Grid and HPC Dynamic Load Balancing with Lattice Boltzmann Models*

Fabio Farina[1,2], Gianpiero Cattaneo[1,2], and Alberto Dennunzio[1]

[1] Università degli Studi di Milano–Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione,
Via Bicocca degli Arcimboldi 8, 20126 Milano (Italy)
{farina, cattang, dennunzio}@disco.unimib.it
[2] INFN Milano–Bicocca, Piazza della Scienza 3, 20126 Milano (Italy)

Abstract. Both in High Performance Computing and in Grid computing dynamic load balancing is becoming one of the most important features. In this paper, we present a novel load balancing model based on Lattice Boltzmann Cellular Automata. Using numerical simulations our model is compared to diffusion algorithms adopted on HPC load balancing and to agent-based balancing strategies on Grid systems. We show how the proposed model generally gives better performances for both the considered scenarios.

Keywords: Lattice Boltzmann Models, Dynamic Load Balancing, diffusion algorithm.

1 Introduction

Dynamic load balancing is one of the most challenging features for the next generation of both Grid and High-performance computing (HPC) systems. The main differences in the formulation for the dynamic load balancing (DLB in the following) problem between these scenarios are related to the capabilities associated to each processing element and its network topology. In High-performance systems the computing elements are considered homogeneous and are connected according to highly regular topologies, in general as meshes, fat-trees or hypercubes [2]. On the contrary Grid infrastructures [9] assume that all the computing elements participating in a Virtual Organization can be very different both in terms of network connectivity and in performance capability.

Let us define an abstract distributed formulation for the DLB Problem that is suited for both the Grid and the HPC scenarios. We are given an arbitrary, undirected, connected graph $G = (V, E)$ describing a general distributed processor system. Two real numbers are associated to any processor $p \in V$: $\rho(p, t) \in \mathbb{R}^+$, the current work load at time t for p, and $c(p, t) \in \mathbb{R}^+$ representing the maximal sustainable work load for the i-th node at time t. Obviously for HPC systems

* This work has been supported by M.I.U.R. COFIN project "Formal Languages and Automata: Theory and Applications".

R. Meersman, Z. Tari et al. (Eds.): OTM 2006, LNCS 4276, pp. 1152–1162, 2006.

the capacity c is assumed as a constant, $\forall p \in V, \forall t \in \mathbb{N}, c(p,t) = \overline{c} \in \mathbb{R}^+$. A node is assumed to be able to communicate bidirectionally with any other node connected to it. The connections for every node are stated in the set $E \subseteq V \times V$. Communication between nonadjacent nodes is not allowed.

The goal is to determine a schedule to move an amount of work load across edges so that finally, the weight on each node is the closest as possible to the node capacity. The performance of a balancing algorithm can be measured in terms of number of iterations it requires to reach a stable state and in terms of the amount of load moved over the edge of the graph. The abstract DLB formulation holds when we associate a node with a processor, an edge with a communication link of unbounded capacity between two processors, and the weight u_i on each node can be divided into a very large amount of independent tasks.

In this paper we present a brand new approach for the DLB problem based on a Cellular Automata model known as Lattice Boltzmann [4]. This local model was originally created to simulate flow phenomena and to solve elliptic and parabolic PDEs [16]. Today Lattice Boltzmann Models are widely used to simulate complex systems such as multiphase turbolent flows and percolation [19].

The rest of the paper is organized as follows. Section 2 presents several commonly used approaches for the solution of the DLB problem. In Section 3 the Lattice Boltzmann model to solve the DLB problem is presented, while in Section 4 some numerical simulations of the new method are reported. In this section we will also show how the model operates efficiently both on Grid and on homogeneous distributed processors with respect to some other efficient DLB solvers. Finally in Section 5 conclusive notes are reported.

2 Related Work

Many different approaches have been used in the recent years to solve efficiently the DLB problem. These techniques take inspiration from numerical approximations and distributed Artificial Intelligence. In this section several successful DLB solvers are described, showing the highlights and drawbacks of each methodology.

2.1 Diffusion Equation Models

Diffusion algorithms assume that a node of the graph is exclusively able to send and receive messages to/from all its neighbors simultaneously. Most of the existing iterative DLB algorithms [7, 10, 17] involve three steps:

- **Diffusion Iteration:** flow balance calculation towards the equilibrium of the system.The diffusion iteration is a preprocessing phase which determines the actual load balance logic and rules the performance of the DLB solver.
- **Flow Calculation:** work load schedule preparation to migrate the amount of exceeding load between neighboring processors.
- **Task Migration:** deciding which particular tasks are to be migrated. This phase is particularly network intensive and is strongly application dependent.

Naturally, these models focus their efforts on the search of an effective way to reach a stable status during the Diffusion Iteration phase.

The original algorithm described in the Cybenko work [6], known as the Diffusion method, assumes the workload vector $\boldsymbol{p}$ over all the nodes $p \in V$ evolves according to the model

$$\boldsymbol{\rho}(t+1) = M\boldsymbol{\rho}(t)$$

where $\boldsymbol{\rho}(0)$ is the initial configuration and M is the diffusion equation governing the iterative process. The optimal matrix M is defined as $M = I - \frac{2}{\lambda_2+\lambda_n}L$ where I is the identity matrix, L is the Laplacian Matrix for the graph G and λ_2, λ_n are the larger and the smaller subdominant eigenvalues of L. Let us remind that $L = \mathrm{diag}(d_{pp}) - A$ where d_{pp} is the degree of node p, and A denotes the adjacency matrix of G. The Flow Calculation for the Diffusion method is performed solving the linear system $L\boldsymbol{d}(t) = \boldsymbol{\rho}(t) - \boldsymbol{c}(t)$, where $\boldsymbol{c}$ denotes the $c(p,t)$ values for each processor $p \in V$ and $\boldsymbol{d}(t)$ defines the schedule for the Task Migration [10]. It can be proved that the linear system solution can be performed online using an iterative schema similar to the one used for $\boldsymbol{\rho}(t)$, in particular $\boldsymbol{d}(t+1) = \boldsymbol{d}(t) + \frac{2}{\lambda_2+\lambda_n}\boldsymbol{\rho}(t)$.

Even in the case of the optimal matrix M, anyhow, the original Diffusion method lacks in performance because of its very slow convergence to the equilibrium.

A huge convergence acceleration is given by the Semi-iterative schema proposed in [11], which is somehow inspired by θ-schemas commonly used in numerical function extrapolations. In this algorithm the Diffusion Iteration is ruled by:

$$\boldsymbol{\rho}(t+1) = \sigma_{t+1}\left(\boldsymbol{\rho}(t) - \frac{2L\boldsymbol{\rho}(t)}{\lambda_2+\lambda_n}\right) + (1-\sigma_{t+1})\boldsymbol{\rho}(t-1) \tag{1}$$

$$\text{with} \quad \sigma_1 = 1, \quad \sigma_2 = \frac{(\lambda_2+\lambda_n)^2}{2\lambda_2\lambda_n}, \quad \text{and} \quad \sigma_{t+1} = \left(1 - \frac{(\lambda_2-\lambda_n)^2}{(\lambda_2+\lambda_n)^2}\cdot\frac{\sigma_t}{4}\right)^{-1} \tag{2}$$

The Flow Calculation step for the Semi-iterative schema requires the evaluation of the following flow potential: $\boldsymbol{d}(t+1) = \sigma_{t+1}\left(\boldsymbol{d}(t) + \frac{2}{\lambda_2+\lambda_n}\boldsymbol{\rho}(t)\right) + (1-\sigma_{t+1})\boldsymbol{\rho}(t-1)$. The use of semi-iterative techniques improves the rate of convergence of an order of magnitude with respect to the classical Diffusion method.

The main lack of Diffusion Models is that they are limited to the HPC systems only, where each node has the same characteristics and the same performances.

2.2 Swarm Intelligences

The Distributed Artificial Intelligence approach for DLB on Grid has been originally proposed in [12], thought they can be easily and effectively adapted to HPC context. The main idea behind this models derives from the artificial ants proposed to approximate efficiently NP-Complete problems solutions [15].

The swarm intelligence for DLB adopts a greedy schema divided in three phases:

- **SearchMax:** a migrating ant locates the most overloaded node;
- **SearchMin:** another other ant locates the less efficiently used node;
- **Transfer:** an ant migrates computational load from the most overloaded node to the most underloaded one.

In HPC or Grid environments a coalition of multiple mobile ants is considered. Each ant takes care of a computational workload granule. Ants have to exploit nodes and federate themselves into small teams with a fixed maximum size. Teams are useful both to save time during the SearchMax/SearchMin phases and to improve the transfer bandwidth. An ant can join or leave teams according the trails left behind by the other ants in the team. Such trails inform the newcomer about the load of the node currently hosting the team. Once the ants are clustered onto the most overloaded and on the most underloaded sites, then the transfer phase can be performed.

The main advantage of the outlined load balancing mechanism is that it requires no global knowledge of the node topology w.r.t. the Diffusion algorithms. Moreover, the direct migration among the nodes does not require a migration schedule calculation. The main drawbacks of swarm intelligence DLB solvers is that the ants may introduce further work load for the system. Such overload could imply a significant overhead for a very high number of agents. The convergence performance of the method is also very sensitive to the work load migration capabilities of ants: a small migration capacity would bring to a large number of ants overcrowding the system. Instead, a too large transfer capacity would imply a very slow convergence rate, with a small number of nodes swapping each other loads repeatedly [20].

3 The Model

Lattice Boltzmann Models (LBM) have been widely used to simulate multiphase fluid flows, porous media and many other complex systems [4] in the last decade. Such Cellular Automata simulate the mesoscopic behavior of the fluid inside an automaton cite. If the evolution rule preserves physical quantities, such as the local mass density and the local momentum density, theoretically it can be proved that macroscopic behavior emerges conformant to PDEs describing the simulated physical phenomena.

The proposed LBM solver assumes that the nodes V in the system graph G introduced in Section 1 are connected according to a regular topology with dimension d defining the set of the edges E (e.g. a ring topology for $d = 1$, a torus for $d = 2$, an n-cube for higher values of d). In HPC systems this configuration can be easily obtained remapping the physical network connections logically among the processing elements. In Grid systems a ring topology can be efficiently obtained constructing a peer-to-peer overlay among the worker nodes. Recently some efficient algorithms for the autonomous organization of nodes in overlays have been proposed [1]. The DLB Lattice Boltzmann solver is an iterative process similar to the Iterated Dynamical Systems adopted by the diffusion equation models described in Section 2. Though, as the multiagent approaches, it is not

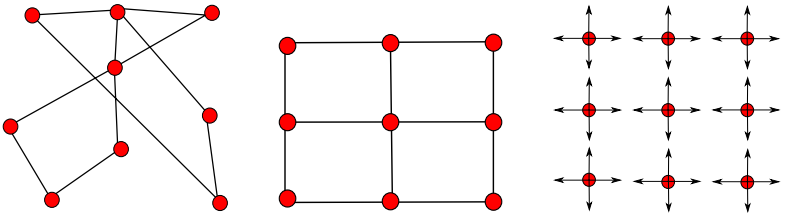


Fig. 1. Nodes arrangement from the physical network topology (a) to a regular overlay network (b) which is used as support to identify the neighbors of each node through the vectors $\{\boldsymbol{e}_i\}$ (c)

necessary to solve a migration scheduling problem because the information in the automaton cells state how the work load has to be transferred.

In general, we assume n automaton cells, corresponding to the computing nodes, are organized in a d-dimensional space so that each node $p \in V$ has exactly degree $2d$. In this way a cell is connected to its $2d$ neighbors along the processor links. The links are modelled as directions corresponding to $2d$ unary link vectors $\boldsymbol{e}_i$. For example, in the case of $d = 2$ the link vectors are

$$\boldsymbol{e}_0 := (0,0) \qquad \boldsymbol{e}_i := \left(\cos\left(\frac{\pi(i-1)}{2}\right), \sin\left(\frac{\pi(i-1)}{2}\right)\right), i \in \{1,2,3,4\} \quad (3)$$

where the vector $\boldsymbol{e}_0$ corresponds to the central position of each cell.

The set $\{\boldsymbol{e}_i\}$ for a generic node $p \in V$ is constructed assigning a node p' s.t. $(p,p') \in E$ to one of the vectors $\boldsymbol{e}_i$. In this way the processor graph can be mapped into a lattice suited for the LBM computation. We also introduce the nearest neighbor function $NN : V \times \{\boldsymbol{e}_i\} \to V$ that associates to each node p the node p' connected to it through a direction vector $\boldsymbol{e}_i$. An example for $d = 2$ of this mapping is reported in Figure 1. A generic cell state is characterized by $2d$ real number quantities modelling the load a processing element is going to move toward a neighbor along its links. More formally, given a processor $p \in V$, at time t its load configuration is defined as $\boldsymbol{f}(p,t) = \{f_i(p,t) : i = 0,1,\ldots,2d\}$ where $f_i(p,t) \in \mathbb{R}^+$ are called load distributions and represent the work load that will be moved along the link i. On the basis of these quantities we evaluate the two variables $\rho(p,t)$ and $c(p,t)$ according to the following equations:

$$\rho(p,t) = \sum_{i=0}^{2d} f_i(p,t) \qquad \text{and} \qquad c(p,t) = \Pi(p,t) \quad (4)$$

where, as stated in Section 1, $\rho(p,t)$ is the total load for a node $p \in V$ at time t and $c(p,t)$ is the maximum node capability with respect to a generic performance measure $\Pi : V \times \mathbb{N} \to \mathbb{R}^+$.

The time-step automaton evolution consists of two phases, according to the common LBM stream-and-collide algorithm:

– *Collision*: the load within a cell is redistributed among the $2d$ directions and the rest load distribution. Original distributions $\boldsymbol{f}(p,t) = \{f_i(p,t) : i = 0, 1, \ldots, 2d\}$ are modified in $\boldsymbol{f}^*(p,t) = \{f_i^*(p,t) : i = 0, 1, \ldots, 2d\}$ applying an instantaneous transition, formalized by the following equation:

$$f_i^*(p,t) = f_i(p,t) + \Omega_i(p,t) \tag{5}$$

where the *collision operators* $\Omega_i(p,t)$ must satisfy the following conservation law:

$$\rho^*(p,t) = \sum_{i=0}^{2d} f_i^*(p,t) = \sum_{i=0}^{2d} f_i(p,t) = \rho(p,t) \tag{6}$$

Equivalently this corresponds to $\sum_{i=0}^{2d} \Omega_i(p,t) = 0$. That is, the work load on each node is conserved during the redistribution. In the classical LBM context the Equation (6) corresponds to the mass conservation principle.

– *Streaming*: the load distributions move along their direction towards adjacent cells/processors in the network. This transfer is described in the automaton by the following equation:

$$f_i(\mathrm{NN}(p, \boldsymbol{e}_i), t+1) = f_i^*(p,t) \tag{7}$$

where the function NN is the previously defined nearest neighbor function, which associates a node p with another node along the direction $\boldsymbol{e}_i$.

Composition of the two Equations. (5) and (7) leads to an evolution rule similar to the *Lattice Boltzmann Equation*:

$$f_i(\mathrm{NN}(p, \boldsymbol{e}_i), t+1) = f_i(p,t) + \Omega_i(p,t) \tag{8}$$

We use a common form of the collision operator Ω_i, known as BGK formulation (see[13, 3, 14]):

$$\Omega_i(f_i(p,t)) = \frac{1}{\tau}\left(f_i^{eq}(p,t) - f_i(p,t)\right) \tag{9}$$

where τ is called the *relaxation time* of the model. It has been proved theoretically that the LBMs with BGK collision operator are unconditionally unstable for $\tau < 1/2$ [18]. Thus, the *evolution equation* of a LBM is

$$f_i(\mathrm{NN}(p, \boldsymbol{e}_i), t+1) = f_i(p,t) + \frac{1}{\tau}\left(f_i^{eq}(p,t) - f_i(p,t)\right) \tag{10}$$

The choice for the specific form for the equilibrium distributions f_i^{eq} is

$$f_0^{eq}(p,t) = (1 - 2d\,\alpha(p,t))\;\rho(p,t) \qquad \text{and} \qquad f_i^{eq}(p,t) = \alpha(p,t)\;\rho(p,t) \tag{11}$$

where $\alpha(p,t)$ is an over-relaxation term [8] introduced to speed up the convergence:

$$\alpha(p,t) = \frac{1}{2d}\left(\rho(p,t) - c(p,t)\right) - b\left(1 - \frac{\rho(p,t)}{c(p,t)}\right) \tag{12}$$

with $b \ll 1$ over-relaxation constant ruling the numerical stability. Using a Chapman-Enskog expansion, it can be proved that the model described above approximates a continuous diffusion equation up to the second order, where the diffusion constant is proportional to the parameter τ [4].

Even though the synchronous automaton evolution is more suited for typical HPC all-to-all optimized communications, it is possible to reproduce the discrete time stepping on Grid systems using the WsNotification standard [5] or analogous Grid messaging systems.

4 Numerical Experiments

In this section we will present numerical experiments results for the LBM solver. These results will be compared to the solutions by both diffusive schemas and swarm intelligence proposed in Section 2. After the comparison we will state our conclusion. For these numerical insights a ring processor graph is taken in account. Ring topology has been chosen because of the lowest connectivity as possible and also because it is the most frequent and natural overlay topology adopted by peer-to-peer Grid systems.

In the first set of simulations the LBM solver performances are compared w.r.t. the Semi-iterative schema on an HPC system. The load balance will be calculated for a number of processors ranging from 32 up to 1024. The maximum allowed number of iterations is 1000. The initial load distribution on the nodes will be generated randomly. The convergence criterion for each test is $err(t) = \sum_{p \in V}(\rho(p,t) - c(p,t)) < \epsilon$, with $\epsilon = 10^{-3}$. For the comparison between the Semi-iterative schema and the LBM solver we considered the following system capacity function $c(p,t) = \overline{c} = \sum_{p \in V} \rho(p,0)/|V|$.

For each test the optimal values of the models parameters are considered. In Table 1 the two iterative schemas are compared by the number of iterations required to reach the steady state $c(p,t)$ according to the criterion stated above. It is particularly interesting to notice how the Lattice Boltzmann approach scales better than the accelerated diffusion model for larger systems. Moreover we have not yet found a formal relation between the network topology and the LBM relaxation time τ, so we had to tune it numerically for each considered graph, while the theoretical optimal parameters for the diffusion models are well-known (see Section 2).

As the LBM solver and the Semi-iterative schema are different numerical approaches to the same class of models, we can focus our attention on the way they damp load distribution frequencies during the diffusion. Studying the dynamical evolution of the distributions toward the equilibrium it has been noticed that the LBM solver results to be faster thanks to its capabilities of damping low wavelengths. In particular, the diffusion process takes advantage from the adaptive parameter $\alpha(p,t)$, defined in Equation (12), that keeps on pushing the exceeding load away from the nodes. This adaptive schema grants, in general, that the convergence ratio does not slow down, even if the nodes are slightly out-of-equilibrium. Instead, the convergence parameter for the Semi-iterative schema of Equation (2),

Table 1. Number of iterations for the LBM and the Semi-iterative schema

# of nodes	LBM	Semi-iterative
32	32	20
64	55	43
128	112	93
256	144	198
512	313	420
1024	783	897

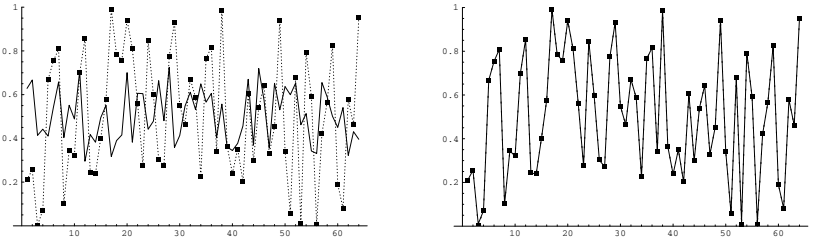
Table 2. LBM performance w.r.t. Semi-iterative schema for strongly biased initial configuration

# of nodes	LBM	Semi-iterative
32	103	20
64	193	39
128	342	77
256	719	152
512	>1000	303
1024	>1000	611

depending on the time only, seems to become less efficient as the load distribution becomes smoother and the low frequencies components become dominant.

The second set of tests compares the performance between the LBM and the Semi-iterative schema in the case of a strongly biased initial condition. In particular, we set only one node with most of the load in the system, while the others are strongly under-loaded. Under this condition the Semi-iterative schema performs much better than the proposed LBM model. More investigation on this singularity will be performed in future, even thought we are quite confident that this abnormal behaviour is related to the poor high frequency damping capabilities exhibited by all the Lattice Boltzmann cellular automata [18, 4]. Some results for these tests are reported in Table 2.

For the Grid scenario we compared the LBM solver approach with the DLB solution proposed by the Swarm intelligence model described in Section 2. Three agents have been assigned to each node in the system: SearchMin, SearchMax and Transfer agents respectively. The agent teams maximum size has been chosen so that it is proportional to the number of nodes in the graph. E.g., for $|V| = 32$ we found, after some empirical tests, that teams of at most 15 agents give best results without introducing agent overcrowding overhead. As a general rule, we noticed that a number of agents of about half of the computational nodes gives the best convergence speed. The network connecting the nodes is still arranged as a ring. The initial load configuration is generated randomly and also the node capacities are chosen randomly. The initial load configuration, in particular, is normalized to the nodes capacities, so that at the steady state the system is fully saturated. In Figure 2 we reported the initial and the final configurations



(a) (b)

Fig. 2. Initial configuration with system capacity (a). Steady state configuration reached in about 50 iterations (b).

Table 3. LBM performance w.r.t. swarm intelligence for a Grid environment

# of nodes	LBM	Swarm
32	43	150
64	52	269
128	113	399
256	210	737
512	404	>1000
1024	795	>1000

for a graph containing 64 nodes. On the abscissa axes the node ID is reported, while on the y axes we reported the work load assigned to each node. In the initial configuration picture we reported also the system capacities (dashed line). Analogous results are obtained with a swarm intelligence schema, but the LBM convergence takes a significantly lower number of iterations. Some results on the convergence speed are reported in Table 3.

The reasons for the different convergence speed could be searched in the LBM solver capabilities of calculating the load transfer without the need of moving agents across the network. Furthermore, fixed payload moved by agents can affect the swarm approach in a significant way: a too small payload can slow down the convergence but transferring a large work load with each agent could jeopardise the whole balancing process. In the worst case a too large payload could bring the balancing to a starvation, with the nodes swapping the work load forever. For these particular tests we tried to tune the payload for each agent so that the better convergence performance are obtained. Further investigation should be performed to better understand the swarm intelligence convergence performances with respect to both the used number of agents and the quantity of computational load each agent can move across the system.

Both the LBM and the Swarm intelligence DLB solver exhibit a proper behavior even in the case of globally overloaded environments. That is, they distribute the exceding load equally even in the case of low system capacity. An example of such behavior is shown in Figure 3. Let us remind that the node IDs are reported on the x axes and that on the y axes the work load is reported.

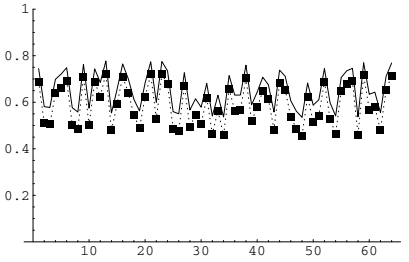


Fig. 3. Steady state reached by the LBM solver in the case of a globally overloaded system

5 Conclusion

In this paper, we have introduced a novel DLB solver which is inspired by the Lattice Boltzmann cellular automata usually used to simulate complex fluids dynamics. In particular, we modified the original LBM to approximate a diffusive phenomenon that suitably solves the DLB problem. We compared the performances of the proposed LBM solver with the ones of an accelerated diffusion equation model usually adopted in DLB on HPC systems. The model has been also compared to swarm intelligence models commonly adopted in Grid context load balancing problems. For both the comparisons it has been shown how the LBM behaves more efficiently than the standard approaches to the DLB problem. Only for ad-hoc initial configurations the LBM always presents worse performances than the other methods.

References

[1] D. Angluin, J. Aspnes, J. Chen, Y. Wu, and Y. Yin, *Fast construction of overlay networks*, Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, 2005, pp. 145–154.

[2] D.P Bertzekas and J.N. Tsitsiklis, *Parallel and distributed computing, 2nd edition*, Athena Scientific, NH, 1997.

[3] H. Chen, S. Chen, and W. Matthaeus, *Recovery of the Navier Stokes equation using a Lattice Gas Boltzmann method*, Physical Review A **45** (1992), 5339–5342.

[4] B. Chopard and M. Droz, *Cellular automata modelling of physical systems*, Cambridge University Press, Cambridge, 1998.

[5] OASIS Consortium, *Ws-notification specification 1.3 (pubblic draft 2)*, 2006, available at http://www.oasis-open.org/committees/wsn.

[6] G. Cybenko, *Dynamic load balancing for distributed memory multi-processors*, Journal of Parallel and Distributed Computing **7** (1989), 279–301.

[7] R. Diekmann, A. Frommer, and B. Monien, *Efficient schemes for nearest neighbor load balancing*, Parallel Comput. **25** (1999), 789–812.

[8] H. Fang, R. Wan, and Z. Lin, *Lattice boltzmann model with nearly constant density*, Physical Review E **66** (2002), 036314.

[9] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke, *Grid Computing: Making the Global Infrastructure a Reality*, ch. The Physiology of the Grid, pp. 217–249, Wiley, 2003.

[10] Y.F. Hu and R.J. Blake, *An improved diffusion algorithm for dynamic load balancing*, Parallel Computing **25** (1999), 417444.

[11] G. Karagiorgos and N.M. Missirlis, *Accelerated diffusion algorithms for dynamic load balancing*, Information Processing Letters **84** (2002), 61–67.

[12] A. Montresor, H. Meling, and O. Babaoglu, *Messor: Load-balancing through a swarm of autonomous agents*, Tech. report, Dept. of Computer Science, University of Bologna, 2002.

[13] E. P. Gross P. L. Bhatnagar and M. Krook, *A model for collision processes in gases. I. Small amplitude processes in charged and neutral one component systems*, Physical Review **94** (1954), 511–525.

[14] Y. H. Qian, D. D'Humieres, and P. Lallemand, *Lattice BGK models for Navier-Stokes equation*, Europhysics Letters **17** (1992), 479–484.

[15] M. Resnick, *Turtles, termites, and traffic jams: explorations in massively parallel microworlds*, MIT Press, Cambridge, MA, USA, 1994.

[16] D. Rothman and S. Zaleski, *Lattice-gas cellular automata: Simple models of complex hydrodynamics*, Cambridge University Press, UK, 1997.

[17] K. Schloegel, G. Karypis, and V. Kumar, *Multilevel diffusion schemes for repartitioning of adaptive meshes*, Journal of Parallel and Distributed Computing **47** (1997), 109124.

[18] J. D. Sterling and S. Chen, *Stability analysis of Lattice Boltzmann methods*, Journal of Computational Physics **123** (1996), 196–206.

[19] Sauro Succi, *The lattice Boltzmann equation, for fluid dynamics and beyond*, Oxford University Press, UK, 2001.

[20] Y. Wang, J. Liu, and X. Jin, *Modeling agent-based load balancing with time delays*, IAT '03: Proc. of the IEEE/WIC Int. Conf. on Intelligent Agent Technology, IEEE Computer Society, 2003, pp. 189–196.