] Section Computational Science, Faculty of Science, University of Amsterdam, Kruislaan 403, 1098 SJ, Amsterdam, The Netherlands
[b] NEC Laboratories Europe, NEC Europe Ltd., Rathausallee 10, D-53757 St. Augustin, Germany
[c] Regionales Rechenzentrum Erlangen, University of Erlangen-Nuremberg, Martensstr.1, D-91058 Erlangen, Germany
[d] HLRS, Nobelstrasse 19, D-70569 Stuttgart, Germany
[e] Institute for Computational Modeling in Civil Engineering, Pockelstrasse 3, D-38106 Braunschweig, Germany

Received 16 August 2007; received in revised form 13 December 2007; accepted 15 January 2008
Available online 26 January 2008

Abstract

We develop a performance prediction model for a parallelized sparse lattice Boltzmann solver and present performance results for simulations of flow in a variety of complex geometries. A special focus is on partitioning and memory/load balancing strategy for geometries with a high solid fraction and/or complex topology such as porous media, fissured rocks and geometries from medical applications. The topology of the lattice nodes representing the fluid fraction of the computational domain is mapped on a graph. Graph decomposition is performed with both multilevel recursive-bisection and multilevel *k*-way schemes based on modified Kernighan–Lin and Fiduccia–Mattheyses partitioning algorithms. Performance results and optimization strategies are presented for a variety of platforms, showing a parallel efficiency of almost 80% for the largest problem size. A good agreement between the performance model and experimental results is demonstrated.

Keywords: Sparse lattice Boltzmann; Partitioning; METIS; MPI performance measurements; Optimization; Performance prediction

1. Introduction

The lattice Boltzmann method (LBM) is a well established scheme in computational fluid dynamics (CFD) [1–3] and has proven to be a promising and reliable solver for fluid flow simulations in complex topologies such as porous medium, fissured rocks and geometries from medical applications [4–11]. However, for such large and complex topologies the simulations are known to be computationally intensive. This is why there

* Corresponding author. Tel.: +31 205257562; fax: +31 205257419.
E-mail addresses: labraham@science.uva.nl (L. Axner), j.bernsdorf@it.neclab.de (J. Bernsdorf), thomas.zeiser@rrze.uni-erlangen.de (T. Zeiser), plammers@hlrs.de (P. Lammers), j.linxweiler@tu-bs.de (J. Linxweiler), alfons@science.uva.nl (A.G. Hoekstra).

0021-9991/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2008.01.013

is an ongoing interest towards optimal LBM simulation codes and efficient parallelization strategies [12–22]. Key to achieve an optimum in parallel performance is the correct choice of domain decomposition method, to preserve the workload balance and to minimize the interprocessor communications.

A new LBM approach based on sparse matrix linear algebra is proposed by Schulz et al. [17]. Here the computational domain is mapped on an unstructured grid, where the geometrical ordering of nodes is rearranged and a new index list of only fluid nodes is composed. Thus each fluid node has an adjacency index list of its neighboring fluid nodes, while the solid nodes are completely eliminated. This method requires storage of three 1D arrays: two for density distributions (as a double buffering is used) and the other one for neighbor node indices. Next, domain decomposition has been performed for this scheme by using the METIS partitioning library [23–25]. With this parallelized sparse LBM approach memory consumption is essentially minimized and a highly optimized parallel performance can be achieved.

In this manuscript we extend this initial parallelized sparse LBM approach by performing detailed analysis of domain decomposition methods within the METIS library and their comparison to a 1D decomposition. This is done in terms of estimation of the number of edge-cuts (to find the amount of data that needs to be communicated between processors) and an estimation of the amount of fluid nodes per processors (to assess the load imbalance). Further we analyze performance results in terms of parallel scalability, and find the sources of loss of parallel efficiency, as well as predict the behavior of the solver for other geometries and/or on other systems using a detailed performance prediction model. This model includes both details of the parallel architecture (single node performance, point-to-point communication) and details of the domain decomposition. The sources of loss of efficiency are estimated in terms of fractional overhead functions [26].

In Section 2 we give a short overview of our sparse LB implementation, Section 3 describes the general idea behind partitioning methods. In Section 4 we develop the performance prediction model and in Section 5 we show the results of the parallel performance on different machine architectures and compare with the scalability prediction model. In Section 6 we present our conclusions.

2. Lattice Boltzmann method

The lattice Boltzmann method is based on the discrete velocity Boltzmann equation. In our simulations we use the lattice Bhatnagar–Gross–Krook model (LBGK) [1–3]. All parameters are in lattice units and we assume $\delta x = \delta t = 1$. The lattice-BGK equation is then as follows,

$$f_i(\mathbf{x}+\mathbf{e}_i, t+1) - f_i(\mathbf{x}, t) = -\frac{1}{\tau}[f_i(\mathbf{x}, t) - f_i^{(\mathrm{eq})}(\mathbf{x}, t)] \tag{1}$$

with $\mathbf{e}_i$ the finite set of discrete velocities, τ the dimensionless relaxation parameter, $f_i(\mathbf{x}, t)$ the density distribution function and $f_i^{(\mathrm{eq})}(\mathbf{x}, t)$ the equilibrium distribution defined by

$$f_i^{(\mathrm{eq})} = \rho w_i \left(1 + \frac{\mathbf{e}_i \cdot \mathbf{u}}{c_s^2} + \frac{(\mathbf{e}_i \cdot \mathbf{u})^2}{2c_s^4} + \frac{\mathbf{u} \cdot \mathbf{u}}{2c_s^2}\right). \tag{2}$$

Here w_i is a weighting factor, $c_s = 1/\sqrt{3}$ the speed of sound, ρ the hydrodynamic density determined by

$$\rho = \sum_{i=0}^{b} f_i = \sum_{i=0}^{b} f_i^{(\mathrm{eq})}, \tag{3}$$

and $\mathbf{u}$ the macroscopic velocity determined by

$$\rho \mathbf{u} = \sum_{i=0}^{b} \mathbf{e}_i f_i = \sum_{i=0}^{b} \mathbf{e}_i f_i^{(\mathrm{eq})}. \tag{4}$$

Here b is the number of directions. The viscosity v of the fluid is determined by

$$v = \frac{1}{6}\left(\tau - \frac{1}{2}\right). \tag{5}$$

2.1. Sparse LB implementation

The lattice Boltzmann method is said to be very efficient and easy to implement. But in most cases, a simple full matrix implementation is used, where not only the fluid and boundary nodes, but also the solid fraction is allocated in computer memory. Depending on the geometry, this is a considerable waste of resources, and also of CPU-cycles due to the long loops over solid nodes in order to reach the fluid ones. In the framework of a simple full-matrix implementation, the density distribution array for the whole bounding box is allocated in memory. This results in $2 \times 19 \times lx \times ly \times lz$ double precision floating point REAL numbers for the D3Q19 model for a $lx \times ly \times lz$ lattice [27,28].

Well-known methods from sparse matrix linear algebra were first applied to the lattice Boltzmann method by Schulz et al. [17]. Here the storage of the density distribution is only for the fluid nodes and thus the complete domain is mapped on an unstructured grid. Instead of geometrical ordering of those fluid nodes, a new index list is composed, where each node has the index list of its neighboring fluid nodes. In this method all data is stored in 1D arrays, where only $2 \times N \times 19$ double precision floating point REAL numbers are stored for the density distribution (N is the number of fluid nodes) and $N \times 18$ INTEGERs for the adjacency. This approach allows to save a considerable amount of memory.

3. The graph partitioning algorithms

The goal of the graph partitioning algorithms is to partition large and complex graphs such that the edge-cut is minimized (and thus the related communication overhead) while trying to keep the number of nodes on the sub graphs as balanced as possible (thus minimizing load imbalance). In sparse LBM the computational domain is mapped on a graph, where graph vertexes represent the fluid nodes and edges are the links connecting the fluid nodes. From this graph the 1D array of fluid nodes and adjacency index list, as described in Section 2.1, are constructed.

In previous studies, the METIS graph partitioning library was used to compute partitions from the graphs [17,18]. However, in neither of these studies one can find information or an evident objective for the chosen function from the multiple partitioning functions of METIS.

The METIS partitioning library contains different functions for graph and mesh partitioning. METIS reduces the size of the original graph by collapsing vertexes and edges, partitions a smaller graph and then uncoarsens it to construct a partition for the original graph [23–25]. In this way it produces high quality partitions and in addition is extremely fast. The partitioning functions are mainly based on the combination of two algorithms: modified Kernighan–Lin and Fiduccia–Mattheyses.

The Kernighan–Lin starts with an initial bipartition of the reduced size graph such that each half contains roughly half of the weight of the original graph and each further iteration it searches a subset of vertices/nodes from each part of the graph, such that their swapping will create a partition with smaller edge-cut [23]. The algorithm continues until it cannot find any two such subsets. The Kernighan–Lin algorithm takes $O(|E|\log|E|)$ time while Fiduccia–Mattheyses takes $O(E)$ [25], where E is the number of edge-cuts.

Fiduccia–Mattheyses can work with multiple partitions instead of two. The main achievement of this algorithm is its ability of analyzing the effect of moving a single vertex. That is, it can compute the change of the number of edge-cuts, caused by the movement of the current vertex from one partition to another, which is called gain. The algorithm repeats by selecting the vertex with the largest gain from the larger partition and moves it to another partition. To prevent the same vertex from moving in the same direction the algorithm marks the vertex [23].

In METIS the Kernighan–Lin partitioning algorithm is modified by the gain computing quality of the Fiduccia–Mattheyses algorithm. Moreover, the coarsening, partitioning and uncoarsening phases are also enhanced with a number of improvements. Thus, depending on the demands of the fluid flow solver one can adjust the parameters of the partitioning functions to achieve an approximate balance between maximum load balance and minimum communication amount. Moreover, METIS allows to use a (heavily) weighted graph, i.e., a graph with labels on nodes and/or edges, to strengthen the different features of partitions. The weights on edges of the graph can be used to strengthen the connection between nodes thus to help to

minimize the communication amounts. The weights on the nodes can be used to strictly preserve the load balance between partitions.

We use two different functions of the METIS partitioning library: multilevel k-way and multilevel recursive bisectioning (RB). The difference is that the *Multilevel RB* approach computes a k-way partitioning by performing recursive bisectioning on all phases of the multilevel graph partitioning. On the other hand, the *multilevel k-way* performs coarsening only once, then the coarsest graph is directly partitioned into k-parts and finally the uncoarsening phase is also performed only once, during which a k-way partitioning refinement algorithm is applied to improve the quality of the partitions [23]. For the 10^5 and 10^6 problem sizes (see Section 5) the measured execution times for *multilevel k-way* partitioning are equal 0.9 and 1.3 min, respectively while for *Multilevel RB* they are 2.1 and 2.5 min (both executed on a single processor of a PC cluster).

4. The performance prediction model

In order to analyze the parallel scalability of the sparse LB solver and to find the sources of loss of parallel efficiency we develop a performance prediction model. Using this model it is also possible to predict the parallel performance of the solver for other geometries. Inspired by the work of Fox et al. [26], who introduced the concept of fractional communication overheads, we write the equation for $T_p(N)$, the execution time on p processors for a problem containing N fluid nodes as

$$T_p(N) = \frac{T_1(N)}{p} + T_{\text{comm}}(N), \tag{6}$$

where T_{comm} is the communication time. Note, that we assume that the communication time on each processor is the same, and the computational time on each processor can be written as $T_1(N)/p$. This will be discussed in more detail below.

Now we can immediately write for the parallel efficiency $\varepsilon_{\text{p}}(N)$

$$\varepsilon_{\text{p}}(N) = \frac{T_1(N)}{p \times T_p(N)} = \frac{1}{1 + f_{\text{comm}}}, \tag{7}$$

where f_{comm} is the fractional communication overhead defined as

$$f_{\text{comm}} = \frac{p \times T_{\text{comm}}(N)}{T_1(N)}. \tag{8}$$

We note that if we would be able to express the parallel execution time in general as

$$T_p(N) = \frac{T_1(N)}{p} + \sum_i T_i(N), \tag{9}$$

where $T_i(N)$ are all overheads that can be identified, the efficiency could be expressed as

$$\varepsilon_{\text{p}}(N) = \frac{1}{1 + \sum_i f_i}, \tag{10}$$

with f_i a fractional overhead defined as

$$f_i = \frac{p \times T_i(N)}{T_1(N)}. \tag{11}$$

Next we will develop a detailed performance model for an application of size N. We assume that N discretizes some computational domain, and that the computation on each grid point is a stencil based operation, i.e., only information from neighboring grid points is needed to update the grid point itself. Parallelization is achieved by dividing the domain into p sub-domains, where each sub-domain has n_j points and

$$N = \sum_{j=1}^{p} n_j. \tag{12}$$

Note, that the n_j are not necessarily equal to N/p, i.e., we assume load imbalance, and therefore we expect to find some form of overhead induced by this load imbalance. Next, due to the stencil operation the amount of communication per domain is determined by the boundary of the sub-domain, which we denote by δn_j. Finally, we assume that the total execution time per processor is determined by computation on the domain n_j followed by communication on the boundary δn_j. This means that we exclude for now the possibility of latency hiding (a technique that may improve the performance a lot, and that could be considered when the fractional communication overheads become large) [29].

Now call t_j the execution time on processor j and write this as a summation of computation time (t_{comp}) and communication time (t_{comm}).

$$t_j = t_{\text{comp}}(n_j) + t_{\text{comm}}(\delta n_j) \tag{13}$$

The execution time on p processors is now determined by the slowest processor, i.e.,

$$T_p(N) = \max_j\{t_j\}, \tag{14}$$

and

$$T_1(N) = t_{\text{comp}}(N). \tag{15}$$

Without any further assumption we can only proceed by writing

$$T_p(N) = \frac{t_{\text{comp}}(N)}{p} + \max_j\{t_j\} - \frac{t_{\text{comp}}(N)}{p}, \tag{16}$$

and

$$\varepsilon_{\text{p}}(N) = \frac{1}{1+f}, \tag{17}$$

with a single fractional overhead defined as

$$f = \frac{p \times \max_j\{t_j\}}{t_{\text{comp}}(N)} - 1. \tag{18}$$

To proceed we need to find approximations for $T_p(N)$. We can find an upper bound to the execution time and use it as an approximation, allowing progress in estimating fractional overheads. We can write

$$T_p(N) = \max_j\{t_j\} \leqslant \max_j\{t_{\text{comp}}(n_j)\} + \max_j\{t_{\text{comm}}(\delta n_j)\} \tag{19}$$

The computational time is linear in the number of grid points. If we call τ_{comp} the time required to execute one grid point, and realize that due to caching and memory layout of the data τ_{comp} actually depends on the number of grid points we find that

$$t_{\text{comp}}(n) = n \times \tau_{\text{comp}}(n). \tag{20}$$

For communication we assume a linear model for point-to-point communication, i.e., to sent m density distributions requires a time $\tau_{\text{setup}} + m \times \tau_{\text{send}}$. τ_{setup} is the time needed to initialize the communication and τ_{send} is the time needed to send one byte of information. Each processor j will sent and receive data to and from d_j other processors. This number d_j depends on the details of the partitioning of the computational domain. To each of the d_j processors an amount of e_{jk} density ditributions (with $k = 1\ldots, d_j$ and 8 bytes per distribution) is communicated. Moreover, the total number of density distributions that are sent to other processors is the edge-cut e_j (in bytes) which is

$$e_j = \sum_{k=1}^{d_j} e_{jk}. \tag{21}$$

In stencil based operations the communication is always an exchange operation, that is, an amount of data is sent to a processor, and the same amount is received back again. We assume that this is implemented as two synchronous blocking point-to-point communication routines. With all these definitions we can now write a closed expression for the communication time from processor j as

$$t_{\text{comm}}(\delta n_j) = 2\sum_{k=1}^{d_j}(\tau_{\text{setup}} + e_{jk}\tau_{\text{send}}) = 2d_j\tau_{\text{setup}} + 2e_j\tau_{\text{send}}. \tag{22}$$

We assumed that the point-to-point communication between any pair of processors in a parallel computer has the same speed. In Section 5 we will refine this further, by distinguishing between two types of point-to-point communication.

From our partitioning tool METIS we can get all information, for each processor, on the number of nodes n_j, the degree of connectivity d_j and the edge-cut e_j.

With all these definitions we finally find

$$T_p(N) = n_{\max}\tau(n_{\max}) + \max_j\{2d_j\tau_{\text{setup}} + 2e_j\tau_{\text{send}}\} \tag{23}$$

By casting Eq. (23) in the form of Eq. (9) we find

$$T_p(N) = \frac{N}{p}\tau(N) + n_{\max}\tau(n_{\max}) - \frac{N}{p}\tau(N) + \max_j\{2d_j\tau_{\text{setup}} + 2e_j\tau_{\text{send}}\} \tag{24}$$

The last term in Eq. (24) is due to communication and gives rise to a fractional communication overhead, just like in Eq. (8). The second and third terms are in fact a mix of two effects, namely load imbalance and the potential change of the speed of a single processors due to the partitioning. We can separate them by writing

$$n_{\max}\tau(n_{\max}) - \frac{N}{p}\tau(N) = n_{\max}(\tau(n_{\max}) - \tau(N)) + \tau(N)\left(n_{\max} - \frac{N}{p}\right) \tag{25}$$

Now the processor speed effect and load imbalance are clearly separated. The first term on the right hand side of Eq. (25) is due to the processor speed effect, whereas the second is due to load imbalance.

This means that we find three fractional overheads:

- a fractional communication overhead

$$f_{\text{comm}} = \frac{p \times \max_j\{2d_j\tau_{\text{setup}} + 2e_j\tau_{\text{send}}\}}{N \times \tau(N)}; \tag{26}$$

- a fractional load imbalance overhead

$$f_l = \frac{p \times n_{\max}}{N} - 1; \tag{27}$$

- a fractional processor speed overhead

$$f_s = \frac{p \times n_{\max}(\tau(n_{\max}) - \tau(N))}{N \times \tau(N)} = \frac{p \times n_{\max}}{N}\left(\frac{\tau(n_{\max})}{\tau(N)} - 1\right). \tag{28}$$

The communication overhead and fractional load imbalance overhead are always equal or larger than zero, giving rise to a loss of efficiency. However, this is not the case for the fractional processor speed overhead. On cache based microprocessor the speed of the processor is typically faster for small problem sizes, when the problem fits completely in cache. If the original problem size N does not fit in cache and the decomposed problem size $n_{\max}$ does, we find that $\tau(n_{\max})/\tau(N) < 1$ and therefore $f_s < 0$, resulting in an increase of the parallel efficiency. This effect is well known and appears in the literature as super linear speedup, i.e., the situation where a measured speedup is larger than p, or equivalent, the efficiency $\varepsilon_{\text{p}} > 1$. Our analysis reveals that this happens when the summation over all fractional overheads is smaller than zero, or in this case, when $-f_s > f_{\text{comm}} + f_l$.

Note that this is all based on the upper bound estimation of the execution time (see Eq. (19)), so we expect to overestimate the execution time and therefore underestimate the parallel efficiency.

5. Results

We perform an extended set of experiments to assess the efficiency of the partitioning algorithms, both for complex and simple geometries. The geometries we use are the human abdominal aorta (AA), a porous

medium with high solid fraction (PM) and a straight square channel (SSC) (see Fig. 1). For all three geometries we have chosen three different domain sizes containing 5×10^6, 10^5 and 5×10^4 fluid nodes respectively. Thus in total we have nine experimental data sets. On each of these data sets we apply both the *Multilevel k-way* and *Multilevel RB* partitioning algorithm and examine their behavior. Due to special treatment of the inlet/outlet boundaries in the current version of our LBM solver we prefer to keep the nodes of first three layers at the inlets and outlets completely in one partition. To achieve this we put high weights on edges of those layers and put low weights on the rest of the edges throughout the complete graph. We compare the amount of edge-cuts between partitions and the amount of fluid nodes per partition. We also compare these two algorithms with 1D (operated on the original 3D domain by allowing only flat interfaces) bisection for SSC. Finally the execution time for all nine sets of data is measured on two different platforms: a NEC SX-8 vector machine [30] and a PC cluster [31], using from 1 up to 128 processors. In order to be able to compare the results with the performance prediction model and to find the sources of loss of parallel efficiency, we need first to evaluate the basic parameters connected with the parallel architecture (single node performance, point-to-point communication) and with the parallel decomposition (load imbalance, edge-cuts, degree distribution).

5.1. Basic parameters

(1) **Single processor speed** – We first measure $t_{\text{comp}}(N)$ by running multiple simulations with a sufficient range of data sizes on a single processor and using Eq. (20) extract $\tau_{\text{comp}}(N)$. Our experimental results (Fig. 2) show that on the PC cluster $\tau_{\text{comp}}(N)$ is almost constant if $N < 5 \times 10^3$ and $N > 10^5$ and increases in between. Thus we have divided the complete set of measurements into three regions. For small and large values of N, $\tau_{\text{comp}}(N)$ is assumed to be constant, and in the middle region we assume that $\tau_{\text{comp}}(N)$ depends linearly on N. As mentioned in Section 4, $\tau_{\text{comp}}(N)$ depends on the cache size of a specific processor. For the PC cluster the cache size is 1 Mbyte. From our computations we know that the memory consumption per lattice node is $2 \times 8 \times 19 = 304$ bytes (for 19 distributions). Thus if the number of lattice nodes per processor is $N < 3 \times 10^3$, the data will completely fit into cache.

It is well-known that vector machines such as the NEC SX-8 perform best for large data sizes. The experimental results show a rather smooth decrease of $\tau_{\text{comp}}(N)$ as N increases. Assume that the execution

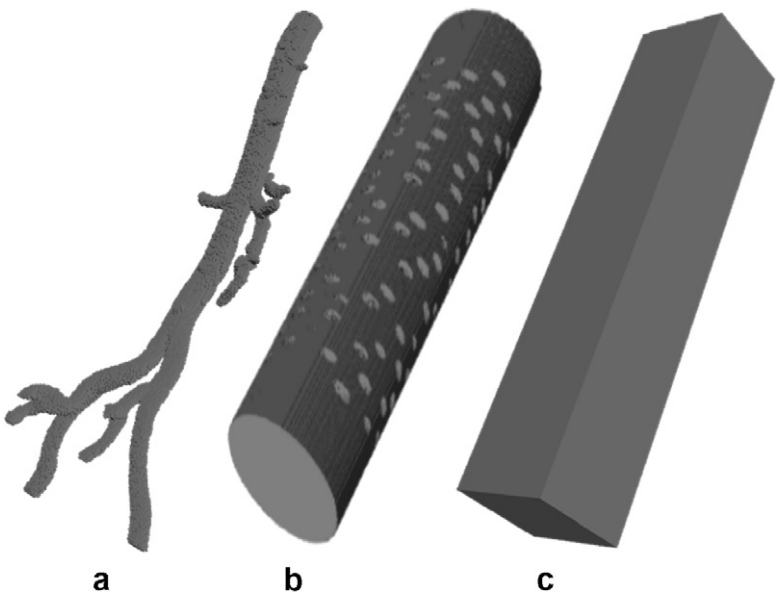

Fig. 1. (a) – Abdominal aorta, (b) – porous medium, and (c) – straight square channel.

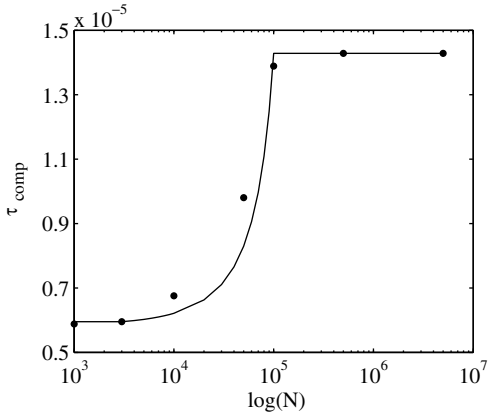

Fig. 2. Single processor performance on the PC cluster as a function of data size for the SSC; (bullets) are the measurements, and (line) is a fit to the data.

time depends linearly on N (e.g. a constant time to fill up the vector units, and then a time proportional to N to produce all results), then the execution time can be written as $a \times N + b$. Using Eq. (20) we find $\tau_{\text{comp}}(N) = a + b/N$, and this function we use to fit the measurements.

In Fig. 2 and 3 we have plotted $\tau_{\text{comp}}(N)$ for the LBM solver on a single processor as a function of the number of fluid nodes for PC cluster and the NEC SX-8 machine respectively for SSC. We see a good agreement between measurements, together with the fitting results. A clear discontinuity due to the cache size for the PC cluster at $N = 3 \times 10^3$ data size can be observed. In Fig. 3 we clearly see that the bigger the problem size the better the performance. We have fitted the experimental results to the models as described above, and the results are summarized in Table 1.

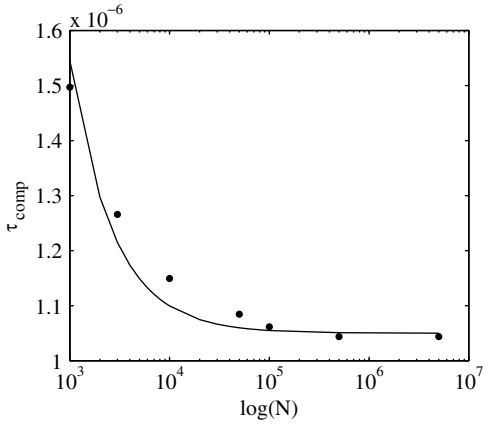

Fig. 3. Single processor performance on the NEC SX-8 vector machines as a function of data size for the SSC; (bullets) are the measurements, and (line) is a fit to the data.

Table 1
Single processor performance

Platform	N	$\tau_{\text{comp}}(N)$ (s)
PC cluster	$<3 \times 10^3$	0.58×10^{-5}
	$>10^5$	1.42×10^{-5}
	Otherwise	$0.58 \times 10^{-5} + \frac{1.42\times10^{-5}-0.58\times10^{-5}}{10^5-3\times10^3} \times (N - 3 \times 10^3)$
NEC SX-8	For all data sizes	$1.05 \times 10^{-6} + 4.95 \times 10^{-4}/N$

(2) **Partitioning** – In Fig. 4 we show the SSC and the AA data sets partitioned using the *multilevel RB* and *multilevel k-way* algorithms. We see that the *Multilevel RB* creates more sliced cuts between partitions, while the partitions of *multilevel k-way* are less structured and have curved cuts. However, for both cases the average number of fluid nodes per processor is very close to N/P. To get a better idea of the load imbalance, in Fig. 5 we have plotted the standard deviation of the distribution of the number of fluid nodes on processors as a function of p. From the measurements of standard deviation we hardly observe a difference between partitionings using *Multilevel RB* and *multilevel k-way* on all geometries, except for SSC for a very large number of partitions. This is due to the fact that *Multilevel RB* performs bipartitioning along the shortest latitude of the geometry until it reaches the minimum possible number of layers per partition and continues bipartitioning along the longitude which causes a high load imbalance for $p = 128$. For complex geometries we prefer to follow the advice of Karypis et al. [23] and use *Multilevel RB* for less than eight partitions while apply *multilevel k-way* for a larger number of partitions. Moreover, we have compared these algorithms with a 1D bisection and noted that both METIS algorithms

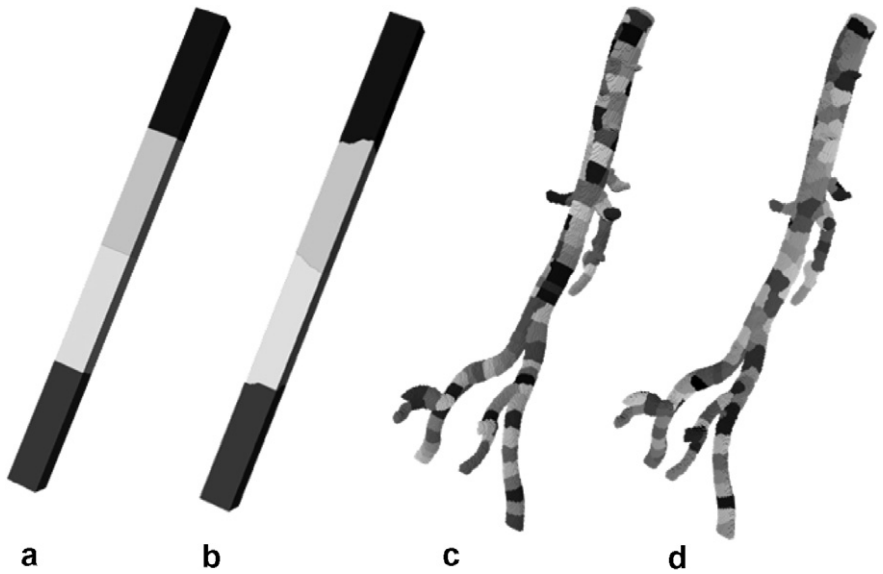

Fig. 4. (a) – SSC with four partitions created with the *RB* method, (b) – as (a) but now with the *k-way* method, (c) – AA, 128 partitions created with the *RB* method and (d) – as (c) but now with the *k-way* method.

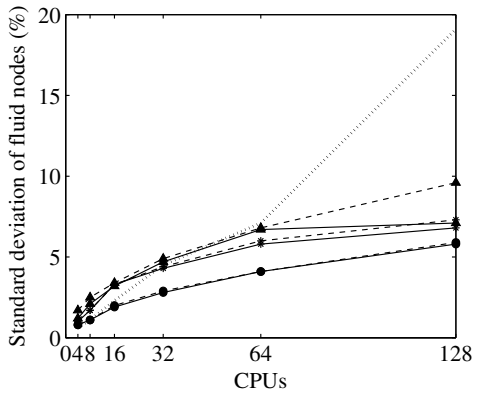

Fig. 5. The standard deviation of fluid nodes per processor for the largest data size (i.e., with $N = 5 \times 10^6$) for AA (bullets), PM (stars) and SSC (triangles) with both RB (solid lines) and k-way (dashed lines). 1D bisection (dotted line) is only for SSC geometry.

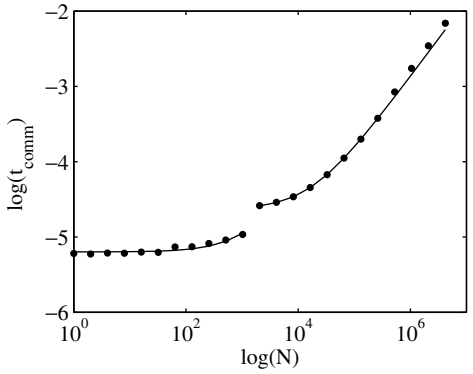

Fig. 6. Point-to-point communication time (seconds) as a function of data size (bytes) on the PC cluster. The bullets are the measured data and the solid line is a fit to the data.

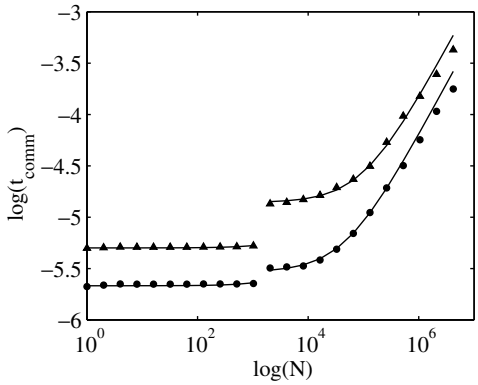

Fig. 7. Point-to-point communication time (seconds) as a function of data size (bytes) on the NEC SX-8 vector machines. Triangles are the measured off-board point-to-point communication times, bullets are the measured on-board point-to-point communication times and the solid lines are the fit to the data.

performed better. That is, even though 1D bisectioning divides the domain into equal parts, after a certain number of partitions the deviation of number of fluid nodes per partition becomes quite high from the average number of fluid nodes per partition.

(3) **Point to point communication** – This overhead depends on the specifics of the machine architectures and the details of the edge-cut and degree of connectivity of the partitions. Moreover, most modern machines have more than one processor per node, and therefore we must also distinguish between on-board communication (with both processors on the same node) and off-board communication (between processors on different nodes). In our case this applies only[1] to the NEC SX-8 which has eight processors per node.

To measure the point-to-point communication we run a "pingpong"-wise communication with different data sizes between two processors. Next we fit the data to a linear function and obtain τ_{setup} and τ_{send}. The results are shown in Figs. 6 and 7. Moreover, in Fig. 7 we see a clear difference between communication times for on-board and off-board measurements. Thus in our further predictions for the NEC SX-8 for $p \leqslant 8$ we take the on-board values and for $p > 8$ the off-board ones. We also note that for $N = 1$ Kbyte a discontinuity appears. This is a buffering effect in the MPI communication routines. From the fits we extract values of τ_{setup} and τ_{send}, which are shown in Table 2.

[1] At the time of experiments on the PC cluster, due to technical reasons, only one processor per node was available.

Table 2
Point to point communication overhead

Platform	N (bytes)	a (s)	b (s/bytes)
PC cluster	<1024	6.34×10^{-6}	4.77×10^{-9}
	>1024	1.38×10^{-5}	1.54×10^{-9}
NEC SX-8 (On board)	<1024	2.15×10^{-6}	1.45×10^{-10}
	>1024	2.90×10^{-6}	6.09×10^{-11}
NEC SX-8 (Off board)	<1024	5.02×10^{-6}	2.64×10^{-10}
	>1024	1.40×10^{-5}	1.86×10^{-10}

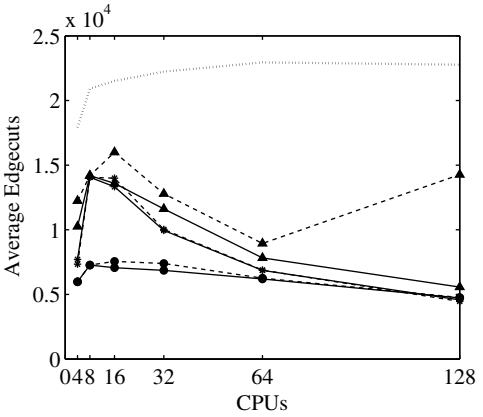

Fig. 8. The average number of edge-cuts per processor for the largest data size (i.e., with $N = 5 \times 10^6$) for AA (bullets), PM (stars) and SSC (triangles) with both RB (solid lines) and k-way (dash lines). 1D bisection (dotted line) is only for SSC geometry.

Next we consider the edge-cuts per processor obtained from the partitioning algorithms. For complex geometries like AA or PM both the *multilevel k-way* and *Multilevel RB* behave almost similarly as shown in Fig. 8. The standard deviations of edge-cuts are in the order of 25%. Considering the fact that the *multilevel k-way* is faster but the partitions of *Multilevel RB* are more structured we follow the advice of Karypis et. al [24] and use *Multilevel RB* if $p \leqslant 8$ and the the *multilevel k-way* otherwise. However, in the case of SSC after certain number of partitions (e.g., 64 in Fig. 8) the standard deviation of edge-cuts produced by *Multilevel RB* is in the order of 30% while in the case of the *multilevel k-way* it is 40%. Thus for simple geometries like SSC we prefer to use *Multilevel RB* as the communication time associated with it will be less. We also conclude that the difference between these two partitioning functions is mainly in the number of edge-cuts rather than load imbalance.

5.2. Performance measurements of the lattice Boltzmann application

After measuring all the parameters required for the performance model we will now present the performance measurements for the LBM simulations. First we show detailed measurements of the total communication times and compare them to the model expressed in Eq. (22), and next we present the total execution time of the LBM simulations as a function of N and p.

Using the point-to-point communication overheads we compute the total communication times from the communication model and compare them with the measured ones in Figs. 9 and 10. We observe a reasonable agreement between measurements and predictions on either machine. The maximum difference is for the SSC case, which for the PC cluster it is about 28% while for the NEC SX-8 it is 40%. As for the AA and PM the agreement is better, especially for the NEC SX-8. Similar behavior was observed for the cases with smaller number of fluid nodes (data not shown).

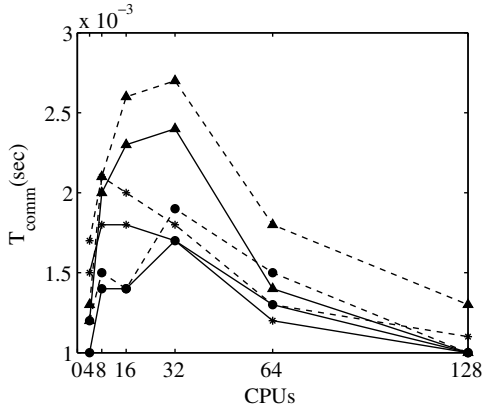

Fig. 9. Total communication time as a function of p on the PC cluster for the largest geometries (i.e., with $N = 5 \times 10^6$). The solid lines are the measurements and the dashed lines are model predictions from Eq. (22). The bullets are for AA, the stars for PM and the triangles for SSC.

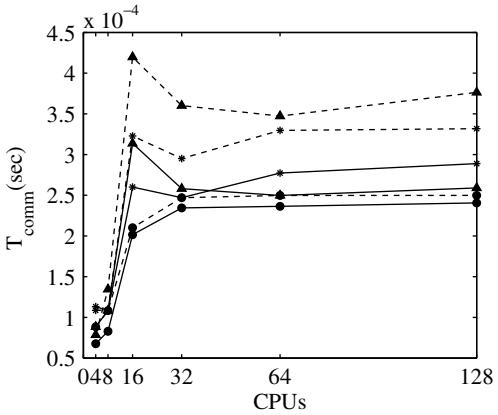

Fig. 10. Total communication time as a function of p on the NEC SX-8 for the largest geometries (i.e., with $N = 5 \times 10^6$).The solid lines are the measurements and the dashed lines are model predictions from Eq. (22). The bullets are for AA, the stars for PM and the triangles for SSC.

Next, we measure the total execution times for all geometries with all three data sizes, and also compute the execution times from performance model (Eq. (24)). The results shown in Figs. 11 and 12 are for the AA case. For the other two geometries we find comparable results (data not shown). The overestimation of the execution times for the biggest data size is about 5%, while for the smallest data size for $p = 128$ it is about 40% on the PC cluster and 10% on the NEC SX-8. As was mentioned in Section 4, this is due to the fact that all the computations of prediction model are based on the upper bound estimations. We also plot in Figs. 13 and 14 the execution times in term of Lattice Updates per second (LUP/s) for all data sizes on both architectures. In Fig. 13 we see the super linear speed-up due to the "cache effect" on the PC cluster for the smallest geometries with 5×10^4 fluid nodes. On NEC SX-8 (Fig. 14), as was expected, the best performance we get for the largest data sizes, of almost 75% peak performance for 128 processors.

In Figs. 15 and 16 we show the efficiencies as a function of p. Here the underestimation of the prediction model is approximately 4%. Thus the agreement between measurements and performance model is quite high. Also in Fig. 16 we see a steep decline of efficiencies for the medium and small data sizes. This is due to the fact that on vector machines for such small problem sizes we see more the single processor effect (as we partition the problem, the single processor performance goes down, see Fig. 3) and therefore a large positive fractional processor speed overhead (Eq. (28)). For small problem sizes on the PC cluster the situation is reversed. Here

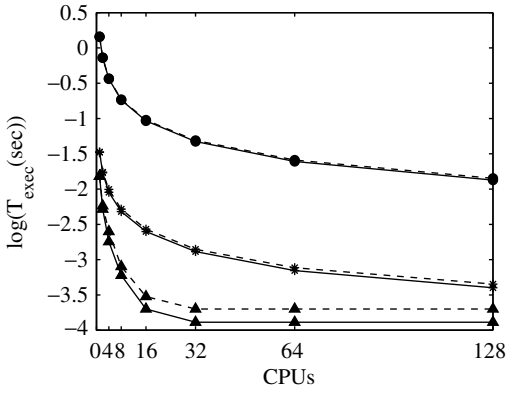

Fig. 11. Execution time as a function of p on the PC cluster for AA for all three problem sizes. The solid lines are measurements and dashed lines are predictions from the performance model. The bullets are for $N = 5 \times 10^6$, the stars for $N = 10^5$ and the triangles are for $N = 5 \times 10^4$.

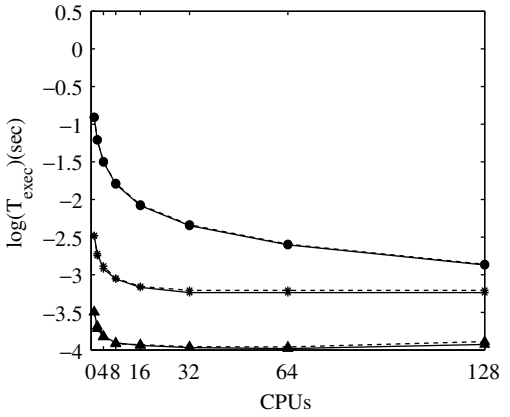

Fig. 12. Execution time as a function of p on the NEC SX-8 vector machines for AA for all three problem sizes. The solid lines are measurements and dashed lines are predictions from the performance model. The bullets are for $N = 5 \times 10^6$, the stars for $N = 10^5$ and the triangles are for $N = 5 \times 10^4$.

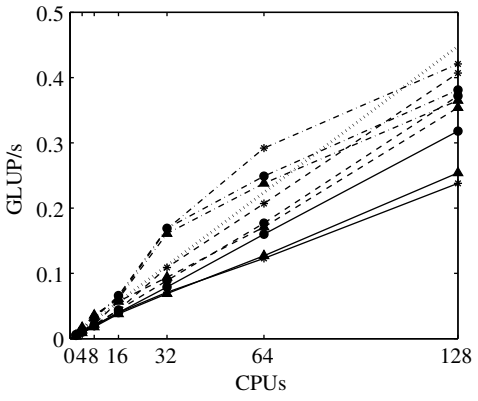

Fig. 13. Lattice updates per second as a function of p on the PC cluster for $N = 5 \times 10^6$ (solid lines), $N = 10^5$ (dashed lines) and $N = 5 \times 10^4$ (dotted lines). Bullets are AA, starts are PM and triangles are SSC. The dotted line indicates the ideal peak performance.

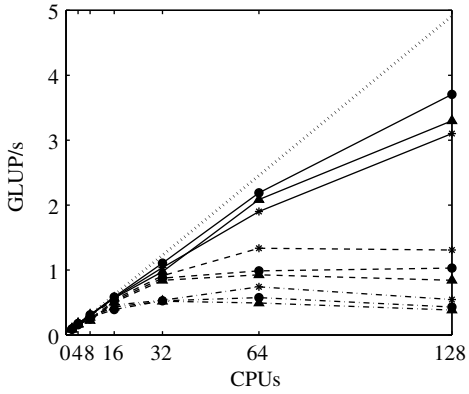

Fig. 14. Lattice Updates per second as a function of p on the NEC SX8 machines for $N = 5 \times 10^6$ (solid lines), $N = 10^5$ (dashed lines) and $N = 5 \times 10^4$ (dotted lines). The bullets are AA, the starts are PM and the triangles are SSC. The dotted line indicates the ideal peak performance.

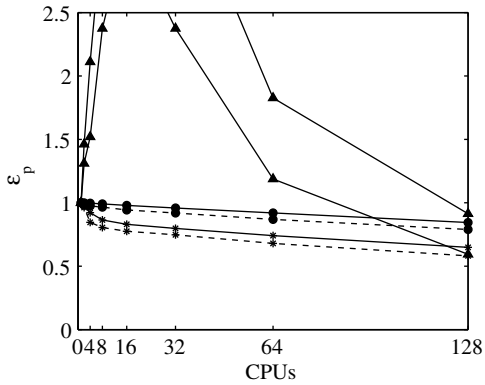

Fig. 15. Efficiencies as a function of p on the PC cluster. The solid lines are measurements and dashed lines are predictions from the performance model. The bullets are for $N = 5 \times 10^6$, the stars for $N = 10^5$ and the triangles are for $N = 5 \times 10^4$.

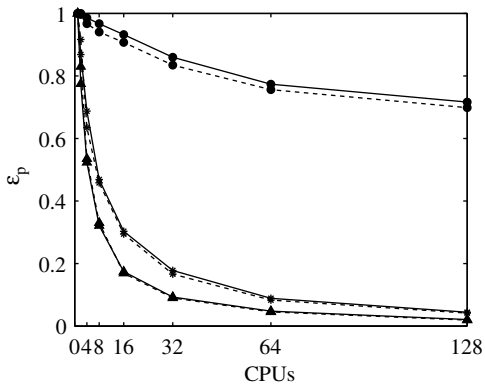

Fig. 16. Efficiencies as a function of p on the NEC SX-8 vector machines. The solid lines are measurements and dashed lines are predictions from the performance model. The bullets are for $N = 5 \times 10^6$, the stars for $N = 10^5$ and the triangles are for $N = 5 \times 10^4$.

the fractional processor speed overhead is negative due to the caching effect, and we observe efficiencies much larger than one (see Fig. 15).

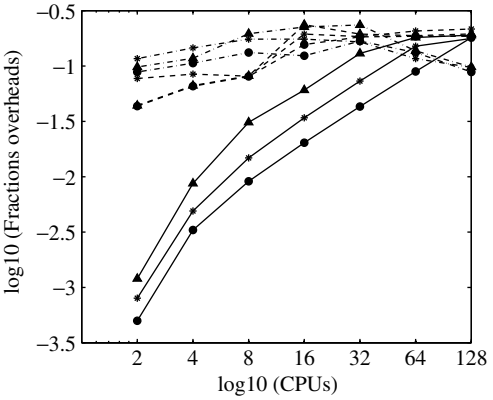

Fig. 17. (a) – Fractional overheads for the largest data size (e.g., $N = 5 \times 10^6$). Bullets are AA, stars are PM and triangles are SSC. The solid lines are the load imbalance overheads, dashed lines and dotted–dashed lines are the communication overheads on the NEC SX-8 and the PC cluster respectively.

In Fig. 17 we have plotted fractional communication and fractional load imbalance overheads (Eqs. (26) and (27)) as a function of p. From the plot it is clear that for large p the loss of the efficiency due to the load imbalance can be as large as 18%, while the loss due to communication overhead is about 7% on the NEC SX-8 and 20% on the PC cluster. For a small number of processors the fractional load imbalance overhead is negligible as compared to fractional communication overhead. Thus from Fig. 17 we conclude that for a small number of processors the best way to improve the efficiency is the implementation of latency hiding [29] to decrease the communication overhead. For a large number of processors we propose to use a better partitioning algorithm to improve the load balance. By comparing the fractional load imbalance overheads as a function of number of fluid nodes we also observed that the bigger the data size the easier it is to achieve good load balance (data not shown). Therefore we can try to extrapolate our results to much larger data sizes. If we assume that the single processor speed overhead for large data sizes is zero, the point-to-point communication overhead decreases as $1/N^{2/3}$ and for a 10^8 data size geometry we decrease the load imbalance by 10% we can achieve almost 95% peak performance.

All these comparisons confirm that the performance model has a good accuracy and it is eligible for predicting the performance for any size/type of geometry, if one knows the estimated fractional overheads.

6. Conclusions

In this paper we have developed a performance prediction model and tested it by comparing the computed performance results with time complexity measurements of our parallel sparse lattice Boltzmann solver. In order to interpret the results, we defined three sources of parallel efficiency loss, i.e., fractional processor speed overhead, fractional communication overhead and fractional load imbalance overhead and measured them for all nine cases and two architecturally different supercomputers: a NEC SX-8 vector machine and a PC cluster.

For the three different geometries with three different data sizes we have preformed a graph partitioning by using two different functions of the METIS graph partitioning library. We compared the partitioning results and conclude that with simple geometries is it better to use the *multilevel RB* method rather than *multilevel k-way*, while for complex ones the combination of both will give optimal solution [23].

The measurements show that for the largest data sizes we obtain almost 75% peak performance on NEC SX-8 vector machine. Moreover with the help of the performance model we estimate that the efficiency on the NEC SX-8 can be improved if one chooses more robust partitioning algorithms as the efficiency loss for large p is mostly due to load imbalance. The comparison of these measurements with prediction model gives us 5% error, which shows that the developed model can be used for scalability prediction of parallel performance for any geometries.

Acknowledgements

Lilit Axner acknowledges financial support from the Dutch National Science Foundation (Token program, Distributed Interactive Medical Exploratory for 3D Medical Images Project – 634.000.024) and HPC-Europa Transnational Access program (RII3-CT-2003-506079). Special thanks to Michael Scarpa from the University of Amsterdam for help with visualization of the geometries. The lattice Boltzmann flow solver used for this publication was developed in the scope of the International Lattice Boltzmann Software Development Consortium.

References

[1] Sauro Succi, The Lattice Boltzmann Equation for Fluid Dynamics and Beyond, Oxford, New York, 2001.
[2] Shiyi Chen, Garry D. Doolen, Lattice Boltzmann method for fluid flows, Annual Review Fluid Mechanics 30 (1998) 329–364.
[3] Y.H. Qian, D. d'Humieres, P. Lallemand, Lattice BGK models for Navier–Stokes equation, Europhysics Letters 17 (1992) 479–484.
[4] A. Koponen, M. Kataja, J. Timonen, Drona Kandhai, Simulation of single-fluid flow in porous media, International Journal of Modern Physics C 9 (1998) 1505–1522.
[5] Drona Kandhai, D. Hlushkou, Alfons G. Hoekstra, Peter M.A. Sloot, H. van As, U. Tallarek, Influence of stagnant zones on transient and asymptotic dispersion in macroscopically homogeneous porous media, Physical Review Letters 88 (23) (2002) 234501-1–234501-4.
[6] D.S. Clague, Drona Kandhai, R. Zhang, Peter M.A. Sloot, On the hydraulic permeability of (un)bounded fibrous media using the lattice-Boltzmann method, Physical Review E 61 (1) (2000) 616–625.
[7] Shiyi Chen, Zheng Wang, Xiaowen Shan, Gary D. Doolen, Lattice Boltzmann computational fluid dynamics in three dimensions, Journal of Statistical Physics 68 (3/4) (1992) 379–400.
[8] Abdel M. Artoli, Alfons G. Hoekstra, Peter M.A. Sloot, Mesoscophic simulations of systolic flow in the human abdominal aorta, Journal of Biomechanics 39 (5) (2006) 873–884.
[9] Abdel M.M. Artoli, Alfons G. Hoekstra, Peter M.A. Sloot, Simulation of a systolic cycle in a realistic artery with the lattice Boltzmann BGK method, International Journal of Modern Physics B 17 (1–2) (2003) 95–98.
[10] J.A. Cosgrove, J.M. Buick, S.J. Tonge, C.G. Munro, C.A. Greated, D.M. Campbell, Application of the lattice Boltzmann method to transition in oscillatory channel flow, Journal of Physics A: Mathematical and General 36 (2003) 2609–2620.
[11] Haiping Fang, Zuowei Wamg, Zhifang Lin, Muren Liu, Lattice Boltzmann method for simulating the viscous flow in large destensible blood vessels, Physical Review E 65 (2002) 051925–052011.
[12] Abdel M. Artoli, Alfons G. Hoekstra, Peter M.A. Sloot, Optimizing lattice Boltzmann simulations for unsteady flows, Computers & Fluids 35 (2) (2006) 227–240.
[13] Drona Kandhai, A. Koponen, Alfons G. Hoekstra, M. Kataja, J. Timonen, Peter M.A. Sloot, Lattice Boltzmann hydrodynamics on parallel systems, Computer Physics Communications 111 (1998) 14–26.
[14] Wei Li, Xiaoming Wei, Arie E. Kaufman, Implementing lattice Boltzmann computation on graphics hardware, The Visual Computer 19 (7–8) (2003) 444–456.
[15] Jean-Christophe Desplat, Ignacio Pagonabarraga, Peter Bladon. Ludwig, A parallel lattice-Boltzmann code for complex fluids, Computer Physics Communications 134 (3) (2002) 273.
[16] Thomas Pohl, Frank Deserno, Nils Thurey, Ulrich Rude, Peter Lammers, Gerhard Wellein, Thomas Zeiser, Performance evaluation of parallel large-scale lattice Boltzmann applications on three supercomputing architectures, in: SC'04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, IEEE Computer Society, Washington, DC, USA, 2004, p. 21.
[17] M. Schulz, M. Krafczyk, J. Toelke, E. Rank, Parallelization strategies and efficiency of CFD computations in complex geometries using lattice-Blotzmann methods on high-performance computers, in: 3rd International FORTWIHR Conference on HPSEC, Erlangen, March 12–14, 2001, 2002, p. 115.
[18] Alexandre Dupuis, Bastien Chopard, An object oriented approach to lattice gas modeling, Future Generation Computer Systems 16 (5) (2000) 523–532.
[19] J. Tlke, S. Freudiger, M. Krafczyk, An adaptive scheme for LBE multiphase flow simulations on hierarchical grids, Computers and Fluids 35 (8–9) (2006) 820.
[20] P. Lammers, G. Wellein, T. Zeiser, M. Breuer, Have the vectors the continuing ability to parry the attack of the killer micros? in: M. Resch, T. Bönisch, K. Benkert, T. Furui, Y. Seo, W. Bez (Eds.), High Performance Computing on Vector Systems. Proceedings of the High Performance Computing Center Stuttgart, March 2005, Springer, Berlin, 2006, pp. 25–39.
[21] G. Wellein, T. Zeiser, P. Lammers, U. Küster, Towards optimal performance for lattice Boltzmann applications on terascale computers, in: A. Deane, G. Brenner, A. Ecer, et al. (Eds.), Parallel Computational Fluid Dynamics: Theory and Applications, Proceedings of the 2005 International Conference on Parallel Computational Fluid Dynamics, May 24–27, College Park, MD, USA, Elsevier, Amsterdam, 2006, pp. 31–40.
[22] C. Pan, J. Prins, C.T. Miller, A high-performance lattice Boltzmann implementation to model flow in porous media, Computer Physics Communication 158 (2) (2004) 89.
[23] George Karypis, Vipin Kumar, Multilevel *k*-way partitioning scheme for irregular graphs, Journal of Parallel and Distributed Computing 481 (1) (1998) 96–129.

[24] Amine Abou-Rjeili, George Karypis, Multilevel algorithms for partitioning power-law graphs, in: IEEE International Parallel & Distributed Processing Symposium (IPDPS), vol. 10, 2006.
[25] George Karypis, Vipin Kumar. Multilevel graph partitioning schemes, in: International Conference on Parallel Processing, vol. 3, 1995, p. 113–122.
[26] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Solman, D. Walker, Solving Problems on Concurent Processors, Prentice-Hall Int., Inc., 1988.
[27] J. Bernsdorf, S.E. Harrison, S.M. Smith, P.V. Lawford, D.R. Hose, High performance computing on vector systems 2006, in: Proceedings of the High Performance Computing Center Stuttgart, March, 2006.
[28] G. Wellein, G. Hager, T. Zeiser, S. Donath, On the single processor performance of simple lattice Boltzmann kernels, Computers and Fluids 35 (8–9) (2006) 910–919.
[29] J.B. White III, S.W. Bova, Where's the overlap? An analysis of popular MPI implementations, in: MPIDC'99: Proceedings of the Message Passing Interface Developer's and User's Conference 1999, March 10–12, Atlanta, 1999.
[30] http://www.hpce.nec.com/.
[31] http://www.sara.nl/userinfo/lisa/description/index.html/.