Graphics processing unit implementation of lattice Boltzmann models for flowing soft systems

Massimo Bernaschi and Ludovico Rossi
Istituto Applicazioni Calcolo, CNR, Viale Manzoni 30, 00185 Rome, Italy

Roberto Benzi and Mauro Sbragaglia
University of Tor Vergata and INFN, via della Ricerca Scientifica 1, 00133 Rome, Italy

Sauro Succi
Istituto Applicazioni Calcolo, CNR, V. dei Taurini 9, 00185 Rome, Italy and Freiburg Institute for Advanced Studies, Freiburg, Germany
(Received 23 September 2009; published 30 December 2009)

A graphic processing unit (GPU) implementation of the multicomponent lattice Boltzmann equation with multirange interactions for soft-glassy materials ["glassy" lattice Boltzmann (LB)] is presented. Performance measurements for flows under shear indicate a GPU/CPU speed up in excess of 10 for 1024^2 grids. Such significant speed up permits to carry out multimillion time-steps simulations of 1024^2 grids within tens of hours of GPU time, thereby considerably expanding the scope of the glassy LB toward the investigation of long-time relaxation properties of soft-flowing glassy materials.

DOI: 10.1103/PhysRevE.80.066707 PACS number(s): 83.10.−y

I. INTRODUCTION

The rheology of flowing soft systems, such as emulsions, foams, gels, slurries, colloidal glasses, and related complex fluids, is a subject of increasing importance in modern nonequilibrium thermodynamics, with a broad range of applications in fluid dynamics, chemistry, and biology. From the theoretical standpoint, flowing soft systems are challenging because they do not fall within any of three conventional states of matter, gas-liquid-solid, but live rather on a moving border among them. Foams are typically a mixture of gas and liquids, whose properties can change dramatically with the changing proportion of the two; wet foams can flow almost like a liquid, whereas dry foams may conform to regular patterns, exhibiting a solidlike behavior. Emulsions can be paralleled to biliquid foams, with the minority species dispersed in the dominant (continuous) one. The behavior and, to same extent, the existence itself of both foams and emulsions are vitally dependent on surface tension, namely the interactions that control the physics at the interface between different phases/components. Indeed, the presence of surfactants, i.e., a third constituent with the capability of lowering surface tension, has a profound impact on the behavior of foams and emulsions. By lowering the surface tension, surfactants can greatly facilitate mixing, a much sought for property in countless practical endeavors, from oil recovery to chemical and biological applications. Living, as they do, out of equilibrium, these materials exhibit a number of distinctive features, such as long-time relaxation, anomalous viscosity, aging behavior, which necessitate profound extensions of nonequilibrium thermodynamics [1]. The study of these phenomena sets a pressing challenge for computer simulations as well, since characteristic time lengths of disordered fluids can escalate tens of decades over the molecular time scales. Among a variety of numerical methods for complex flows, both atomistic and macroscopic, mesoscopic lattice Boltzmann models have recently been developed, which prove capable of reproducing a number of qualitative features of soft-flowing materials, such as slow relaxation, dynamical heterogeneities, aging and others [2]. These models are based on suitable generalizations of the multicomponent Shan-Chen scheme for nonideal fluids, with multirange competing interactions, namely, short-range attractions (standard Shan-Chen) plus midrange repulsion. The competition between short-range attraction and midrange repulsion lies at the heart of the very rich behavior of the density field. Owing to this complexity, and particularly the slow relaxation properties, the investigation of the dynamical behavior of these systems requires very long time integrations, typically of the order of tens of million of time steps (as a reference, one lattice Boltzmann (LB) time step can be taken of the order of 100–1000 molecular-dynamics time steps). As a result, even if the extended LB code, *per se*, is not particularly more demanding than a standard Shan-Chen version, this need of a very long simulation span sets a strong incentive for efficient implementations. In this work, we discuss the implementation of this extended LB model on graphics processing unit (GPU) architectures and provide a few examples of the very significant CPU time gains versus the corresponding CPU implementations.

II. MODEL

The kinetic lattice Boltzmann equation takes the following form [3]:

$$f_{is}(\vec{r}+\vec{c}_i\Delta t,t+\Delta t)-f_{is}(\vec{r},t)=-\frac{\Delta t}{\tau}[f_{is}(\vec{r},t)-f_{is}^{(eq)}(\vec{r},t)]+F_{is}\Delta t, \quad (1)$$

where f_{is} is the probability of finding a particle of specie s at site $\vec{r}$ and time t, moving along the ith lattice direction defined by the discrete speeds $\vec{c}_i$ with $i=0,\ldots,b$. The left-hand side of Eq. (1) stands for molecular free streaming, whereas the right-hand side represents the time relaxation (due to collisions) toward local Maxwellian equilibrium on a time scale τ and F_{is} represents the volumetric body force due to intermolecular (pseudo-) potential interactions.

The pseudopotential force within each species consists of an attractive (a) component, acting only on the first Brillouin region (belt, for simplicity) and a repulsive (r) one acting on both belts, whereas the force between species (X) is short ranged and repulsive,

$$\vec{F}_s(\vec{r},t) = \vec{F}_s^a(\vec{r},t) + \vec{F}_s^r(\vec{r},t) + \vec{F}_s^X(\vec{r},t),$$

where

$$\vec{F}_s^a(\vec{r},t) = -G_s^a \Psi_s(\vec{r},t) \sum_{i\in belt1} w_i \Psi_s(\vec{r}_i,t)\vec{c}_i,$$

$$\vec{F}_s^r(\vec{r},t) = -G_s^r \Psi_s(\vec{r},t) \sum_{i\in belt1} p_i \Psi_s(\vec{r}_i,t)\vec{c}_i - G_s^r \Psi_s(\vec{r},t) \sum_{i\in belt2} p_i \Psi_s(\vec{r}_i,t)\vec{c}_i,$$

$$\vec{F}_s^X(\vec{r},t) = -\frac{1}{\rho_0^2}\rho_s(\vec{r},t) \sum_{s'\neq s} \sum_{i\in belt1} G_{ss'} w_i \rho_{s'}(\vec{r}_i,t)\vec{c}_i. \quad (2)$$

In the above, the groups "belt 1" and "belt 2" refer to the first and second Brillouin zones in the lattice and $\vec{c}_i$, p_i, and w_i are the corresponding discrete speeds and associated weights. Also, $G_{ss'}=G_{s's}$, $s'\neq s$, is the cross coupling between species, ρ_0 a reference density to be defined shortly, and finally, $\vec{r}_i=\vec{r}+\vec{c}_i\Delta t$ are the displacements along the $\vec{c}_i$ velocity vector. The first belt is discretized with nine speeds, whereas the second with 16 for a total of $b=24$ connections, plus a rest particle. The weights are chosen in such a way as to fulfill the following normalization constraints [4]: $w_0+\Sigma_{i\in belt1}w_i=p_0+\Sigma_{i\in belt1}p_i+\Sigma_{i\in belt2}p_i=1$, $\Sigma_{i\in belt1}w_i c_{ix}^2 = \Sigma_{i\in belt1}p_i c_{ix}^2+\Sigma_{i\in belt2}p_i c_{ix}^2=c_s^2$, $c_s^2=1/3$ being the lattice sound speed. The pseudopotential Ψ_s is taken in the form first suggested by Shan and Chen [5],

$$\Psi_s(\rho) = \rho_0(1-e^{-\rho/\rho_0}), \quad (3)$$

where ρ_0 marks the density value, at which nonideal effects come into play. Full details of the model and its continuum limit are provided in [6]. Here we shall just remind that a proper tuning of the couplings $G's$ as well as ρ_0 permits to realize a vanishingly small surface tension.

Due to the intrinsically slow relaxation of soft-glassy materials, the simulations entail very long time spans, covering several millions of time steps. This motivates the migration from CPU to GPU architectures.

III. GPU IMPLEMENTATION

The features of the NVIDIA graphics hardware and the related programming technology named CUDA are thoroughly described in the NVIDIA documentation [7]. Here, we report just the key aspects of the hardware and software we used.

Most of the simulations ran on a NVIDIA Tesla C1060 equipped with 30 multiprocessors with eight processors each, for a total of 240 computational cores that can execute at a clock rate of 1.3 GHz. The processors operate integer types and 32-bit floating point types (the latter are compliant with the IEEE 754 single-precision standard). Each multiprocessor has a memory of 16 Kbyte size shared by the processors within the multiprocessor. Access to data stored in the shared memory has a latency of only two-clock cycles allowing for fast nonlocal operations. Each multiprocessor is also equipped with 16 384 32-bit registers.

The total on-board *global* memory on the Tesla C1060 amounts to 4.0 Gbyte with a 512-bit memory interface to the GPU that delivers 102.4 Gbit/s memory bandwidth. The latency for the access to this global memory is approximately 200 cycles (two orders of magnitude slower than access to shared memory) with any location of the global memory visible by any thread, whereas shared memory variables are *local* to the threads running within a single multiprocessor.

For the programming of the GPU, we employed the CUDA Software Development Toolkit, which offers an *extended* C compiler and is available for all major platforms (Windows, Linux, Mac OS). The extensions to the C language supported by the compiler allow starting computational kernels on the GPU, copying data back and forth from the CPU memory to the GPU memory and explicitly managing the different types of memory available on the GPU.

The programming model is a single instruction multiple data (SIMD) type. Each multiprocessor is able to perform the same operation on different data 32 times in two clock cycles, so the basic computing unit (called *warp*) consists of 32 threads. To ease the mapping of data to threads, the threads identifiers may be multidimensional and, since a very high number of threads run in parallel, CUDA groups threads in *blocks* and *grids*.

One of the crucial requirements to achieve a good performance on the NVIDIA GPU is that global memory accesses (both read and write) should be coalesced. This means that a memory access needs to be *aligned* and coordinated within a group of threads. The basic rule is that the thread with id n ($\in 0,\ldots,N-1$) should access element n at byte address:

StartingAddress+sizeof(type)$*n$ where sizeof (type) is equal to either 4, 8, or 16 and StartingAddress is a multiple of 16$*$sizeof(type).

Although NVIDIA last generation hardware (such as the C1060 at our disposal) has better coalescing capability with respect to previous generations, the performance difference between fully coalesced memory accesses and uncoalesced accesses is still remarkable.

Functions running on a GPU with CUDA have some limitations: they cannot be recursive; they do not support *static* variables; they do not support variable number of arguments; function pointers are meaningless. Nevertheless, CUDA makes GPU programming much more simple as compared to other approaches such as that described in [8] where the Lattice Boltzmann method was implemented using directly the graphics operations provided by the hardware.

The porting of our multicomponent Lattice Boltzmann code for flowing soft systems to the GPU entailed some changes to the original code. First of all, the routines in charge of the LB update have been ported to CUDA and modified to better adapt to the GPU architecture, while additional routines were added to initialize and shutdown the CUDA module. All the CUDA routines have been integrated into the original code without modifying its structure, thus

maintaining compatibility with other code components (for instance, the I/O parts) and facilitating future updates to both the CUDA and the Fortran modules.

The initialization routine copies the data needed for executing the LB update from the CPU main memory to the GPU global memory. In the original Fortran code, the nine fluid populations of a lattice node are stored contiguously in memory (following the conventionally called array-of-structures layout), but using such layout on the GPU would force the threads to access global memory in an uncoalesced fashion, that is to violate the already mentioned *best practices* for GPU memory access patterns. Therefore, when copied to the GPU global memory, data are reordered following the structure-of-arrays layout [9,10], thus allowing coalesced accesses to the global memory. As a consequence the fluid populations of a lattice site are not contiguous in the GPU global memory. Data not modified during the simulation, such as coefficients, are precomputed during the initialization phase and stored in the GPU constant memory, which has performances analogous to those of registers if, as in our case, all the threads running on the same SIMD processor access the same constant memory location.

After the initialization phase, all the computation required for the LB update is performed on the GPU. A single step of the simulation is implemented through a sequence of CUDA kernels guaranteeing the correct sequential order of the substeps. Each CUDA kernel implements a substep of the update procedure (e.g., collision, streaming) by splitting the work among a configurable number of threads and blocks, which was fine tuned to achieve optimal performances, with respect to occupancy [11], on the CUDA devices available to us. Each thread works sequentially on a group of lattice nodes assigned to it. For each lattice node, the thread copies data from the global memory into registers, performs the computation and writes the results back in the global memory. In order to manage the parallelization of the streaming phase without causing conflicts among multiple threads, fluid populations are stored in the global memory using a double buffer policy [9,10]. At the end of the simulation, final results are copied back to the CPU main memory in order to be saved on a secondary storage device. Through a configuration file it is possible to require also the saving of partial results of the simulation at regular intervals (e.g., for check-pointing purposes).

Most of the global memory read and write operations are coalesced, with the exception of few read operations relative to the computation of the interaction forces and few write operations relative to the streaming phase. In the first case, the calculation of the force for a lattice node depends on values related to other lattice nodes, which must be loaded from global memory even if alignment requirements that allow coalesced accesses are not satisfied. In the second case, target locations of the streaming phase are defined by the lattice topology, and in general they do not comply with memory alignment requirements. In both cases, however, techniques that employ the GPU shared memory as a temporary buffer could be applied, in future versions of the code, to mitigate the overhead due to uncoalesced accesses.

For the function that computes the value of the hydrodynamic variables all memory operations are local, meaning that only the fluid populations of a lattice site are required and that the resulting hydrodynamic variables belong to the same lattice site. As a consequence, there is not a single uncoalesced memory access.

TABLE I. Timing (in seconds) required for 1000 iterations on different domains of increasing size. CPU timings were obtained on an Intel Xeon CPU E5462 at 2.80 GHz. GPU timings were obtained on a Tesla C1060 whose features are described in the text. Note that a typical run of one million time steps on a 1024^2 lattice requires about 3 weeks on a CPU and only less than 2 days on a GPU.

Domain size	CPU time	GPU time
128^2	13	6.6
256^2	77	16.5
512^2	429	48.5
1024^2	1740	145
2048^2	7050	533

Finally, as already mentioned, the fluid populations once uploaded on the GPU memory do not need to be copied back to the main memory unless a dump of the whole configuration is required. However, hydrodynamic variables might be written back to the main memory much more frequently since they represent the main physical output of the simulation. Although the number of hydrodynamic variables per lattice site is small compared to the number of fluid populations (there are four hydrodynamic variables vs nine fluid populations) so that the run time overhead of the copy from the GPU memory to the CPU memory is small compared to the initialization overhead, better speedups, with respect to the CPU version of the code, are obtained by reducing the number of these copy-back operations. In Table I, we report a comparison of the timings required by the CPU and GPU version of the code for different domains.

Profiling a code on a GPU has been a bit tricky for a long time but NVIDIA has recently introduced a visual profiler that makes easier to find out which *kernels* take more time. Table II shows data produced by such profiler for a typical run. What we found is that the breakdown of the time among the different GPU computational kernels is very similar to the CPU case were profiling can be easily done by using the *gprof* tool.

Other authors [12–14] proposed high-performance implementations of the lattice Boltzmann method for GPUs. To the best of our knowledge, all those implementations focus on the classic and general formulation of the lattice Boltzmann method (albeit they consider the possibility of having a different number of populations), whereas our code deals with the specific case in which the pseudopotential force within each species consists of two components and the force between species is short ranged and repulsive. The effect of more complex interactions on the speedup that is possible to achieve by using a GPU can be indirectly assessed by making a comparison between the speedup we obtain on the largest test case (2048^2), which is approximately 13.5, and the speed up reported in [12] for the plain D2Q9 method on a mesh of the same size, which is approximately 20. It is worth noting that we did not resort to the technique proposed

TABLE II. A sample of the data produced by the CUDA Visual Profiler for a typical run. The third and fourth column report the number of coalesced (i.e., optimal) load and store operations (for these routines there are no *uncoalesced* memory accesses). The fifth column reports the number of so-called *divergent* branches that are particularly bad for the performance. We managed to remove all of them from the most time consuming routines.

Method	%GPU time	gld_coherent	gst_coherent	Divergent_branch
Hydro	24.88	1.32096×10^8	2.10125×10^8	0
Colli	22.84	3.31776×10^7	7.74144×10^7	0
ExtForce	12.28	1.8432×10^7	5.16096×10^7	0
Move	12.09	1.42848×10^7	6.88128×10^7	0
Equil	11.31	4.3008×10^6	7.74144×10^7	0
Force AB	5.52	1.3068×10^6	5.2272×10^6	180
Force BB	5.45	653400	2.6136×10^6	180
Force AA	5.44	653400	2.6136×10^6	180

in [12] to improve the performance of the propagation phase of the LB update mainly because, as already mentioned, the NVIDIA card we used has better coalescing capability than the hardware used in [12]. In the near future we may experiment that approach but we do not expect a major change in the scenario.

Good performance are useless if the results of a simulation are not reliable. It is well known that GPU support for double precision is still in infancy stage (using double precision increases the GPU computing time of almost one order of magnitude) and all applications running on GPU make use of single-precision arithmetic. So it does our code. For an iterative method, such as the lattice Boltzmann, it is crucial to double check that long simulations do not produce (significantly) different results by using single and double precision arithmetic. To this purpose we ran a set of tests on the CPU since it was much simpler to switch from single to double precision and then we ran the same set of tests in single precision on the GPU. The result is that, on average, the percentage of mass/momentum loss due to the usage of the single precision corresponds, for the test cases described in the present study, to $\sim 0.5\times10^{-4}$.

IV. NUMERICAL RESULTS

We next proceed to present the results of numerical simulations. The baseline simulations are performed on a two-dimensional grid $N_x\times N_y=1024\times1024$. The two fluids are initialized at zero speed and random initial conditions for the two densities ρ_A and ρ_B. More specifically, after a preliminary tuning process, we choose $\rho_A=\rho_B=0.612$, with a standard deviation ±0.01 from the background density value. The reference density is taken as $\rho_0=0.7$. The couplings have been set to the following values (in lattice units):

$$G_A^a=-15.0,\quad G_A^r=14.1,$$

$$G_B^a=-14.0,\quad G_B^r=13.1,$$

$$G_{AB}=0.045. \tag{4}$$

These parameters secure that both components A and B are in the dense (liquid) phase. The relaxation time is fixed to $\tau=1$, corresponding to a kinematic viscosity $\nu=1/6$. The corresponding value of the surface tension is approximately $\sigma_{AB}\sim0.01$. All values are given in lattice units.

The fluid is subject to a periodic forcing $F_0\sin(2\pi ky/N_y)$ along the streamwise direction, x where $F_0=0.5\langle\rho\rangle\nu(2\pi k y/N_y)^2\sin(2\pi ky/N_y)$ and $\langle\rho\rangle$ is the average density. For a normally flowing fluid, such forcing would produce a sinusoidal flow with amplitude U_0. The effective viscosity of the flow is monitored through the ratio (response function) $R(t)=U(t)/U_0$, where $U(t)=(2/(N_xN_y))\Sigma_{x,y}\sin(2\pi ky/N_y)u(x,y;t)$ is the average flow speed projected upon the forcing. By definition, $R=1$ identifies standard flow conditions, whereas $R<1$ denotes enhanced effective viscosity, $\nu_{eff}=\nu/R$, due to caging effects

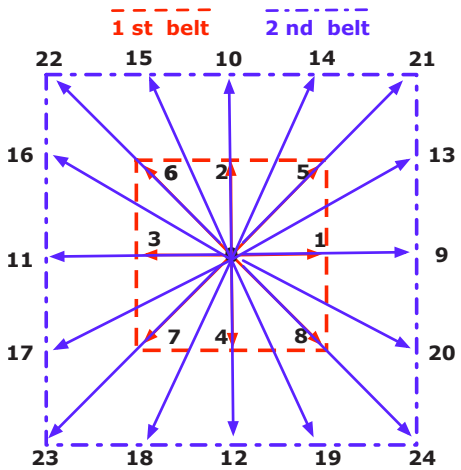

FIG. 1. (Color online) The two-belt 25-speed lattice used for the force evaluation. Each component experiences an attractive interaction in the first Brillouin zone and a repulsive one acting on both Brillouin zones. The integers refer to the square of the corresponding discrete velocity. Each of these interactions is controlled by a separate coupling constant.

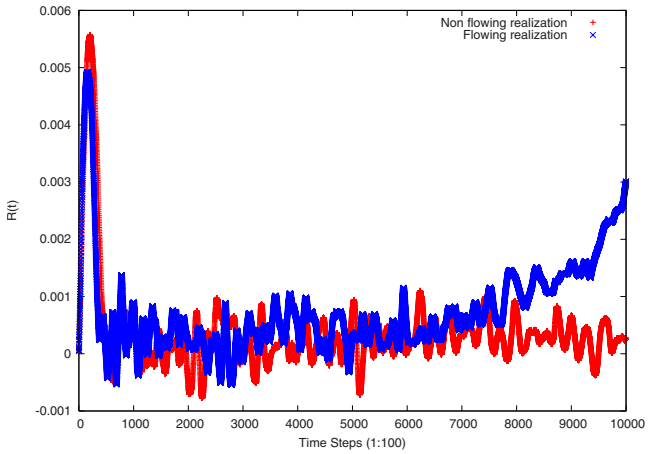

FIG. 2. (Color online) Time evolution of the response function for the case of a flowing (F, blue or dark gray line) and nonflowing system (N, red or light gray line), respectively.

and dynamic heterogeneities. In the present study, we have taken $U_0=0.01$. In previous studies, it was shown that, depending on initial conditions, forcing strength and surface tension, the system shows evidence of nearly arrested states, characterized by an effective flow speed much lower than U_0 (typically two to three orders). Since the occurrence of these

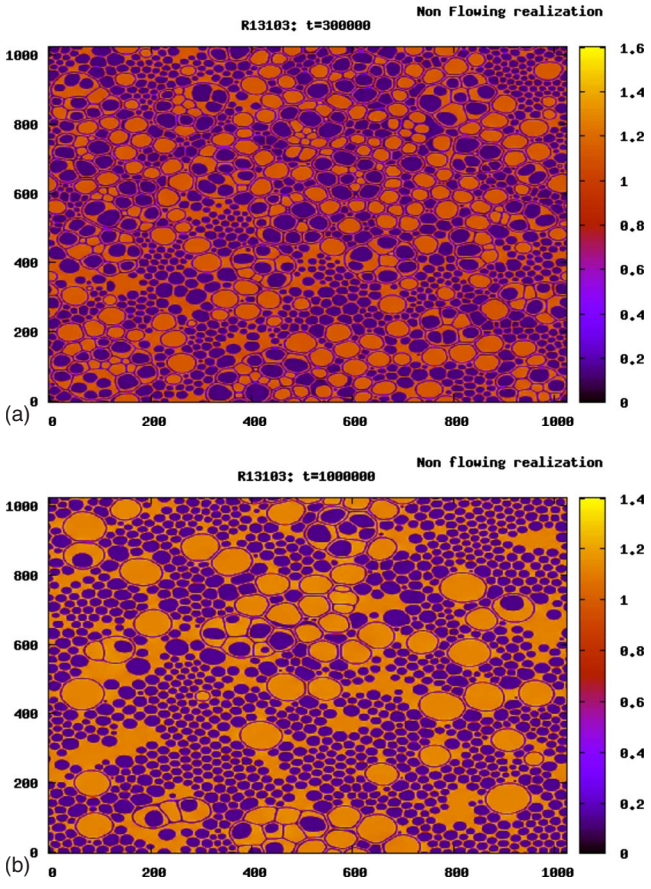

FIG. 3. (Color online) Contour plots of the density of fluid A for realization N at $t=3\times10^5$ (top panel) where a large number of cages is well visible and $t=10^6$ (bottom panel) where cages are still visible.

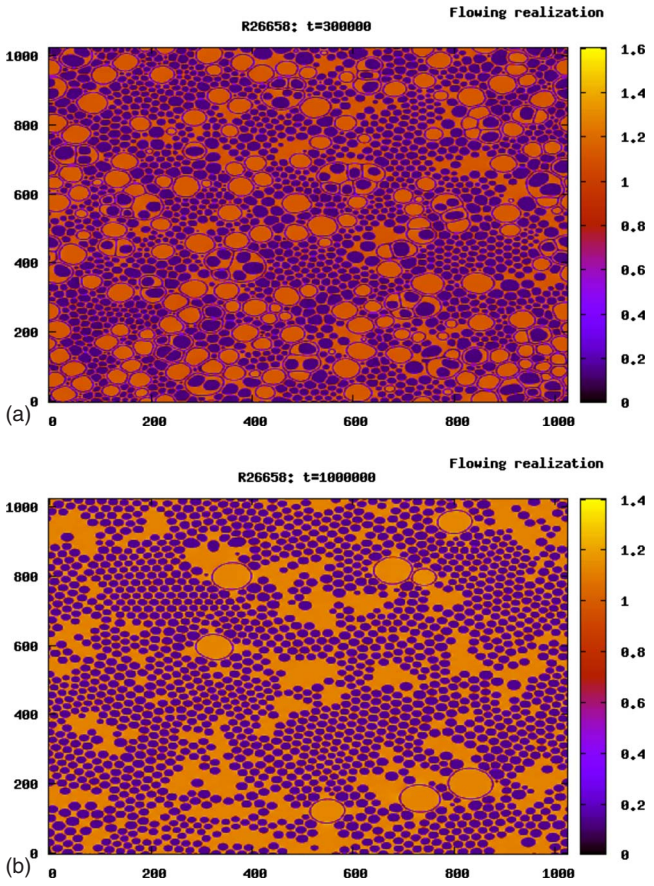

FIG. 4. (Color online) Contour plots of the density of fluid A for realization F at $t=3\times10^5$ (top panel) where cages are visible and $t=10^6$ (bottom panel) where most cages have disappeared, and the fluid can regain a flowing state.

arrested states is a prime signature of soft-glassy behavior, it is of great interest to investigate whether it still survives for systems of larger sizes. To this purpose, we have performed a number of simulations at grid resolution 1024^2, i.e., 8^2 times larger than our own previous simulations (Fig. 1). These simulations do indeed confirm the existence of such arrested states even in larger systems. A typical example of response function for the case of a flowing and arrested system is given in Fig. 2. The set of physical parameters is the same in the two cases, the only difference being a different realization of the random initial conditions. This figure shows that after an initial stage, where the fluid builds up a macroscopic speed under the effect of the external forcing, both realizations enter a nearly arrested state with an effective response $R\sim10^{-4}$. For realization N, such arrested state persists till the end of the simulation (1 million time steps), whereas for realization F, around $t=7\times10^5$, the system regains motion, although still at a much lower speed ($R\sim10^{-3}$, three orders below the normal-fluid value). Visual inspection of selected snapshots of the density configuration provides a valuable insight into the physics of the problem. In top panel of Figs. 3 and 4, we report the density contours of both realizations at $t=3\times10^5$, i.e., in a early, nonflowing, stage of the evolution. As it is well visible, the density field shows a granular morphology, with droplets of dense fluids

surrounded by a sea of light fluid, itself entrapped by a thin belt of dense fluid (cages). The macroscopic motion of such foamlike configurations is highly frustrated by the presence of these dynamic cages, which have been identified as the prime cause for enhanced viscosity.

Such an interpretation is confirmed by visual inspection of the density field configurations at $t=1\times10^6$, when the configuration F has regained significant motion. For the case of realization N (bottom panel of Fig. 3), no qualitative change is visible, with cages still alive. For the case of realization F, however, bottom panel of Fig. 4 clearly shows that cages have mostly disappeared, thereby allowing the system to regain a flowing state, if only at a much reduced speed than a normal fluid.

The process of cage formation/annihilation is a highly complex phenomenon, whose statistical dynamics depends on the physical parameters, as well as on initial conditions and system size. A quantitative characterization of such phenomenon calls for intensive and systematic computational investigations, involving long-time simulations over a substantial ensemble of realizations, for a broad range of physical parameters. As shown in this work, GPU implementations prove instrumental in cutting down the computational costs of such investigations, thereby opening the way to systematic computational studies of the statistical dynamics of flowing soft-glass systems using the glass-LB scheme.

V. CONCLUSIONS

Summarizing, we have described the implementation on a GPU architecture of a Lattice Boltzmann model recently introduced for the study of soft-glass flowing systems. The GPU version is shown to provide major savings (more than an order of magnitude) in elapsed time over the corresponding CPU version, with a growing trend with increasing system size. This opens the way to systematic LB studies of the statistical dynamics of soft-glass flowing systems, typically from months to days elapsed time.

VI. SUMMARY AND OUTLOOK

A GPU implementation of the multicomponent lattice Boltzmann equation with multirange interactions for soft-glassy materials has been discussed. Performance measurements for soft flows under periodic shear indicate a GPU/CPU speed up ranging from 2 to 12 for grids from 128^2 to 1024^2, respectively. Such major speed up permits to handle multimillion time-steps simulations of 1024^2 grids within very few days of GPU time, thereby considerably expanding the scope of the glassy LB toward the investigation of long-time relaxation properties of soft-flowing glassy materials. Based on these results, the present GPU-LB implementation is expected to offer an appealing computational tool for future investigations of the *nonequilibrium* rheology of a broad class of flowing disordered materials, such as microemulsions, foams and slurries, on space and time scales of experimental interest.

Finally, we expect to develop, in the near future, a combined (MPI+GPU) version that will allow to exploit the huge computational capabilities of GPU clusters as we already did for another code described in [10].

ACKNOWLEDGMENTS

S.S. wishes to acknowledge financial support from the project INFLUS (Grant No. NMP3-CT-2006-031980) and SC financial support from the ERG EU grant and consorzio COMETA. Fruitful discussions with A. Cavagna, L. Biferale, and M. Cates are kindly acknowledged.

[1] W. B. Russel, D. A. Saville, and W. R. Schowalter, *Colloidal Dispersion* (Cambridge University Press, Cambridge, England, 1989); P. H. Poole, F. Sciortino, U. Essmann, and H. E. Stanley, Nature (London) **360**, 324 (1992); P. Sollich, F. Lequeux, P. Hébraud, and M. E. Cates, Phys. Rev. Lett. **78**, 2020 (1997); R. G. Larson, *The Structure and Rheology of Complex Fluids* (Oxford University Press, New York, 1999); T. Eckert and E. Bartsh, Phys. Rev. Lett. **89**, 125701 (2002); F. Sciortino, Nature Mater. **1**, 145 (2002); K. N. Pham *et al.*, Science **296**, 104 (2002); H. Guo *et al.*, Phys. Rev. E **75**, 041401 (2007); P. Schall *et al.*, Science **318**, 1895 (2007); P. J. Lu, E. Zaccarelli, F. Ciulla, A. B. Schofield, F. Sciortino, and D. A. Weitz, Nature (London) **453**, 499 (2008); M. P. Allen and D. J. Tildesley, *Computer Simulations of Liquids* (Oxford University Press, New York, 1990); D. Frankel and B. Smith, *Understanding Molecular Simulation* (Academic Press, San Diego, 1996); K. Binder, and D. W. Herrman, *Monte Carlo Simulation in Statistical Physics* (Springer, Berlin, 1997); W. Kob, in *Slow Relaxation and Nonequilibrium Dynamics in Condensed Matter*, edited by J.-L. Barrat, M. Feigelman, and J. Kurchan (Les Houches, Summer School Session LXXVII, 2003).

[2] R. Benzi, S. Chibbaro, and S. Succi, Phys. Rev. Lett. **102**, 026002 (2009).

[3] R. Benzi, S. Succi, and M. Vergassola, Phys. Rep. **222**, 145 (1992); S. Chen and G. D. Doolen, Annu. Rev. Fluid Mech. **30**, 329 (1998).

[4] S. Chibbaro, G. Falcucci, X. Shan, H. Chen, and S. Succi, Phys. Rev. E **77**, 036705 (2008).

[5] X. Shan and H. Chen, Phys. Rev. E **47**, 1815 (1993).

[6] R. Benzi, M. Sbragaglia, S. Succi, M. Bernaschi, and S. Chibbaro, J. Chem. Phys. **131**, 104903 (2009).

[7] NVIDIA CUDA Compute Unified Device Architecture Programming Guide, http://www.nvidia.com/cuda.

[8] W. Li, X. Wei, E. Arie, and A. E. Kaufman, Image Vis. Comput. **19**, 7 (2003).

[9] G. Wellein, T. Zeiser, G. Hager, and S. Donath, Comput. Fluids **35**, 910 (2006).

[10] M. Bernaschi, M. Fatica, S. Melchionna, S. Succi, and E. Kaxiras. A flexible high-performance Lattice Boltzmann GPU code for the simulations of fluid flows in complex geometries. Con-

currency: Practice and Experience (2009).
[11] M. Harris, Optimizing CUDA. Part of the High Performance Computing with CUDA tutorial held at SUPERCOMPUTING 2007 on Sunday, November 11, 2007, slides available at http://gpgpu.org/static/sc2007/SC07_CUDA_5_Optimization_Harris.pdf.
[12] J. Tölke, Implementation of a lattice Boltzmann kernel using the Compute Unified Device Architecture developed by NVIDIA. Computing and Visualization in Science (2008).
[13] J. Tölke and M. Krafczyk, Int. J. Comput. Fluid Dyn. **22**, 7 (2008).
[14] A. Kaufman, Z. Fan, K. Petkov, J. Stat. Mech.: Theory Exp. 2009, P06016.