Parallel Simulation of Fluid Slip in a Microchannel *

Jingyu Zhou, Luoding Zhu, Linda Petzold, and Tao Yang
Department of Computer Science
University of California, Santa Barbara
{jzhou, zhuld, petzold, tyang}@cs.ucsb.edu

Abstract

Fluid flow in channels has traditionally been assumed to satisfy a no-slip condition along the channel walls. However, it has recently been found experimentally that the flow in microchannels can slip along hydrophobic (repelling water) walls. The slip can have important physical consequences for such flows. The physical mechanism underlying this phenomena is not well understood. An accurate, physically realistic model that can be simulated rapidly is critical for obtaining a better understanding of these results, and ultimately for modeling and for optimizing the flow in microdevices to achieve desired objectives.

This paper investigates the parallel simulation of fluid slip along microchannel walls using the multicomponent lattice Boltzmann method (LBM) with domain decomposition. Because of the high complexity for microscale simulation, even a parallel computation of fluid slip can take days or weeks. Any slowness in the participating nodes in a cluster can drag the entire computation substantially, due to frequent node synchronization involved in each computational phase of the algorithm. We augment the parallel LBM algorithm with filtered dynamic remapping for lattice points. This filtered scheme uses lazy remapping and over-redistribution strategies to balance the computational speed of participating nodes and to minimize the performance impact of slow nodes on synchronized phases. Our experimental results indicate that the proposed technique can greatly speed up fluid slip simulation on a non-dedicated cluster over a long period of execution time.

*This work was supported by NSF/ITR ACI-0086061

1 Introduction

In fluid mechanics, a no-slip boundary condition is usually assumed for viscous flow over a solid surface. This means that the fluid velocity at the surface is the same as the velocity of the surface (zero if the surface is at rest). However, it has recently been found experimentally that the no-slip assumption may be not valid for some micro scale flows, for example flows in micro-electro-mechanical systems (MEMS). Many researchers [6, 20, 15, 40, 41, 26, 1, 7, 48, 24, 35, 38, 36, 37] have investigated fluid slip phenomena for microscale flows. Apparent fluid slip at the micro-device walls can have a significant influence on the mass and heat transfer in the system and has drawn much recent attention. Among these results is a recent laboratory experiment performed by Tretheway and Meinhart [36]. They found an approximate 10% fluid slip with respect to the main stream flow velocity in a three-dimensional microchannel with hydrophobic walls (the walls are coated with a material which repels water molecules). [1] The mechanism of fluid slip on hydrophobic surfaces is not yet well understood. A

[1] For a comprehensive review of fluid slippage over hydrophobic surfaces, see Vinogradova [39] and the references therein. The hydrophobicity of a solid surface is not well understood. We refer the following papers [22, 19, 17, 3, 4, 28, 38] to interested readers.

possible mechanism for generating the observed slip has been proposed in [37] (see Section 2 for details).

We investigated the proposed mechanism by simulating the water-air two-phase system with the multicomponent lattice Boltzmann method for flow in a three-dimensional hydrophobic microchannel [47]. Micro-scale simulation for fluid slip is computation-intensive. It can take hundreds of days on a fast single-processor machine, especially when high resolution is required. Previous work has focused on the parallelization of the single component LBM with static domain decomposition. Satofuka [29] parallelized the single-component LBM in two and three dimensions via domain decomposition. Kandhai *et al.* [18] adopted the Orthogonal Recursive Bisection method to parallelize the single-component LBM. Suviola [34] parallelized the single-component LBM for simulating suspension flow. Previous work (e.g. [33]) has focused mainly on static-decomposition of the grid into equal sub-volumes, based on slice, box, or cubic partitioning. Here we report on our parallelization of the multicomponent LBM via domain decomposition with *filtered dynamic lattice point remapping*.

For the multi-component LBM, we use domain decomposition, which divides the problem domain into a number of sub-domains. Computation on sub-domains is performed by separate processes that execute the same functions on different data. Because the sub-domains are connected at their boundaries, processes belonging to neighboring sub-domains must synchronize on their boundaries. Synchronizations divide the computation into phases. While full linear speedup for parallel multi-component LBM can be achieved in a dedicated cluster, the parallelized code can still take weeks in a modest-sized cluster for a high-resolution microscale simulation. In a production environment for scientific simulation, it is very possible that a cluster is shared with other tasks. Phase-based synchronization with slow nodes in LBM drags the entire computation significantly in a non-dedicated environment. For example, a single heavily-loaded node can slow down the program's execution time by a factor of two to three as we will show later.

It is clear that we need to improve self-adaptiveness of the LBM algorithm in responding to the slowness of some nodes. The challenge for dynamic load balancing is to tolerate transient spikes and to avoid excessive communication needed for data remapping. We have proposed a scheme called filtered remapping which adopts lazy migration that minimizes unnecessary re-balancing, and uses an over-redistribution strategy during lattice point migration, to proactively minimize the impact of slow nodes. Our experimental results show that the proposed scheme can balance work load in the presence of slow nodes and effectively handle transient spikes, with significant improvement compared to other methods.

This paper is structured as follows. Section 2 presents the multicomponent LBM and its parallelization with domain decomposition. Section 3 gives motivation and details on filtered dynamic lattice point remapping. Section 4 provides the simulation and evaluation results. Finally, Section 5 concludes the paper.

2 Fluid Slip and Lattice Boltzmann Method

For fluid flows at micro or nano scale, the lattice Boltzmann method provides a more physically realistic means of simulation approach than the Navier-Stokes (N-S) equations. When the Kn number [2] is much larger than 1, (which is often true for problems at micro or nano scale) the N-S equations are not valid, but the lattice Boltzmann method is still valid. In our work, the parallelized multi-component lattice Boltzmann method with filtered dynamic load rebalancing was applied to investigate a possible generating mechanism for apparent fluid slip [37] at hydrophobic microchannel walls. The key idea of the mechanism is as follows. The water used in the experiment [36] was not treated to remove dissolved or entrained gases. It is conjectured that the gases and the water vapor may form a depleted thin

[2] Defined as the ratio of the mean free path of molecules and the characteristic length of a problem.

layer between the hydrophobic surfaces and the water. This depleted water region could generate the observed apparent fluid slip because the overall density in the thin layer is much lower than that of water.

We investigated the above mechanism by simulating the water-air two-phase system with the parallelized multicomponent lattice Boltzmann method for flow in a three-dimensional hydrophobic microchannel. The hydrophobic walls were modeled by applying a force in a region very close to the walls. This force is repulsive to the water molecules, and neutral to the air molecules. Note that, unlike the methods of Molecular Dynamics or Direct Simulation Monte Carlo, there are no "molecules" in the lattice Boltzmann method. The introduced hydrophobic force acts on the single particle velocity distribution functions, instead of on the molecules themselves. These forces decay exponentially away from the wall. The initial water-air mixture is assumed to be uniform. The initial density of the air in the water is calculated under standard conditions. The multicomponent lattice Boltzmann model we use is standard, see [30, 31, 32, 23], except that we introduced the additional hydrophobic wall forces into the formulation.

2.1 Lattice Boltzmann Method

The lattice Boltzmann method [25, 5, 2, 8, 27, 21, 43, 14, 16] is an alternative to traditional numerical methods for simulating fluid flows governed by the Navier-Stokes equations. It solves for the single particle velocity distribution functions that satisfy a simplified Boltzmann equation, instead of solving for the macroscopic quantities like fluid velocity and pressure. Compared to the direct discretization of the incompressible Navier-Stokes equations, the lattice Boltzmann method has several advantages (See [27] for details). For example, the simplified Boltzmann equation is a scalar linear differential equation, while the Navier-Stokes equations are a system of nonlinear differential equations. Among these comparative advantages is the local behavior of both the collision and streaming operators. Therefore this method is very natural for parallelization.

The multicomponent lattice Boltzmann method used in our work is the S-C model [11, 30, 31, 32]. A brief account of this method is as follows. Let σ denote the fluid components. For each fluid component σ ($\sigma = 0, 1$ in our case), a single particle velocity distribution function $f^\sigma(\mathbf{x}, \boldsymbol{\xi}, t)$ is introduced, which solves the LBGK model for that component:

$$\frac{\partial f^\sigma(\mathbf{x}, \boldsymbol{\xi}, t)}{\partial t} + \boldsymbol{\xi} \cdot \frac{\partial f^\sigma(\mathbf{x}, \boldsymbol{\xi}, t)}{\partial \mathbf{x}} = -\frac{1}{\tau^\sigma}(f^\sigma(\mathbf{x}, \boldsymbol{\xi}, t) - f^{\sigma(0)}(\mathbf{x}, \boldsymbol{\xi}, t)).$$

Here τ^σ and $f^{\sigma(0)}$ are the relaxation time and the equilibrium distribution function for component σ, respectively.

After discretization in particle velocity space $\boldsymbol{\xi}$ and in time t, the multicomponent LBE is obtained:

$$f_j^\sigma(\mathbf{x}+\boldsymbol{\xi}_j, t+1) = f_j^\sigma(\mathbf{x}, t) - \frac{1}{\tau^\sigma}(f_j^\sigma(\mathbf{x}, t) - f_j^{\sigma(0)}(\mathbf{x}, t)),$$

where f_j^σ is the distribution function for the σ component along the direction $\boldsymbol{\xi}_j$. Note that the discretization in $\boldsymbol{\xi}$ is the same for each component.

An interparticle interaction potential, which is intended to model the interaction between different components, can be defined as

$$V(\mathbf{x}, \mathbf{y}) = \sum_\sigma \sum_{\sigma'} G_{\sigma\sigma'}(\mathbf{x}, \mathbf{y}) \psi^\sigma(\mathbf{x}) \psi^{\sigma'}(\mathbf{y}).$$

Here the Greens function, $G_{\sigma\sigma'}(\mathbf{x}, \mathbf{y})$, characterizes the nature of the interaction between different components (attractive or repulsive and its strength). The choice of ψ determines the equation of state of the system under study. By selecting different functions G and ψ, various fluid mixtures and multiphase flows can be simulated.

The equilibrium distribution $f_j^{\sigma(0)}$ can be written as

$$f_j^{\sigma(0)}(\mathbf{x}, t) = \rho^\sigma(\mathbf{x}, t) w_j (1 + 3\boldsymbol{\xi}_j \cdot \mathbf{u}^\sigma(\mathbf{x}, t) + \frac{3}{2}(3\boldsymbol{\xi}_j \boldsymbol{\xi}_j : \mathbf{u}^\sigma(\mathbf{x}, t)\mathbf{u}^\sigma(\mathbf{x}, t) - \mathbf{u}^\sigma(\mathbf{x}, t) \cdot \mathbf{u}^\sigma(\mathbf{x}, t))),$$

where w_j is the weight. The mass density of component σ is defined by

$$\rho^\sigma(\mathbf{x}, t) = \sum_j m^\sigma f_j^\sigma(\mathbf{x}, t),$$

where m^σ is the molecular mass of component σ. The velocity, $\mathbf{u}^\sigma$, is computed via

$$\rho^\sigma(\mathbf{x},t)\mathbf{u}^\sigma(\mathbf{x},t) = \rho^\sigma(\mathbf{x},t)\bar{\mathbf{u}}(\mathbf{x},t)$$

$$+\tau^\sigma \frac{dp^\sigma}{dt}(\mathbf{x},t) + \tau^\sigma \mathbf{h}^\sigma(\mathbf{x}),$$

where the average velocity $\bar{\mathbf{u}}$ is defined by

$$\bar{\mathbf{u}}(\mathbf{x},t)\sum_\sigma \frac{\rho^\sigma(\mathbf{x},t)}{\tau^\sigma} = \sum_\sigma \left(\frac{m^\sigma}{\tau^\sigma} \sum_j f_j^\sigma(\mathbf{x},t)\boldsymbol{\xi}_j \right).$$

Here $\frac{dp^\sigma}{dt}$ is the net rate of momentum change that can be computed in terms of the interaction potential:

$$\frac{dp^\sigma}{dt}(\mathbf{x},t) = -\sum_{\mathbf{y}} \nabla_{\mathbf{y}}(\mathbf{x},\mathbf{y}) =$$

$$-\psi^\sigma(\mathbf{x}) \sum_{\sigma'} \sum_j G^j_{\sigma\sigma'} \psi^{\sigma'}(\mathbf{x}+\boldsymbol{\xi}_j)\boldsymbol{\xi}_j.$$

The forces $\mathbf{h}(\mathbf{x})$ are the hydrophobic forces we introduced to simulate the hydrophobic walls. Our choice of $\mathbf{h}^\sigma(\mathbf{x})$ is as follows: (index 0 denotes the fluid in the model to simulate the water and index 1 the fluid to simulate the air/water vapor)

$\mathbf{h}^1(\mathbf{x}) = 0$,

$\mathbf{h}^0(\mathbf{x}) = (0, g_2(y), g_3(z))$,

where y is the distance away from the side walls along the inward normal direction, and z has a similar meaning for the top and bottom walls. $g_2(y)$ and $g_3(z)$ are in the form of $c_1 \times exp(-y/c_2)$. Here c_1 and c_2 are constants to be determined.

The macroscopic quantities are connected to distribution functions by the following relations:

$$\rho(\mathbf{x},t) = \sum_\sigma \rho^\sigma(\mathbf{x},t),$$

$$(\rho\mathbf{u})(\mathbf{x},t) = \sum_\sigma m^\sigma \sum_j f_j^\sigma \boldsymbol{\xi}_j + \frac{1}{2}\sum_\sigma \frac{dp^\sigma}{dt}(\mathbf{x},t).$$

The dimensionless viscosity of the system is defined by

$$\nu = \frac{\sum_\sigma \frac{2\rho^\sigma\tau^\sigma}{\rho} - 1}{6}.$$

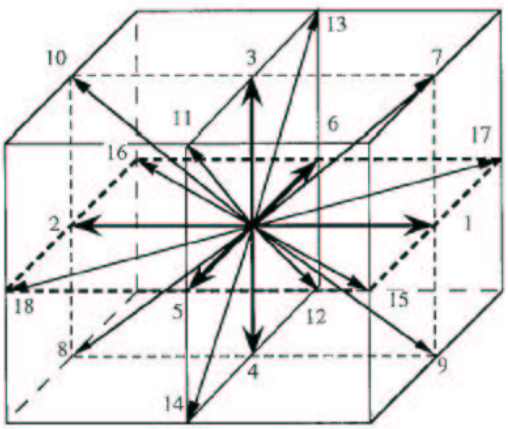

Figure 1: The three-dimensional lattice for D3Q19 model. Each node has 19 different possible movement of directions.

2.2 Parallelization of LBM

We performed the parallelization of the multi-component LBM in three-dimensions via domain decomposition. Because of the special geometry in our application (the x direction is much longer than the y and z directions), one-dimensional decomposition along the x direction was used. The microchannel was partitioned into N cubics along the x direction. Each cubic was assigned to a processor. For each processor, the distribution functions of the two components at both ends along directions $1, 7, 9, 15, 17$ must be sent to its right neighbor and the distribution functions along directions $2, 8, 10, 16, 18$ must be sent to its left neighbor. In the meantime, each processor must wait to receive the distribution functions of the two components at both ends along directions $2, 8, 10, 16, 18$ from its right neighbor, and the distribution functions along directions $1, 7, 9, 15, 17$ from its left neighbor, see Figure 1.

In each phase of the LBM computation, a lattice point requires a constant number of floating operations. Our application's problem domain is a cubic by size $N_x \cdot N_y \cdot N_z$. The time complexity of the program is $O(N_X \cdot N_y \cdot N_z)$. Figure 2 gives the pseudo-code of the Lattice Boltzmann method. Different processors are assigned different starting and ending indices on the x axis. Lines 4 through 17 represent

```
1  s = starting index on X axis;
2  e = ending index on X axis;
3  for (phase = 1; phase <= TOTAL_PHASES; phase++) {
4     Compute collision from s to e;
5     Compute streaming from s to e;
6
7     /* communication */
8     Exchange distribution func. with neighbors;
9
10    Compute bounce back;
11    yz_direction(s, e);
12
13    /* communication */
14    Exchange number density with neighbors;
15
16    Compute force from s to e
17    Compute velocity from s to e;
18
19    /* lattice points remapping */
20    if ( phase % REMAPPING_INTERVAL == 0 ) {
21        t_local = estimate_time();
22
23        /* communication */
24        Exchange t_local with neighbors;
25
26        Compute remapping amount;
27
28        /* communication */
29        Redistribute data;
30
31        Update s and e;
32    }
33 }
```

Figure 2: Pseudo code of parallelized Lattice Boltzmann method.

the computation of a phase. The velocity computed on line 17 is used in the collision computation of line 4 in the next iteration. The distribution function and the number density data on the boundaries must be exchanged with neighboring nodes in each phase (line 8 and line 14, respectively). The application requires a large number of simulation steps. Every few phases, the program performs a remapping of the lattice points (from line 20 to 32) to improve performance by adjusting the load on different processors and reducing the synchronization overhead.

3 Filtered Dynamic Remapping of Lattice Points

We have parallelized the fluid slip code using MPI. In this section we will discuss the motivation for dynamic lattice point remapping and then present a filtered remapping method to minimize the impact of slow nodes.

3.1 Motivation

While the code can achieve an almost perfect speedup in a dedicated cluster, the parallel code still takes a long time (weeks) to complete the simulation for even a small microchannel. Any machine slowness due to resource sharing or operational error can dramatically slow down the entire computation due to the nature of synchronization among nodes conducted at each phase of the computation. This is because other nodes must wait for the slowest one, in order to synchronize.

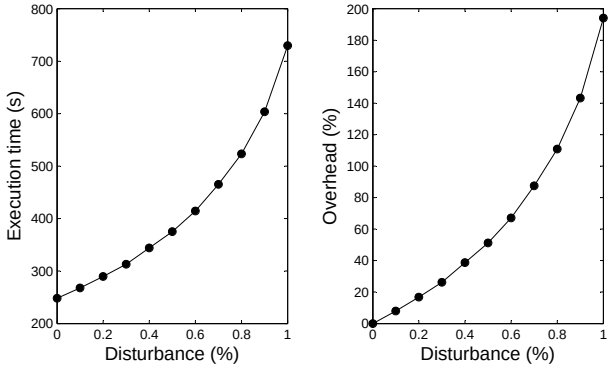

Figure 3: Increased time caused by competing jobs.

To find out whether short load spikes or long load spikes have the most impact, we performed a series of experiments. In these experiments, our MPI-based fluid slip program used 20 nodes and 600 phases representing a small percentage of phases required in the simulation described in Section 4. One of the nodes was slowed down by a CPU-intensive competing job which ran throughout the execution of our MPI program. Every 10 seconds, it spent a certain percentage of time competing for CPU resources; it slept the rest of the time. We varied that percentage from 0 to 100% and plotted the results in Figure 3. On the left, the execution time of our MPI program is plotted with respect to different disturbance levels. The increased time per program phase is plotted on the right. The curve shows that the

overhead is close to a linear increase when the disturbance is less than 60% (which corresponds to 6 seconds in each 10 seconds period) and sharply increases after that point. This suggests that short load spikes have a relatively small impact on the overall performance. However, the sharp increase after 60% means that if the high load persists longer, the MPI program's performance suffers greatly. Considering that the time for running our MPI program under perfect conditions is about 250 seconds for this selected case, the overhead caused by persistent competing jobs could be several times higher.

The reason for significant slowdown even by a single slow node is due to a ripple effect of the LBM code with domain decomposition. As all neighboring nodes must synchronize in each phase, at one phase the neighbor nodes are slowed down by the slowest node; in two phases, nodes with distance two away are slowed down; the slowdown continues until all processes are affected in 10 to 20 phases depending on the position of the slowest node.

These experiments suggest that the dominant performance slowdown is caused by a persistent slow node and that transient load spikes have less performance impact. Thus, the remainder of this section focuses on improving performance in the case of persistent slow nodes.

3.2 Self-Adaptiveness with Dynamic Data Remapping

The above experimental result shows that the impact of a persistent slow node must be minimized to reduce program execution time. Dynamic load balancing [42] is a common technique used to schedule computation in an adaptive manner such that fast nodes perform more work. We first describe dynamic lattice mapping used to achieve load balancing in general and explain the dynamic mapping challenge in our context, and then propose a solution called filtered dynamic remapping.

3.3 Issues for Dynamic Mapping of Lattice Points

Since computational complexity of our application at each processor is proportional to the number of lattice points assigned to the processor, remapping of lattice points is needed to achieve load balancing in the presence of slow and fast nodes. Each node needs to monitor program execution and adaptively reduce the number of lattice points if this machine becomes slower compared to others in the cluster. Figure 4 illustrates the dynamic data remapping operation. Initially, lattice points are evenly distributed among all nodes (Figure 4-a). When node 2 becomes slower, some of its lattice points will be shifted to neighboring nodes 1 and 3 (Figure 4-b).

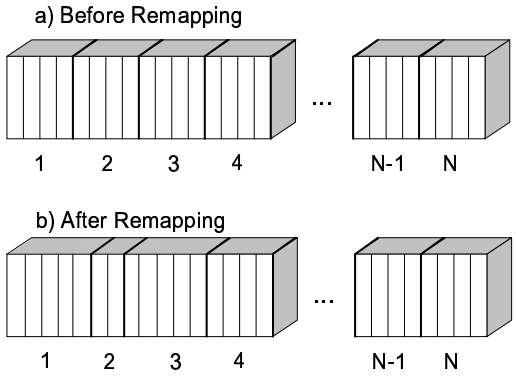

Figure 4: Example of data remapping.

There are three important issues that we need to address for remapping. Our solutions for the first and the third issues are different from what have been proposed in the load balancing literature [42].

- **Performance prediction.** Each machine needs to predict how long it would take to process the existing lattice points in the next phase based on the performance in the previous phases. Earlier results have found that CPU load can be predicted using simple models [9], and future load is closer to the most recent data [46, 13]. Our finding is that prediction that mainly depends on performance data of the most recent phase can result in excessive data migration

(we call it migration oscillation) when the cluster sharing pattern changes rapidly. Our design choice is to use the harmonic average of processing time at the last n phases that minimizes the chance of excessive remapping.

- **Global vs local information exchange.** We consider two different approaches for dynamic data remapping, i.e., global or local information exchange. In the global approach, load information is exchanged among all nodes and a global decision is made to distribute lattice points evenly among all nodes such that all nodes are expected to complete in the same time before the next phase starts.

  The major problem with the above global solution is that synchronization overhead can be significant. This is because group communication is required among all nodes for making a global decision. Also a node may have to shift data to multiple destinations or to receive new lattice points from multiple sources. Since prediction of load may not be accurate, the benefits of global load balancing guided by prediction may not pay off, considering the excessive communication overhead involved.

  We have followed the previous work in distributed load sharing [42] where balancing is decided locally among neighbor nodes.

- **Conservative redistribution vs. over-redistribution.** When fast nodes and slow nodes are identified among neighbor nodes and lattice point migration is planned, a decision must be made on the number of lattice points to be transfered. The previous approach for general load balancing using local information exchange [42] has taken a conservative view: when a node is considered light and a heavily-loaded node needs to shift x amount of load, typically a smaller amount such as $x * \delta$ will actually be transferred (e.g. $\delta = 30\%$). The reason is that a light node may be considered "light" by everybody and this node can be overloaded by receiving loads from multiple sources independently.

  In our setting with a linear array of processors and with neighbor data dependence in synchronization for LBM, we find that over-redistribution, which aggressively shifts points, performs much better than conservative redistribution. We will discuss the reason below.

We call our overall approach filtered dynamic lattice point mapping. Our design choice is based on the following strategies.

- **Lazy remapping.** Frequent re-balancing of load can hurt overall performance in two aspects. First, it can cause excessive data migration when the load changes rapidly. Secondly, remapping can introduce an unbalanced load if estimation is not accurate, and can lead to greater communication overhead.

  Our strategy in designing the filtered approach is to remap lazily, which reduces synchronization cost and also allows the system to carefully respond to usage spikes in a cluster. We use this strategy in two aspects of our filtered remapping method. 1) use the average load index of the last n phases to guide prediction instead of using the load index of the most recent phase. 2) We avoid moving data points from a fast node to a slow node even if the slow node has much fewer lattice points. The reason is that slow nodes can result in sluggish communication and the benefit of exploiting slow nodes for their computing power is not significant considering that communication to these nodes can be very slow.

- **Over-redistribution of lattice points from confirmed slow nodes.** When a node is detected to be slow with high confidence, it is better to minimize the use of such a node since it not only processes data at a low speed but also drags the entire computation due to sluggish communication. Once the system finds that a node is really slow, our strategy is to aggressively shift data from slow nodes to faster nodes.

3.4 Filtered Dynamic Remapping

Performance prediction. Let the execution time for the last n phases at a node x be T_x^1, T_x^2,, T_x^n. The predicted time (load index) for the next phase T_x is defined:

$$\frac{n}{\frac{1}{T_x^1} + \frac{1}{T_x^2} + \cdots \frac{1}{T_x^n}}.$$

Thus if there is a load spike during the last phase, no migration will be made unless this machine is really slow for the last n phases (in our experiment, n=10).

Load index exchange and remapping condition checkup. Only neighboring nodes in the linear array of cluster nodes will communicate. We describe the formula for making a migration decision among three neighboring nodes. The formula is similar for the first node and the end node in the linear array.

Given three neighboring nodes $i-1$, i, and $i+1$, we decide whether we should move some data points from node i to $i+1$ as follows. Let L_{i-1}, L_i, L_{i+1} be the number of lattice points on nodes $i-1$, i, $i+1$. Let T_{i-1}, T_i, T_{i+1} be their predicted times respectively. Let L'_{i-1}, L'_i, L'_{i+1} be the *intended* number of lattice points after remapping. We define the *processing speed* of node i as

$$S_i = \frac{L_i}{T_i}.$$

The intention of remapping is to balance these three nodes in the next phase through this local information change on the behalf of node i. The objective is:

$$\frac{L'_{i-1}}{S_{i-1}} = \frac{L'_i}{S_i} = \frac{L'_{i+1}}{S_{i+1}} = \frac{L_{i-1} + L_i + L_{i+1}}{S_{i-1} + S_i + S_{i+1}}.$$

L'_{i-1}, L'_i, L'_{i+1} can be computed from the above equation. When $L'_{i+1} > L_{i+1}$, the number of data points that need to be remapped from node i to $i+1$ is $L'_{i+1} - L_{i+1}$. This condition $L'_{i+1} > L_{i+1}$ can be rewritten as:

$$\frac{L_{i-1} + L_i + L_{i+1}}{S_{i-1} + S_i + S_{i+1}} > T_{i+1}.$$

A similar condition can be derived for deriving the tentative amount of points redistributed from node i to $i-1$.

Remapping decision and over-redistribution. An additional consideration is whether we should move ΔL_{i+1} points from node i to $i+1$, where $\Delta L_{i+1} = L'_{i+1} - L_{i+1}$. The following remapping condition needs to be met: ΔL_{i+1} is greater than a threshold and $S_{i+1} > S_i$. Namely, we don't move a small number of points and don't move points from a fast node to a slow node. In our experimental setting discussed in Section 4, the minimal migration is one 2D plane of size 200×20 for a microchannel with $400 \times 200 \times 20$ lattice points. Thus we use 4,000 as the threshold.

If node i is considered to be slow and redistribution of lattice points from i is necessary, we will shift data aggressively from this slow node to another faster node. Given three consecutive nodes $i-1$, i, and $i+1$, let S_{i-1}, S_i, and S_{i+1} be their processing speed respectively. Instead of ΔL_{i+1}, $\alpha \Delta L_{i+1}$ lattice points will be redistributed from node i to $i+1$. The scaling factor we use is $\alpha = \frac{S_{i+1}}{S_i}$.

In some rare cases, when the local information exchange among nodes $i-1$, i, $i+1$ concludes node i needs to shift points to $i+1$, if node $i+1$ concludes that it needs to shift data from node $i+1$ to i based on local information exchange among nodes i, $i+1$, and $i+2$, a conflict resolution is deployed between node i and $i+1$ to redistribute a proper amount of lattice points.

4 Simulation of Fluid Slip and Parallel Performance

We performed the simulation of fluid slip on a $0.1 \times 1 \times 2\,\mu m^3$ micro-channel. (Figure 5 shows the diagram of the microchannel.) The grid spacing is $5\,nm$. The nondimensional hydrophobic wall force used in the simulation is 0.2, corresponding to a physical force of $4 \times 10^{-3}\ dyne/cm^3$ with a decay length of $6.5\,nm$. The appropriate magnitude for this force is not well understood. For the current simulation, the force function was chosen so that the simulation results would be consistent with experimental observations. The main simulation results are as follows.

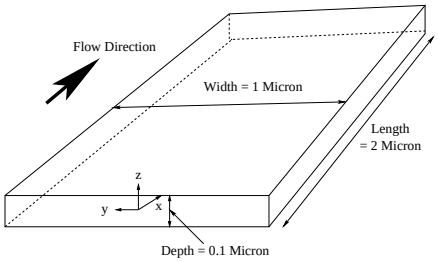

Figure 5: Diagram of the microchannel.

We will first discuss the simulation results, which match well the experimental observations. Then we will present the performance results on parallel simulation of this fluid slip instance.

4.1 Simulation Results

Figure 6 shows fluid densities close to the two side walls as a function of distance away from the side wall at the cross-section $x = 1\ \mu m$ and $z = 50\ nm$. The x-axis is the density, and the y-axis is the distance from the side wall. Figure 6 (A) shows the density of the fluid used to simulate water in the model along the y-direction (in the middle of the z direction) on a cross-section in the middle of the channel (x-direction). Figure 6 (B) shows the density of the fluid used to simulate water vapor/air. We can see that the density of water is decreased and that of water vapor/air is increased close to the walls. Sakurai [28] *et al.* have also observed a drastic decrease of the water molecule number density at a monolayer-water interface from the simulation results of water between hydrophobic surfaces, via molecular dynamics. Our results are consistent with theirs.

Figure 7 shows the normalized streamwise velocity profile and a local blowup along the y-direction at cross-section $x = 1\ \mu m$ for $z = 50\ nm$. The x-axis is the normalized velocity, and the y-axis is the position from the side wall (unit: micron). The solid line (in A and B) is the velocity profile when no wall forces are present. The dotted line (in part A), or the dashed line (in part B) is the case where wall forces are introduced. In contrast to the former

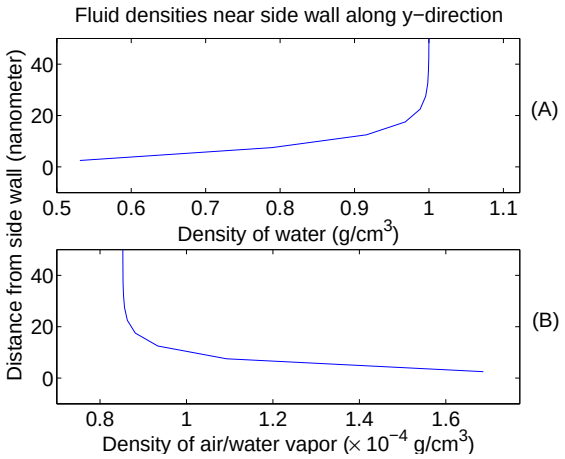

Figure 6: Fluid densities as a function of distance away from the side wall at the cross-section $x = 1\ \mu m$ and $z = 50\ nm$. The 50 nm region close to the side wall is shown. The x-axis is the density of water or air/water vapor, and the y-axis is the distance from the side wall. The graph (A) is the density profile for water and the graph (B) is the density profile for air/water vapor.

case, the latter results in apparent slip at the walls. (See Figure 7 (B) for the local blowup near the side wall.) We can see from Figs. 7 and 8 that in the region very close to the walls, the water density decreases and the water vapor/air density rises. This enables the fluid slip on the walls (approximately 9% of free stream velocity), compared to the solid lines in Figure 7, which illustrate the case where no hydrophobic wall forces were applied.

4.2 Parallel Simulation with Filtered Dynamic Lattice Mapping

We have conducted parallel simulation of the above fluid flow instance on a Linux cluster of 32 PC nodes with 3GB RAM and dual-processor 2.6 GHz Xeon, connected by a Gigabit Ethernet switch. For the simulation setting we have used $400\times200\times20$ lattice points with slice decomposition on 20 cluster nodes, i.e., a $20\times200\times20$ lattice for each node. The total running time for this problem with 20,000 LBM steps (phases) on a single machine is 43.56 hours. We choose this problem size because it allows us to repeat various experiments quickly.

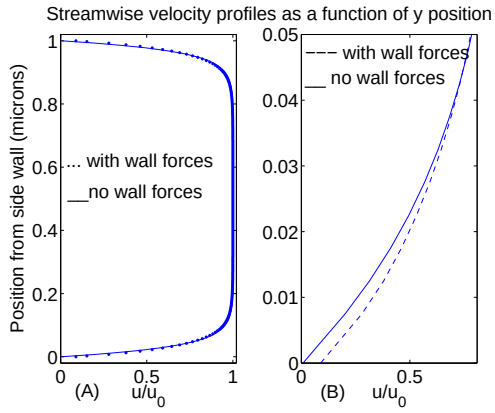

Figure 7: Normalized streamwise velocity profiles along the y-direction at cross-section $x = 1\,\mu m$ for $z = 50\,nm$. The solid lines are the velocity profiles when no wall forces are used. The dotted or dashed line is the case where wall forces are introduced. The graph (A) is normalized velocity at $z = 50\,nm$ as a function of distance from the side wall (y-direction). The graph (B) shows the normalized velocity profile near the side channel wall.

Simulation for higher resolution would take much longer time [3]

With a dedicated cluster, our parallel code achieves almost full linear speedup when varying the number of nodes. The speedup is 18.97 with 20 nodes. Our focus in the experiments is to study the performance of the filtered dynamic lattice mapping and compare it with other approaches under two types of workload: 1) *Fixed slow nodes:* There is a fixed set of nodes which are shared by other jobs. In this setting, a background job runs on each selected node taking 70% CPU resource. 2) *Transient spikes:* A node is randomly chosen and a background job is executed which takes 70% of CPU resource for a length varying from 1 to 4 seconds. After 10 seconds, another node is randomly chosen to execute a background job. This random process repeats every 10 seconds.

[3] A real application can require about 500,000 LBM phases to reach the steady-state. Our current application setting represents the simplest possible geometry, the smallest physical dimensions, and the coarsest resolution necessary to study the slip phenomenon. To study the effects of slip in a typical MEMS device would require substantially more computation.

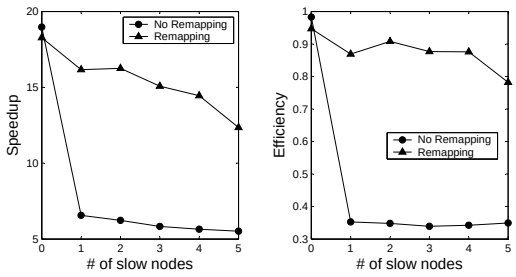

Figure 8: Speedup and normalized efficiency with 20,000 phases.

4.2.1 Overall Performance with Fixed Slow Nodes

The left part of Figure 8 shows the speedup of the parallel code with filtered dynamic remapping when the number of slow nodes varies from 0 to 5. The speedup is defined as the execution time of the sequential program divided by the execution time of parallel program. When the cluster is dedicated, the speedup is close to 19. When there is one slow node, the speedup is about 16, whereas the parallel code without dynamic remapping performs poorly with more slow nodes and its speedup drops dramatically. With five slow nodes, the parallel code with filtered dynamic remapping can still achieve a speedup of 13.

We use the following normalized efficiency $\frac{Speedup}{20-0.3m}$ to examine the utilization under this non-dedicated cluster with m slow nodes in which a background job takes 70% CPU resource at each node. The right part of Figure 8 shows the normalized efficiency metric, indicating that our code can achieve very good resource utilization which is 90% when the number of slow nodes is less than four and 80% for five slow nodes. In comparison, the utilization is low without remapping.

4.2.2 Execution profiling of different approaches with one fixed slow node

We provide execution profile and cost distribution of different approaches with one slow node, which allows us to understand behavior and effectiveness of proposed optimization strategies. The following

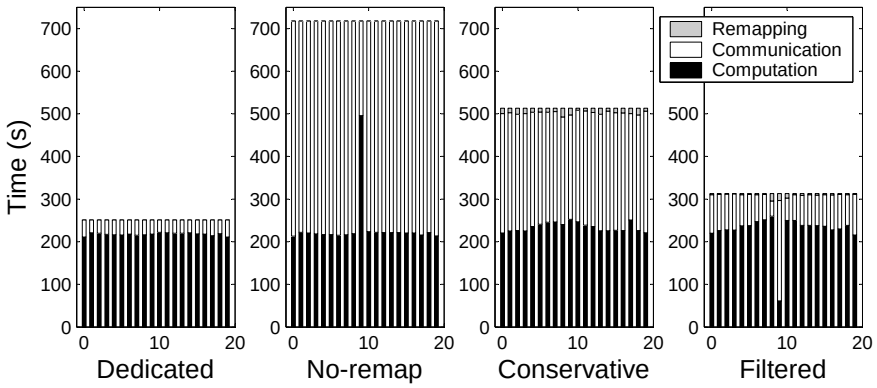

Figure 9: Execution profile and cost distribution for different schemes for 600 phases.

three approaches are compared with filtered remapping: 1) **Dedicated** scheme, where there is no slow node and dynamic data remapping is disabled; 2) **No-remapping** scheme, where one node is slow without dynamic data remapping; 3) **Conservative** scheme, which is the same as the filtered scheme except that over-redistribution is not used.

The results are shown in Figure 9 for 20 cluster nodes. This experiment takes 600 phases. With 20 dedicated nodes, the computation takes about 251 seconds. When no remapping is used and node 9 is a shared node, the total time increases from 251 seconds to 717 seconds and the increasing ratio is 185.6%. The "ripple" effect of a slow node drags the entire computation significantly.

The remapping method with conservative redistribution balances the computation nicely, but it does not effectively reduce the synchronization overhead caused by sluggish communication in node 9. The filtered approach aggressively redistributes the lattice points of node 9 to others and uses only 313.0 seconds. The overall parallel time increasing ratio is only 24.7% compared to the dedicated case. This method reduces the time of non-remapping by 56.3% and that of conservative redistribution by 39%. From this figure, one can see that filtered remapping moves most of the lattice points from node 9 to its neighbors using the over-redistribution strategy, and then it shifts these points further to other nodes. The slow node (node 9) accounts for most of the time spent in communication in this LBM-based simulation.

Finally, it should be noted that cost of remapping in both the filtered and conservative schemes is low, as the profile shows. That is because both of them use lazy remapping.

4.2.3 Comparison of different approaches with multiple slow nodes

We further compare filtered remapping with conservative redistribution and non-remapping when the number of slow nodes varies from 0 to five. In this comparison, we also include the performance of remapping using global information exchange, which tries to re-assign lattice points to nodes proportionally to their speeds. This global method employs lazy remapping but does not use the over-redistribution strategy.

Results are shown in Figure 10. The filtered approach performs best. It outperforms the conservative approach by up to 39%, and the no-remapping method by up to 57.8%. Global remapping performs well with one slow node; however after the number of slow nodes becomes more than 2, the global method is worse than the others. The reason is that the global approach incurs too much communication overhead, and slow nodes still take on too many lattice points.

4.2.4 Tolerance of transient spikes

In this experiment, we compare how different algorithms tolerate transient load spikes caused by back-

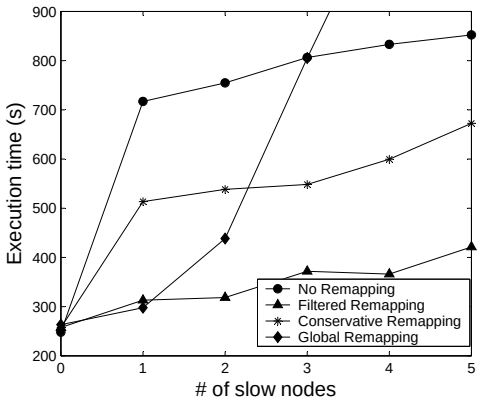

Figure 10: Execution time of 600 phases for different remapping techniques.

ground jobs executed on randomly-selected nodes every 10 seconds as we have discussed previously. The number of LBM phases for this experiment is 100.

Table 1 lists the execution slowdown ratio of four approaches compared to the dedicated case. The result shows that the filtered, conservative, and no-remapping methods have similar performance and are much better than global remapping. No-remapping performs well because every node has an equal chance to become slow, and there is no benefit to do re-balancing. The filtered method uses the lazy remapping strategy which can effectively tolerate the transient spikes. The conservative redistribution algorithm uses the same lazy remapping strategy and thus performs similarly. The performance of the global approach is worse than others because it involves too much global synchronization, which is sensitive to the existence of slow nodes.

Spike Length	No-Remap	Global	Filtered	Cons.
1 s	7.4%	5.8%	6.7%	10.9%
2 s	11.9%	37.2%	15.6%	16.0%
3 s	23.7%	40.9%	23.3%	24.9%
4 s	35.6%	49.5%	38.1%	39.8%

Table 1: Slowdown ratio of different algorithms compared to the dedicated case with various length of disturbance for 100 phases. Spike length is in seconds.

5 Concluding Remarks

We have parallelized the multicomponent lattice Boltzmann method for simulation of fluid slip in a microchannel. While domain decomposition can parallelize the code with a fairly good speedup in a dedicated environment, the long computation time and synchronization overhead at each phase of this computation require us to improve load balancing through dynamic data remapping. Our experiments have shown that the proposed filtered remapping technique can effectively re-distribute the workload among computing nodes and significantly reduce the overhead resulting from slow nodes in a partially heavily loaded cluster. For the tested cases, filtered remapping outperforms no-remapping by up to 57.8% and more compared to a global strategy. The over-redistribution strategy used in filtered remapping improves performance by up to 39% compared to a conservative approach.

Previous work in load balancing has studied process migration and scheduling independent of application characteristics, e.g. [10, 42, 45]. Another category of research in load balancing has focused on application-specific strategies. For example, Hamdi and Lee [12] have studied load redistribution for parallel image processing with a centralized approach. The main contribution of our load balancing work is a filtered approach which uses lazy remapping and over-redistribution of data from slow nodes by taking the application characteristic of LBM-based simulation.

Prediction of CPU usage has been studied in [44, 9, 46]. In a study of Unix load from a large set of computational settings, Dinda and O'Halloran [9] concluded that load is consistently predictable with simple linear models. In [46], it was shown that giving more weight to the most recent data can improve the accuracy of the load predictions. For fluid slip, instead of relying on the most recent performance data, we have used the harmonic average of the last several sampled times for usage prediction. This helps to minimize the impact of transient load spikes.

References

[1] J. Barrat and L. Bocquet. Large slip effect at a nonwetting fluid-solid interface. *Phys. Rev. Lett.*, 82:4671, 1999.

[2] P. L. Bhatnagar, E. P. Gross, and M. Krook. A model for collision processes in gases, I: small amplitude process in charged and neutral one-component system. *Phys. Rev.*, 94:511, 1954.

[3] N. F. Bunkin, O. A. Kiseleva, A. V. Lobeyev, and T. G. Movchan. Effect of salts and dissolved gas on optical cavitation near hydrophobic and hydrophilic surfaces. *Langmuir*, 13:3024–3028, 1997.

[4] D. Chandler. Two faces of water. *Nature*, 417:491, May 2002.

[5] S. Y. Chen, H. D. Chen, and W. M. D. Martinez. Phys. rev. lett. *Phys. Rev. Lett.*, 67:3776, 1991.

[6] C. H. Choi, K. J. A. Westin, and K. S. Breuer. Apparent slip flows in hydrophilic and hydrophobic microchannels. *submitted*, 2003.

[7] N. Churaev, V. Sobolev, and A. Somov. Slippage of liquids over lyophobic solid surface. *J. Colloid Interface Sci.*, 97:574, 1984.

[8] S. Z. D. H. Rothman. Lattice gas models of phase separation: interface, phase transitions and multiphase flow. *Rev. Mod. Phys.*, 66:1417, 1994.

[9] P. Dinda and D. O'Hallaron. An Evaluation of Linear Models for Host Load Prediction. In *Proc. of HPDC-8*, 1999.

[10] E. Frachtenberg, D. Feitelson, F. Petrini, and J. Fernandez. Flexible CoScheduling: Mitigating Load Imbalance and Improving Utilization of Heterogeneous Resources. In *IPDPS03*, April 2003.

[11] D. Grunau, S. Y. Chen, and K. Eggert. A lattice Boltzmann model for multiphase fluid flows. *Phys. of Fluids A*, 5:2557, 1993.

[12] M. Hamdi and C.-K. Lee. Dynamic Load Balancing of Data Parallel Applications on a Distributed Network. In *Proc. of 9th Sumpercomputing*, pages 170–179, 1995.

[13] M. Harchol-Balter and A. B. Downey. Exploiting Process Lifetime Distributions for Dynamic Load Balancing. In *Proceedings of the 1996 ACM SIGMETRICS*, pages 13–24, May 1996.

[14] X. He and L. S. Luo. Theory of lattice Boltzmann method: from the Boltzmann equation to the lattice Boltzmann equation. *Phys. Rev. E*, 56:6811, 1997.

[15] R. G. Horn, O. I. Vinogradova, M. E. Mackay, and N. Phan-Thien. Hydrodynamic slippage inferred from thin film drainage measurements in a solution of nonadsorbing polymer. *J. Chem. Phys.*, 112(14), April 2000.

[16] S. Hou. *Lattice Boltzmann Method for incompressible viscous flow*. PhD thesis, Kansas State University, 1995.

[17] D. M. Huang and D. Chandler. The hydrophobic effect and the influence of solute-solvent attractions. *J. Phys. Chem.*, 106:2047–2053, 2002.

[18] D. Kandhai, A. Koponen, A. Hoekstra, M. Kataja, J. Timonen, and P. Sloot. Lattice-Boltzmann Hydrodynamics on Parallel Systems. *Computer Physics Communications*, 111(1-3):14–26, 1998.

[19] K. Lum, D. Chandler, and J. D. Weeks. Hydrophobicity at small and large length scales. *J. Phys. Chem.*, 103:4570–4577, 1999.

[20] D. Lumma, A. Best, A. Gansen, F. Feuillebois, J. O. Radler, and O. Vinogradova. Flow profile near a wall measured by double-focus fluorescence cross-correlation. *Phys. Rev. E*, 112(14), 2000.

[21] L. S. Luo. Unified theory of the lattice Boltzmann models for nonideal gases. *Phys. Rev. Lett.*, 81:1618, 1998.

[22] L. Mainbaum and D. Chandler. A coarse-grained model of water confined in a hydrophobic tube. *J. Phys. Chem.*, 107:1189–1193, 2003.

[23] N. S. Martys and H. Chen. Simulation of multicomponent fluids in complex three-dimensional geometries by the lattice Boltzmann method. *Physical Review E*, 53(1):743, 1996.

[24] R. Pit, H. Hervet, and L. Leger. Direct experimental evidence of slip in hexadecane: Solid interfaces. *Phys. Rev. Lett.*, 85:980, 2000.

[25] Y. H. Qian. *Lattice gas and lattice kinetic theory applied to the Navier-Stokes equations*. PhD thesis, University Pierre et Marie Curie, Paris, 1990.

[26] E. Ruckenstein and P. Rajora. On the no-slip boundary condition of hydrodynamics. *J. Colloid Interface Sci.*, 96:488, 1983.

[27] G. D. D. S. Y. Chen. Lattice Boltzmann method for fluid flows. *Annu Rev. Fluid Mech.*, 30:329, 1998.

[28] M. Sakurai, H. Tamagawa, K. Ariga, T. Kunitake, and Y. Inoue. Molecular dynamics simulation of water between hydrophobic surfaces. Implication for the long-range hydrophobic force. *Chem. Phys. Lett.*, 289:567, 1998.

[29] N. Satofuka and T. Nishioka. Parallelization of lattice Boltzmann method for incompressible flow computations. *Comput. Mech.*, 23:164, 1999.

[30] X. Shan and H. Chen. Lattice Boltzmann model for simulating flows with multiple phases and components. *Physical Review E*, 47(3):1815, 1993.

[31] X. Shan and H. Chen. Simulation of nonideal gases and liquid-gas phase transitions by the lattice Boltzmann equation. *Phys. Rev. E*, 49:2941, 1994.

[32] X. Shan and G. D. Doolen. Multicomponent lattice Boltzmann model with interparticle interaction. *J. Stat. Phys.*, 81:379, 1995.

[33] P. A. Skordos. Parallel Simulation of Subsonic Fluid Dynamics on a Cluster of Workstations. In *Proc. of HPDC-4*, 1995.

[34] T. Suviola. Parallelization of a lattice Boltzmann suspension flow solver. *Applied parallel computing lecture notes in computer science*, 2367:603, 2002.

[35] P. A. Thompson and S. M. Troian. A general boundary condition for liquid flow at solid surfaces. *Nature*, 389(6649):360–362, 1997.

[36] D. C. Tretheway and C. D. Meinhart. Apparent fluid slip at hydrophobic microchannel walls. *Physics of Fluids*, 14(3), 2002.

[37] D. C. Tretheway and C. D. Meinhart. A generating mechanism for apparent fluid slip in hydrophobic microchannels. *submitted to Physics of Fluids*, August 2003.

[38] O. I. Vinogradona. Possible implications of hydrophobic slippage on the dynamic measurements of hydrophobic forces. *J. Phys: Condens. matter*, 8, 9491 1996.

[39] O. I. Vinogradova. Slippage of water over hydrophobic surfaces. *Int. J. Miner. Process.*, 56:31–60, 1999.

[40] K. Watanabe, Yanuar, and H. Mizunuma. Slip of Newtonian fluids at solid boundary. *JSME Int. J.*, 41:525, 1998.

[41] K. Watanabe, Yanuar, and H. Mizunuma. Drag reduction of Newtonian fluid in a circular pipe with a highly water-repellant wall. *J. Fluid Mech.*, 381:225, 1999.

[42] M. H. Willebeek and A. P. Reeves. Strategies for Dynamic Load Balancing on Highly Parallel Computers. *IEEE Trans. on Parallel and Distributed Systems*, 4(9), 1993.

[43] D. A. Wolf-Gladrow. *Lattice-gas cellular automata and lattice Boltzmann Models – an introduction*. Springer, Berlin, 2000.

[44] R. Wolski, N. Spring, and J. Hayes. Predicting the CPU Availability of Time-shared Unix System on the Computational Grid. In *Proc. of HPDC-8*, 1999.

[45] C. Xu, F. Lau, and R. Diekmann. Decentralized Remapping of Data Parallel Applications in Distributed Memory Multiprocessors. *Concurrency: Practice and Experience*, 9(12):1351–1376, Dec. 1997.

[46] L. Yang, I. Foster, and J. M. Schopf. Homeostatic and Tendency-based CPU Load Predictions. In *Proceedings of IPDPS 2003*, April 2003.

[47] L. Zhu, D. Tretheway, L. Petzold, and C. Meinhart. Simulation of fluid slip at hydrophobic microchannel walls by the lattice Boltzmann method. *submitted to J. Comput. Phys.*, 2003.

[48] Y. Zhu and S. Granick. Rate-dependent slip of Newtonian liquid at smooth surfaces. *Phys. Rev. Lett.*, 87, 2001.