MUPHY: A parallel MUlti PHYsics/scale code for high performance bio-fluidic simulations

MUPHY: A Parallel MUlti PHYsics/Scale Code for High Performance Bio-Fluidic Simulations

M. Bernaschi[†], S. Melchionna[°,*], S. Succi[†,◇], M. Fyta[*], E. Kaxiras[*], J. K. Sircar[*]

[†] *Istituto Applicazioni Calcolo, CNR,*
Viale del Policlinico, 137 - 00161 Rome, Italy
[°] *SOFT, Istituto Nazionale Fisica della Materia, CNR,*
P.le A. Moro, 2 - 00185 Rome, Italy
[*] *Department of Physics and School of Engineering and Applied Sciences,*
Harvard University,
Cambridge MA 02138, USA
[◇] *Initiative in Innovative Computing,*
Harvard University,
Cambridge MA 02138, USA

Abstract

We demonstrate the excellent parallel scalability and global performance on the IBM BlueGene Supercomputer of MUPHY, a new multiscale/physics code. MUPHY provides a unique parallel multiphysics combination of microscopic Molecular Dynamics (MD) with a hydro-kinetic Lattice Boltzmann (LB) method, which permits to simulate a variety of bio-fluidic problems across a wide range of scales in a fully concurrent fashion. The features of MUPHY are hereby demonstrated for the case of translocation of biopolymers through nanometer-sized, multi-pore configurations, taking into account explicitly the hydrodynamic interactions of the translocating molecules with the surrounding fluid. The parallel implementation exhibits excellent scalability on the BlueGene platform and includes techniques which may improve the flexibility and efficiency of other complex multi-physics parallel applications, such as hemodynamics, targeted-drug delivery and others.

Key words: Lattice Boltzmann, Molecular Dynamics, Multi Physics, Polymer Translocation, Parallel Processing.

1 Introduction

A wide variety of physical systems exhibit complex behavior which involves several spatial and temporal scales. Modeling this type of behavior requires the combination of several types of physical theory, that can be implemented in multi-

scale/multi-physics computational approaches. The need for such sophisticated computational models is driven both from an interest in fundamental understanding of complex systems behavior as well as from a practical perspective to improve technological and industrial applications. Indeed, many advanced-technology applications, from high efficiency turbines to transistors with nano-scale functional elements, rely heavily on multiscale/multiphysics approaches for their design [1]. Much progress has been made during the last decade in formulating multi-scale approaches based on composite computational schemes in which cross-scale information is exchanged in either a sequential or, less frequently, a concurrent manner [2].

A particularly interesting class of complex phenomena that involve multiple scales and physics are biological processes. Biological systems often exhibit a complexity and diversity that straddles across many decades in space-time resolution, from the quantum mechanical level of interactions responsible for bonding at the atomic scale to the continuum level of fluid motion at macroscopic scales. We focus here on a representative example of such a system, the translocation of DNA through a nano-pore with the ultimate goal of ultrafast electronic sequencing. In this example, as in many similar biological systems, the construction of an *efficient and accurate* multiscale/multiphysics computational method is particularly challenging due to the diverse and competing requirements of the individual scales involved. We report the construction of such a computational method (MUPHY) and demonstrate its efficient implementation on a highly parallel architecture, the IBM BlueGene Supercomputer.

The approach we follow is genuinely multiphysics, as it combines different levels of the statistical description of matter, continuum fluids and individual molecules. It is also genuinely multiscale, since fluid and molecular degrees of freedom are advanced concurrently, while invoking a sub-cycle time-stepping for these degrees of freedom to take into account the fact that atomistic dynamics is generally faster than the dynamics of the surrounding fluid (solvent). At variance with the vast majority of hybrid fluid/atomistic schemes, the present work is based on a *mesoscopic/hydrokinetic* representation of the solvent, via the so-called Lattice Boltzmann (LB) method [3,4,5].

On general grounds, kinetic theory, which lies in-between the continuum and atomistic description of matter, provides a natural candidate for multiscale fluid applications. In the case of Lattice Boltzmann, this general observation results into a series of specific computational assets, as briefly described in the following. LB is a minimal form of kinetic Boltzmann equation, based on the collective dynamics of fictitious particles, existing on the nodes of a regular lattice, and representing a local ensemble of solvent molecules (Boltzmann distribution function). The dynamics of such particles is designed in such a way as to reproduce hydrodynamic behaviour in the continuum limit, in which the molecular mean free path is much shorter than typical macroscopic scales.

Full details on our coupled LB/MD schemes are reported in [7]. It is worth noting that LB and MD with Langevin dynamics have been coupled before [6]. However, to the best of our knowledge, this is the first work in which such coupling is carried out using a flexible high performance parallel code able to support long molecules consisting of tens of thousands of beads of biological interest. In addition, the indirect addressing of the main lattice data structure allows our LB code to handle efficiently nearly arbitrary geometries at a minimum extra-memory cost.

The paper is organized as follows: Section 2 reviews the simulation method used to couple the Lattice Boltzmann description for the solvent to the Molecular Dynamics description of the solute. Section 3 describes the issues faced in the development of the parallel code and the solutions we adopted. Section 4 presents the numerical demonstration for the case of biopolymer translocation across a multi-hole membrane. Finally, a brief discussion of the future perspectives of this activity is offered.

2 Coupling between Lattice Boltzmann and Molecular Dynamics

In this section, we review the basic methodology, for which further details can be found in previous works [7,8].

In the LB method the basic quantity is $f_i(\vec{x},t)$, representing the probability of finding a "fluid particle" at the spatial mesh location $\vec{x}$ and at time t with discrete speed $\vec{c}_i$. Actually, "fluid particles" do not correspond to individual solvent molecules, but they represent instead the collective motion of a group of physical particles (populations). Once the discrete populations f_i are known, the local density, flow speed and momentum-flux tensor are obtained by a direct summation upon all discrete populations at a given lattice site. For the present study, we use the common three-dimensional 19-speed lattice [5]. The fluid populations are advanced in time through the following evolution equation:

$$\begin{gathered} f_i(\vec{x}+\vec{c}_i\Delta t, t+\Delta t) = \\ f_i(\vec{x},t) + \omega\Delta t(f_i - f_i^{eq})(\vec{x},t) + F_i\Delta t + S_i\Delta t \end{gathered} \tag{1}$$

where the discrete velocities $\vec{c}_i$ connect mesh points to first and second topological neighbors, with particle displacements $\Delta\vec{x}_i = \vec{c}_i\Delta t$. The right hand side of Eq. (1) represents the effect of fluid-fluid molecular collisions, through a relaxation towards a local equilibrium, f_i^{eq}, typically a second-order expansion in the fluid velocity of a local Maxwellian with speed $\vec{u}$:

$$f_i^{eq} = w_i[\beta\vec{u}\cdot\vec{c}_i + \frac{\beta^2}{2}(\vec{u}\vec{u}\cdot(\vec{c}_i\vec{c}_i - \frac{1}{\beta}\overleftrightarrow{I})] \tag{2}$$

where $\beta = 1/k_BT$ is the inverse temperature, w_i a set of weights normalized to unity, and $\overleftrightarrow{I}$ is the unit tensor in Cartesian space. The relaxation frequency ω con-

trols the kinematic viscosity of the fluid, $\nu = k_B T \left(\frac{1}{\omega} - \frac{\Delta t}{2}\right)$. The F_i term represents a stochastic force accounting for fluctuations in the fluid (fluctuating hydrodynamics). This term is local in space and time and acts at the level of the shear as well as non-hydrodynamic modes carried by the LB populations (see ref. [9] for full details).

The source term S_i accounts for the presence of atomic-scale particles embedded in the LB solvent and represents the momentum and momentum-flux input per unit time due to the influence of atomic-scale particles on the fluid population f_i.

Whenever the LB method is used to simulate a plain fluid, with no thermal fluctuations, the F_i and S_i terms disappear from the evolution equation.

Before describing the MD part, we emphasize that a LB solver is particularly well suited to the study of a number of interesting problems for several reasons: first, free-streaming of the fluid proceeds along straight trajectories which greatly facilitates the imposition of geometrically complex boundary conditions, such as those required to describe membranes and nano-pores. Second, fluid diffusivity emerges from the first-order LB relaxation-propagation dynamics, so that the kinetic scheme can march in time-steps which scale only linearly with the mesh resolution. Third, since both fluid-fluid and fluid-polymer collisions are completely local, the LB scheme is well suited to parallel computing. These features make the LB an appealing method as compared to other available alternatives, which typically scale superlinearly with the number of particles.

We now describe the MD section of the code, bearing in mind that the embedded solute has a molecular topology, such as DNA, where a linear collection of N_0 beads (each bead or solute particle representing a collection of atoms or molecules) compose the polymer. The solute particles are advanced in time according to the following MD equations for the positions $\vec{r}_p$ and velocities $\vec{v}_p$

$$\begin{aligned} \frac{d\vec{r}_p}{dt} &= \vec{v}_p \\ m\frac{d\vec{v}_p}{dt} &= \vec{F}_p^{\,c} + \vec{F}_p^{\,f} + \vec{F}_p^{\,r} + \vec{F}_p^{\,b}, \qquad p = 1, N_0 \end{aligned} \tag{3}$$

where the forces on the right-hand side are given by

$$\vec{F}_p^{\,c} = -\sum_q \partial_{\vec{r}_p} V(|\vec{r}_p - \vec{r}_q|)$$

and $\vec{F}_p^{\,f} = \gamma(\vec{u}_p - \vec{v}_p)$. For the bead-bead interaction potential $V(r)$ we chose the repulsive part of a standard $6-12$ Lennard-Jones potential truncated at its minimum. The term $\vec{F}_p^{\,f}$ represents the mechanical friction between a single particle and the surrounding fluid, $\vec{v}_p$ being the particle velocity and $\vec{u}_p \equiv \vec{u}(\vec{r}_p)$ the fluid velocity,

evaluated at the particle position, with γ the friction coefficient. In addition, the particles feel the effects of stochastic fluctuations of the fluid environment through the random term $\vec{F}_p^{\,r}$, obeying the fluctuation-dissipation relations. Finally, $\vec{F}_p^{\,b}$ corresponds to the bonding forces acting between particles with labels p and $p+1$ of the polymeric chain; that is, consecutive beads along the chain. The stiff bonding forces introduce fast oscillations which can make the numerical scheme unstable at large time-steps. Typically, such modes carry frequencies up to two orders of magnitude higher than those relative to non-bonding forces. To take into account such oscillations, a small integration time step should be used, with a resulting penalty of the computational efficiency. Actually, as will be described in the following, a multiple time step algorithm permits to achieve stability without compromising numerical efficiency. Clearly, in the LB approach all quantities have to reside on the lattice nodes, which means that the frictional and random forces need to be extrapolated from the particle to the lattice location. This is obtained by extracting the fluid velocity field $\vec{u}_p$ at the nearest lattice point from each particle position and, similarly, by assigning these forces as feed-back on the fluid populations through the same simple recipe.

The numerical solution of the stochastic equations is achieved through the Stochastic Position Verlet (SPV) scheme, as introduced in [10], a propagator which is second order accurate in time. For the translocating polymer, the MD solver is marched in time with a fraction of the LB time-step, $dt = \Delta t/M$ (a typical value of the time-step ratio M is 2). A multiple time step integrator is employed [10] by introducing a nested sub-cycle over a time-step $dt^b = dt/M^b$. The multiple time step solver is marched in time with time-step ratios $M = 2$ and $M^b = 5$, providing accurate results in terms of stability and unbiased statistical averages, as verified by monitoring the system average temperature, which remains equal to the preset value. To be noted that the LB/MD coupling in our approach is much tighter than for current Navier-Stokes/MD hybrid schemes, typically featuring an order-of magnitude larger separation, $M \sim 100$.

3 Code Parallelization

MUPHY (MUlti PHYsics/multiscale computer code) is written in Fortran 90. We chose MPI as the communication interface for the parallelization since it offers high portability among different platforms and allows good performances due to the availability of highly tuned implementations. The code has been tested on different platforms (including clusters of PC) and in particular on the IBM BlueGene/L system [11], whose main features can be summarized as follows:

- dual processors per node with two working modes: co-processor (1 user process/node: computation and communication work is shared by two processors) and virtual node (2 user processes/node);

- system-on-a-chip design with superscalar 700 MHz PowerPC 440 cores;
- a large number of nodes (scalable up to at least 65,536);
- three-dimensional torus interconnect with auxiliary networks for global communications, I/O, and management;
- lightweight, Unix-like, OS per node for minimum system overhead.

The parallelization of the Lattice Boltzmann method and Molecular Dynamics algorithms, as two distinct and *separate* problems, has been extensively studied for a number of years [12,13,14,15,16,17,18]. However, the coupling of these techniques raises new issues that need to be solved in order to achieve scalability and efficiency for large scale simulations. We addressed these issues starting from a serial version of the combined code, instead of trying to combine two existing parallel versions. The original LB code was primarily updated to take advantage of optimizations like *i)* removal of redundant operations; *ii)* buffering of multiply-used operations and *iii)* "fusion" of the collision and streaming steps in a single loop. This latter technique, already in use in other high-performance LB codes, significantly reduces data traffic between main memory and cache/registers of the processor, since there is only one read and one store of all LB populations at each time step.

With these optimizations in place, we are able to achieve $\sim 30\%$ of the peak performance of a single BlueGene core for the plain LB method (equation (1) with no F_i and S_i terms). This result is in line with other highly tuned LB kernels [19]. Indeed we wish to remind that: *i)* the algorithm for the update of the LB populations has an unfavorable ratio between number of floating point operations and number of memory accesses; *ii)* no optimized libraries are available like for other computational kernels (*e.g.,* matrix operations or FFTs); *iii)* it is not possible to exploit the SIMD-like operations of the PowerPC 440 processor since they require stride one access whereas the LB method has a "scattered" data access pattern. For the generation of the random numbers required by the simulation of the stochastic fluctuations (the F_i term in equation 1) we resorted to the very efficient library functions available in the IBM ESSL.

As to the parallelization, we followed an approach that entails a sort of "run-time" pre-processing. For the LB part of the code, this initial stage can be summarized as follows. Each node of the LB lattice is labeled with a *tag* that identifies it as belonging to a specific subregion of the computational domain (*i.e.*, fluid, wall, inlet, outlet, solid), as read from an input file. It is possible to define a *default* value for the tag so that nodes that are not explicitly tagged are assumed to have that value for the tag (this assumption reduces significantly the size of the input file). The *tag* file is read by a single task that distributes the input data to all the other tasks using collective communication primitives. Nodes are stored according to a linearized indirect addressing scheme [20][21]. Obviously this indirection requires, for each node, an additional data structure that contains the list of its neighboring (fluid, wall, inlet, outlet) nodes. At first glance a mechanism like this may appear a waste of time and memory, but, actually, for most (non-trivial) geometries it provides huge sav-

ings in memory requirements and simulation times [22]. In the beginning of the simulation, a subset of nodes is assigned to each computational task, attempting to balance the number of nodes *per* task as much as possible (obviously, in some cases, this operation cannot be done exactly). The assignment takes into account the domain decomposition strategy, that can be one-, two- or three-dimensional. All possible combinations (that is, Cartesian decompositions along $X, Y, Z, XY, XZ, etc.$) are supported. Moreover *custom* decompositions, *e.g.,* those produced by tools like METIS[23], which are necessary for irregular domains, are also supported. After the assignment of the nodes to the tasks, the *pre-processing* phase begins. Basically, each task checks which tasks own the nodes to be accessed during the subsequent phases of simulation, in particular for the streaming part of the LB algorithm. Such information is exchanged by using MPI collective communication primitives, so that each task knows the neighboring peers for send/receive operations. Information about the size of data to be sent/received is exchanged as well. In principle, each node could determine by itself all the information required for the communication phase, but the availability of highly tuned collective communication primitives makes more efficient to exchange part of these data among the tasks than computing everything locally.

In a parallel LB scheme there are several sections requiring data exchange: *i)* streaming; *ii)* handling of periodic boundary conditions; *iii)* presence of reflecting or absorbing *walls* within the computational domain. In our code, both boundary conditions and *walls* are managed implicitly during the streaming phase using indirect addressing. The communication scheme is based on the following approach: the *receive* operations are always posted in advance by using corresponding non-blocking MPI primitives, then the *send* operations are carried out using either blocking or non-blocking primitives, depending on the parallel platform in use (unfortunately, as it is well know, few platforms allow real overlapping between communication and computation). Then, each task waits for the completion of its receive operations, using the MPI *wait* primitives. The latter operation, in case of non-blocking *send* operations, is to wait for their completion. Also the choice between blocking and non-blocking *send* can be done at run time. The evaluation of global quantities (*e.g.,* the momentum along the X, Y, Z directions) is carried out by using MPI collective *reduction* primitives.

For the Molecular Dynamics section, a parallelization strategy suitable for the multi-scale method described in Section 2 had to be developed. In typical MD applications the spatial distribution of particles may be highly inhomogeneous. A straightforward approach to achieve a good load balancing is to resort to a domain decomposition such that each task has (approximately) the same number of particles. In this way, the size of the spatial sub-domains assigned to each task may vary significantly. In a stand-alone Molecular Dynamics simulation, this is acceptable but in our case the LB component would be hindered, since the computational load is proportional to the size of the spatial domain assigned to each task. One might opt for two separate domain decompositions for the LB and the MD part

of the simulation. However, the exchange of momentum among particles and the surrounding fluid would become a non-local operation, with a very high cost due to the long-range point-to-point communications imposed on the underlying hardware/software platform. For the IBM BlueGene such communications are explicitly discouraged.

In the end, we decided to resort to a domain decomposition strategy where the parallel sub-domains coincide with those of the LB scheme. In this way, each computational task performs both the LB and MD calculations and the interactions of the particles with the fluid are completely localized (there is no need to exchange data among tasks during this stage). Moreover, the task mapping becomes trivial since the computational domain of the LB is a regular box. To compensate the resulting unbalance of the MD computational load, we resort to a *local dynamic* load balancing algorithm, as outlined in the following.

At first, during the *pre-processing* step a subset of particles is assigned to each computational task according to the position in space of the particles. As the simulation proceeds, particles migrate from one domain to another and particle coordinates, momenta and identities are re-allocated among tasks. Non-bonding forces between intra- and inter-domain pairs of particles, involving the communication of particle positions between neighboring tasks, are computed. The molecular topology is also taken into account by exchanging details on the molecular connectivity, in order to compute bonding forces locally. In this respect, the way parallel MD is implemented is rather standard [18] and will not be described in detail. The main difference with respect to the standard procedure is that each task carries out an additional step, that is the exchange of momentum with the fluid. By design, this operation is carried out by each task without exchanging data with other tasks.

The dynamic load balancing strategy impacts the computation of the non-bonding forces, representing the most CPU demanding part of MD. At first, the strategy requires a communication operation between neighboring tasks that tracks the number of particles assigned to the neighbors. On the IBM BlueGene, it is more efficient to resort to a global collective communication primitive (*i.e.,* the *AllGather*) which involves all processors than to a sequence of 6 point-to-point communications with the nearest neighbor tasks. Our strategy, depicted in Fig. 1, works as follows. Whenever the number of interacting pairs owned by a given task exceeds the number of pairs assigned to a neighbor by a threshold, a fraction of the excess pairs is sent to that neighbor. This way, a *local* load balancing for the computation of non-bonding forces is achieved. A set of *precedence* rules prevents situations in which a task sends pairs to a neighbor and receives pairs from another. The receiving task computes the corresponding non-bonding forces and sends them back to the task that actually owns the particles. For the system under study, the communication/computation ratio is such that this strategy results in a sizeable advantage for the global efficiency.

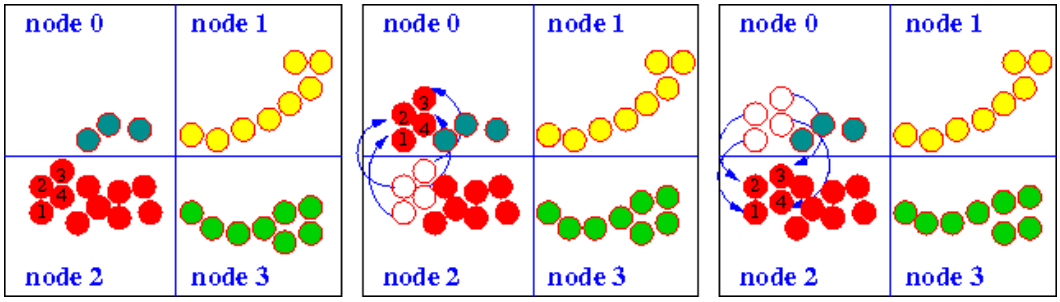

Figure 1. Local load balancing in a simplified 2D case. Prior to the MD phase (left panel) node 2 has 11 beads whereas node 0 has only three beads. Node 1 and node 3 have the same number of beads. To balance the load during the MD phase, node 2 "virtually" transfers the coordinates of four atoms to node 0 (central panel). After the calculation of the non-bonding forces, the results are returned to node 2 (right panel). The color of the beads is used to identify those belonging to the same subdomain.

4 Numerical Set-up

Motivated by recent experimental studies[24], we applied our multiscale approach to the simulation of the translocation of biopolymers through a series of narrow pores. This type of biophysical process is important in phenomena like viral infection by phages, inter-bacterial DNA transduction or gene therapy [25]. In addition, it is argued that this type of process may open a way to ultrafast DNA-sequencing by sensing the base-sensitive electronic signal as the biopolymer passes through a nanopore with attached electrodes.

The translocation is a complex phenomenon involving the competition between many-body interactions at the atomic or molecular scale, fluid-atom hydrodynamic coupling, as well as the interaction of the biopolymer with wall molecules in the region of the pores.

In our simulations, we use a three-dimensional box of size $N_x \times N_y \times N_z$ in units of the lattice spacing Δx. Periodicity is imposed for both the fluid and the polymer in all directions. A separating wall is located in the mid-section of the x direction, at $x/\Delta x = N_x/2$, with a set of 64 holes of side $h = 3\Delta x$ (distributed in a regular way along 8 rows and 8 columns) through which 64 polymers (each composed by 4000 beads) translocate from one chamber to the other. The total particle and mesh size is 256000 beads and $N_x = N_y = N_z = 512$. At $t = 0$ the polymer resides entirely in the right chamber at $x/\Delta x > N_x/2$. Two snapshots of the polymers configuration in the beginning and in the mid of the simulation are reported in Figure 2. From this picture, the load unbalancing of the MD part is apparent. Translocation is induced by a constant electric force (F_{drive}) which acts along the x direction and is confined in a rectangular channel of size $3\Delta x \times \Delta x \times \Delta x$ along the streamwise (x direction) and cross-flow (y, z directions). The solvent density and kinematic viscosity are 1 and 0.1, respectively, and the temperature is $k_B T = 10^{-4}$. All parameters are in units of the LB time-step Δt, lattice spacing Δx and

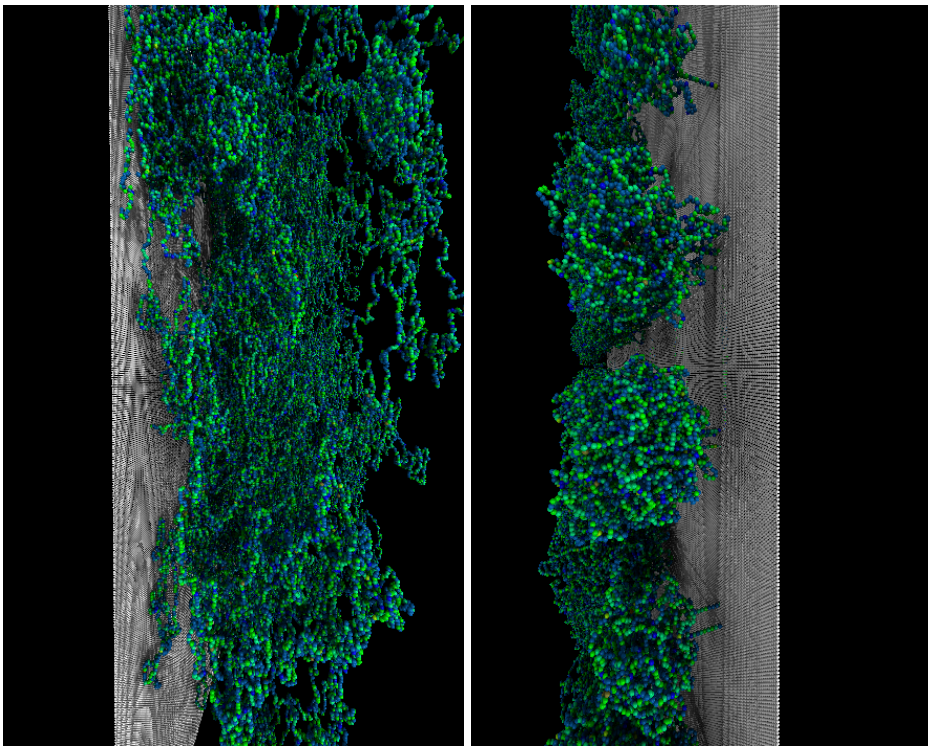

Figure 2. The top panel shows the configuration right after the beginning of the translocation. The bottom panel shows the configuration near the end of the translocation. The beads are colored according to the value of the hydrodynamic work *per* unit time they absorb from the fluid.

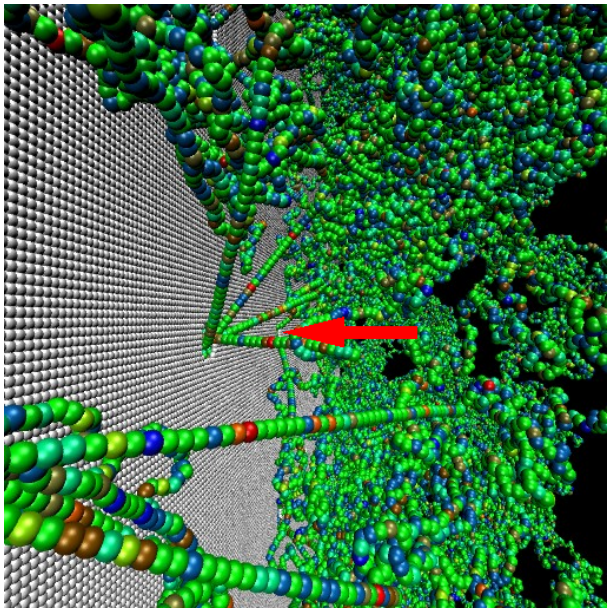

Figure 3. A detail of the wall with one of the nanopores and a polymer entering the pore driven by the electric field (indicated by the red arrow).

solvent mass Δm, which we set equal to 1. The friction coefficient γ governs both the structural relation of the polymer towards equilibrium and the strength of the coupling with the surrounding fluid. With this parameterization, the process falls within the fast translocation regime, where the total translocation time is much smaller than the Zimm relaxation time. Additional details have been presented in Ref. [26].

In a typical run the Lattice Boltzmann part of the code takes approximately 65% of the computing time. The exchange of momentum between MD and LB takes 3% of the time. The Molecular Dynamics takes approximately 30% of the time. Diagnostics and periodic output take the rest.

We ran our tests for the simulation of 64 polymers traversing 64 holes (for a total of 256000 particles) in a $512 \times 512 \times 512$ box on an IBM BlueGene system with up to 16384 nodes, corresponding to 32768 Virtual Processors. Visual inspection (see Figure 2) reveals that in the initial stage of translocation, the polymers are crowding one chamber in a rather dense and uniform way, and their distribution decays to zero at the far away edge of the initial translocation chamber. The second translocation chamber is initially empty and gradually fills up as translocation proceeds. As shown in Table 1, the time per step of a simulation of this size is < 0.04 sec on 32768 processors in *Virtual Node (VN)* mode, with a reasonable scaling for such large number of processors taking into account the special features of the problem under study. By analyzing the different sections of the code, the LB part appears to scale linearly (at times even superlinearly, probably due to a positive side effect of the reduction of lattice nodes *per* task on cache usage). To be noted that also the MD part exhibits a significant speed-up, showing that the saturation regime is still far from being hit. Without the dynamic local load balancing, the time required by the MD part of the simulation increases, on average, by $\sim 30\%$. It is also interesting to note that with a $512 \times 512 \times 512$ box and 32768 tasks, each task owns 4096 lattice nodes. The fact that the scalability remains good, especially for the LB part of the simulation, confirms the efficiency of the IBM BlueGene communication network. All floating point computations are performed in double precision.

For a correct interpretation of the timings reported in Table 1 it is worth noting that in the beginning of the simulation (approximately) half of the processors sit completely idle during the molecular dynamics part of the run. As the simulation proceeds, processors in charge of the region of the lattice where the polymers translocate, receive particles so those processors are no longer idle during that phase. At no stage of the simulation, uniform load balancing is achieved because the translocating molecules never fill up completely the computational domain.

Most results were produced using the nodes in "CoProcessor" (CO) mode. It is worth noticing that switching from CO to VN, at the same number of computational nodes (i.e. hardware resources) impacts both the LB and the MD section. As a matter of fact, in "Virtual Node" (VN) mode, the performance does not double

since all resources (cache, memory bandwidth, *etc.*) are shared between the two virtual CPUs of the node.

Number of tasks	Total Time	Total Efficiency	LB Time	MD Time
1024 (CO)	883.0	N/A	581.4	260.0
2048 (CO)	452.3	98%	290.0	135.2
4096 (CO)	233.3	95%	144.3	70.5
8192 (CO)	118.6	93%	72.1	38.1
16384 (CO)	59.2	93%	36.1	21.1
16384 (VN)	66.2	83%	40.1	23.0
32768 (VN)	36.1	76%	20.1	13.0

Table 1
Times (in seconds) for 1000 iterations of a 256000 particles multi-biopolymer translocation in a $512 \times 512 \times 512$ lattice. The simulation requires at least 1024 tasks due to its large memory requirements.

To measure the number of floating points operations *per* second, we resorted to the IBM HPCT library, which supports multiple instrumentation sections and nested instrumentation. The LB part of the simulation carries out slightly more than 210 Mflops/sec *per* task. Let us remind that this part includes the simulation of the hydrodynamic fluctuations (as a consequence its performance can not be measured in the *standard* Lattice-Updates-per-second unit) and the load balancing is almost perfect across all tasks (the tasks whose computational domain includes the separating wall show a limited deviation). As for the MD part, *on average*, each task performs slightly below 160 Mflops/sec, with about 50% of the tasks having zero load. Taken together, *on average*, each task performs at $\simeq 190$ Mflops/sec. On the largest configuration at our disposal (32768 computational tasks) these figures lead to an estimate of a total of 6.2 Teraflops/sec aggregate performance.
In addition, we ran some tests on 16 nodes of an IBM/SP system. The nodes are connected by a proprietary IBM switch and each node features 8 Power5 processors at 1.9 Ghz with 32 GB of RAM per node for a total of 128 processors and 512 GB of RAM. The execution time for 1000 iterations has been, with this system, 2005 seconds, which leads to estimate that almost 10000 Power5 processors are required to obtain the same performance of the BlueGene system (assuming a similar efficiency).

Table 2 shows the distribution of communication overheads for a bio-polymer translocation using 1024 tasks. For one thousand iterations, the total execution time is about 880 seconds with a communication overhead of slightly more than 110 seconds, in line with other molecular dynamics codes. The resulting *sustained* performance of point-to-point communications is ≥ 32 Mbytes/sec. A similar profile when the `MPI_Isend` primitive is used confirms that, at least on the BlueGene

MPI Routine	n. calls	avg. bytes	time(sec)
MPI_Send	609080	2260	41.0
MPI_Irecv	605185	393100	0.4
MPI_Waitall	26005	0.0	32.1
MPI_Allgather	2004	8	34.5
MPI_Reduce	712	28	4.8
MPI_Allreduce	14112	128	1.4

Table 2
Communication overhead on one out of 1024 processors for 1000 time-steps of the translocation of 64×4000 particles in a $512 \times 512 \times 512$ lattice.

platform, there is no real advantage in asynchronous send operations since time apparently saved in the *send* operation is actually spent in the corresponding *wait* operation.

5 Flexibility and Future Perspectives

To the best of our knowledge, MUPHY is one of the first examples of integrated high performance parallel code for the simulation of multi-physics/scale bio-fluidic phenomena. To place our result in perspective, let us consider that one bead corresponds to about 150 base pairs, that is one persistence length of double-stranded DNA (50 nm), whereas a single timestep covers about 170 ps [7]. With our sustained performances, a one-day simulation on the largest BG/L configuration (212992 processors) would cover a physical time span of about 3 ms for a domain of 20 μm in linear size. These space-time scales are clearly beyond reach of fully atomistic simulations and can only be attained by multi-scale/multi-physics approach.

For a number of years, sharp-shooted, highly-tuned, application codes have been used to run large scale simulations in many different fields of computational physics. However the attempts of coupling such application codes to simulate more complex, interdisciplinary, phenomena, have been often based on a simple "serial-pipe" paradigm (*i.e.,* the output of the microscopic code provides the input of the macroscopic code) with very limited (if any) run-time concurrency and system integration.

Although in the present paper we focused on just one specific application of MUPHY (the translocation of biopolymers), the code can be used to study a variety of other systems. For instance, thanks to indirect addressing, MUPHY seamlessly handles real-life geometrical set-ups, such as blood flows in human arteries in presence of white cells (see Figure 4 for an example), lipids, drug-delivery carriers and other suspended bodies of biological interest.

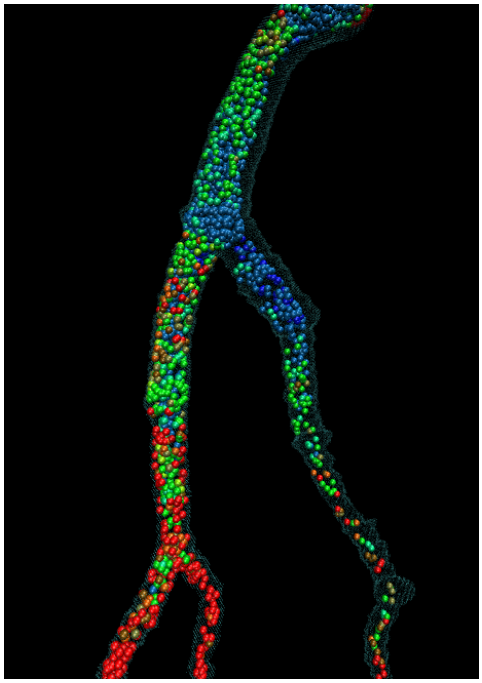

Figure 4. Blood flow pattern in a human coronary artery. Spheres represent transported white cells, whose size has been magnified as compared to realistic values, to highlight their distribution inside the arterial region. The color of the spheres represent the Shear Stress (SS) experienced locally during blood motion (blue: low SS, green: intermediate SS, red: high SS).

Acknowledgments

MB, SS and SM wish to thank the Initiative for Innovative Computing of Harvard University for financial support and the Harvard Physics Department and School of Engineering and Applied Sciences for their hospitality. MF acknowledges support by Harvard's Nanoscale Science and Engineering Center, funded by NSF (Award No. PHY-0117795).
We thank Giorgio Amati for useful discussions about the LB method. We thank IBM Research for granting access to their BlueGene installation in Yorktown Heights and CASPUR for access to their IBM/SP system.

References

[1] Lethbridge, P., Multiphysics Analysis, The Industrial Physicist, Dec 2004/Jan 2005

[2] Lu, G. and Kaxiras, E.: Overview of Multiscale Simulations of Materials, Handbook of Theoretical and Computational Nanothechnology , Vol. X:1-33, edited by M. Rieth and W. Schommers, American Scientific Publishers, 2005.

[3] Benzi, R. Succi, S., and Vergassola, M., Phys. Rep. 222:145-197, 1992.

[4] D.A. Wolf–Gladrow, Lattice gas cellular automata and lattice Boltzmann models, Springer Verlag, New York, 2000.

[5] Succi, S., The lattice Boltzmann equation. Oxford University Press, Oxford, 2001.

[6] Ahlrichs, P. and Duenweg, B.: Lattice-Boltzmann simulation of polymer-solvent systems. Int. J. Mod. Phys. C 9:1429-1438, 1999; Simulation of a single polymer chain in solution by combining lattice Boltzmann and molecular dynamics. J. Chem. Phys. 111:8225-8239, 1999.

[7] Fyta, M.G., Melchionna, S., Kaxiras, E., and Succi, S.: Multiscale coupling of molecular dynamics and hydrodynamics: application to DNA translocation through a nanopore. Multiscale Model. Simul. 5:1156-1173, 2006.

[8] Fyta, M., Sircar, J., Kaxiras, E., Melchionna, S., Bernaschi, M., and Succi, S.: Parallel multiscale modeling of biopolymer dynamics with hydrodynamic correlations. Intl. J. Multiscale Comput. Eng. 6:25, 2008.

[9] Adhikari, R., Stratford, K., Cates, M.E. and Wagner, A.J.: Fluctuating Lattice-Boltzmann, Europhys.Lett. 71:473-477, 2005.

[10] Melchionna, S.: Design of quasi-symplectic propagators for Langevin dynamics. J. Chem. Phys., in press, 2007.

[11] `http://www-03.ibm.com/systems/deepcomputing/bluegene`

[12] Amati, G. Piva, R. and Succi, S.: Massively parallel Lattice-Boltzmann simulation of turbulent channel flow. Intl. J. Mod. Phys. C 4:869-877, 1997.

[13] Boyer, L.L. Pawley, G.S.: Molecular dynamics of clusters of particles interacting with pairwise forces using a massively parallel computer. J. Comp. Phys. 78:405-423, 1988.

[14] Heller, H. Grubmuller, H. Schulten, K.: Molecular dynamics simulation on a parallel computer. Molec. Sim. 5:133-165, 1990.

[15] Rapaport, D.: Multi-million particle molecular dynamics: II.Design considerations for distributed processing. Comp.Phys.Comm. 62:198-216, 1991.

[16] Brown, D. Clarke, J.H.R. Okuda, M. Yamazaki, T.: A domain decomposition parallelization strategy for molecular dynamics simulations on distributed memory machines. Comp.Phys.Comm. 74:67-80, 1993.

[17] Esselink, K. Smit, B. Hilbers, P.A.J.: Efficient parallel implementation of molecular dynamics on a toroidal network: I.Parallelizing strategy. J. Comp. Phys. 106:101-107, 1993.

[18] Plimpton, S.: Fast parallel algorithms for short-range molecular dynamics. J. Comp. Phys. 117:1-19, 1995.

[19] Wellein, G. Zeiser, T. Donath, S. Hager, G.: On the single processor performance of simple lattice Boltzmann kernels Computers & Fluids 35, 2006.

[20] Dupuis, A., Chopard, B.: Lattice gas: an efficient and reusable parallel library based on a graph partitioning technique, in P. Sloot, M. Bubak, A. Hoekstra and B. Hertzberger, editors, HPCN Europe 1999, Springer, 1999.

[21] Schulz, M., Krafczyk, M., Toelke, J., Rank, E: Parallelization strategies and efficiency of CFD computations in complex geometries using the Lattice Boltzmann methods on high-performance computers. In M. Breuer, F. Durst, and C. Zenger, eds. High Performance Scientific and Engineering Computing, Proceedings of the 3rd International Fortwihr Conference on HPESC, Erlangen, March 12-14, 2001, vol 21 of Lecture Notes in Computational Science and Engineering, Springer, 2002.

[22] Axner, L., Bernsdorf, J.M., Zeiser, T., Lammers, P., Linxweiler, J., and Hoekstra, A.G.: Performance evaluation of a parallel sparse lattice Boltzmann solver, J. Comp. Phys., 227:10, 2008.

[23] `http://glaros.dtc.umn.edu/gkhome/views/metis/`

[24] Kim, M.J., Wanunu, M., Bell, D.C., Meller, A.: Rapid fabrication of uniformly sized nanopores and nanopore arrays for parallel DNA analysis. Adv. Mater. 3149-3153, 18, 2006.

[25] Lodish, H., Baltimore, D., Berk, A., Zipursky, S., Matsudaira, P., and Darnell, J.: Molecular Cell Biology, W.H. Freeman and Company, New York, 1996.

[26] Fyta, M., Kaxiras, E., Melchionna, S., Bernaschi, M., and Succi, S.: Quantized current blockade and hydrodynamic correlations in biopolymer translocation through nanopores: evidence from multiscale simulations Nano Letters, vol. 8:4, 2008.