Model-to-model interface for multiscale materials modeling

by

Perry Edward Antonelli

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Materials Science and Engineering

Iowa State University

Ames, Iowa

2016

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

This work was funded by US Department of Energy's Office of Fossil Energy at the Ames Laboratory under Contract No. DE-AC02-07CH11358. The Ames Laboratory is operated for the DOE by Iowa State University.

ABSTRACT

A low-level model-to-model interface is presented that will enable independent models to be linked into an integrated system of models. The interface is based on a standard set of functions that contain appropriate export and import schemas that enable models to be linked with no changes to the models themselves. These ideas are presented in the context of a specific multiscale material problem that couples atomistic-based molecular dynamics calculations to continuum calculations of fluid flow. These simulations will be used to examine the influence of interactions of the fluid with an adjacent solid on the fluid flow. The interface will also be examined by adding it to an already existing modeling code, Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) and comparing it with our own molecular dynamics code.

CHAPTER 1. INTRODUCTION

1.1 Background

Modeling is an important part of materials research. It provides a way to test ideas and predict materials properties without needing to make physical samples and perform experiments. Experiments are still an important part of the process, in fact, modeling works best when used in conjunction with experiments. Modeling can be used to determine which materials or processing conditions are mostly like to produce desirable properties. Additionally, experiments can be used to validate models; if simulations and experiments disagree, something is likely wrong with one of them.

Materials modeling often requires multiscale modeling, the use of two or more models at different length or time scales. This is necessary because materials are multiscale in nature; structures and processes at different length and time scales are often strongly linked together and affect the overall properties of a material. These different scales often require different models, which can be difficult to combine together.

In the current state of multiscale modeling, modeling codes are generally written to be specific to an application. The physics and boundary conditions are often hard coded to simulate only one system. Additionally the code for one model is hardwired to code for other models. The data structures and methods of passing information between models are strongly interconnected. A change in one model would require rewriting code relevant to all models. The code is inflexible and thus difficult to reuse in different applications.

Models at each scale are developed by experts who are often independent of an overall modeling effort. For example, someone who works on molecular dynamics models is rarely an expert on other methods, and may not know best how to couple their models to another. This is a barrier to linking models together. There does not currently exist a way to combine them into a multiscale modeling scheme. This way of doing multiscale modeling makes it difficult to change codes or to add additional physics.

We attempt to solve these problems through the use of a model-to-model interface that enables communication between independent models. The inspiration for and development of this interface are discussed in more detail in Chapter 2. Eventually, this interface will allow the creation of a library of independent materials models that can easily linked together for multiscale materials modeling. In this thesis we show how the interface works with a few different models: two of our own and the molecular dynamics code developed at Sandia National Laboratories, Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS).

1.2 Thesis Organization

Chapter 1 of this thesis in an introduction. Chapter 2 is the paper that has been published in Computation Materials Science, which shows how the interface works with two of our own models in the context of a simple fluid flow problem. In Chapter 3 is information about LAMMPS, details about the addition of our interface to LAMMPS, and a comparison of it with the results presented in Chapter 2.

CHAPTER 2. A MODEL-TO-MODEL INTERFACE FOR CONCURRENT MULTISCALE SIMULATIONS

A paper published in *Computational Materials Science*

P.E. Antonelli, K. M. Bryden, and R. LeSar

2.1 Introduction

Multiscale modeling is ubiquitous in science and engineering, arising in many and quite divergent domains, from the design of materials to simulations of power plants. All of these systems are governed by important features that control their behavior at multiple scales of length and time. Describing such systems often depends on having independent models based on differing physics for each scale [1][2].

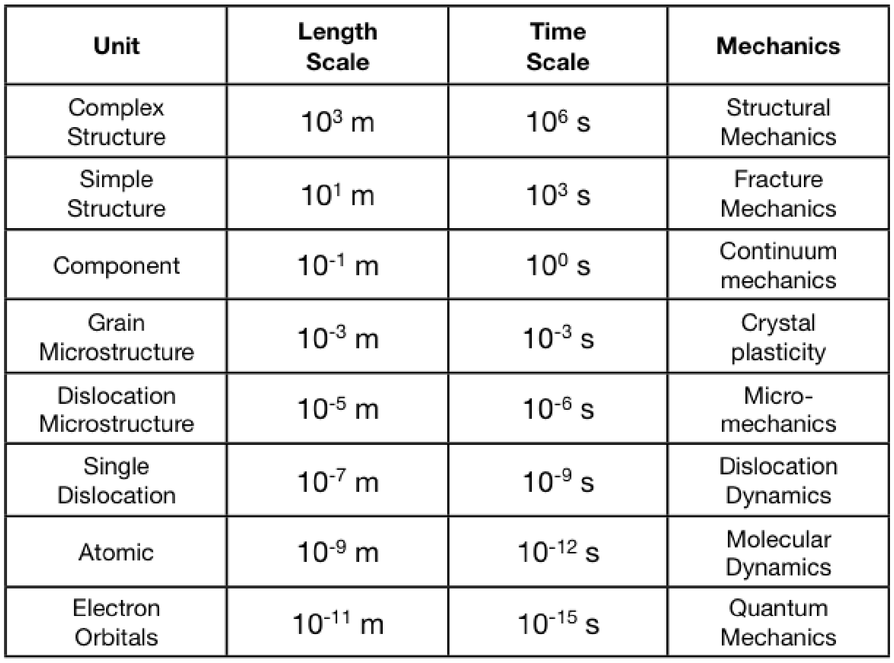
Integrated modeling [3][4], which can be defined as modeling that "includes a set of interdependent science-based components (models, data, and assessment methods) that together form the basis for constructing an appropriate modeling system" [3], provides a structure to develop and organize the relevant information about a multiscale system and to apply that information to model the behavior of that system in response to external forces. The challenges of creating such integrated modeling systems are common to many disciplines. The environmental modeling community, for example, has been very active in this area, driven by their need to coordinate between various disciplines and to integrate a wide range of models that are often developed autonomously and independently from the integrated modeling efforts, and generally by experts who may not be tightly connected to the overall effort[3][4]. Inclusion of new models and codes into an integrated modeling system thus often requires coordination between disparate groups, which can inhibit addition of new capabilities into the integrated scheme. As discussed in more detail below, we have similar challenges in multiscale materials modeling.

In this paper we present an approach to integrated modeling based on the development of a model-to-model interface that enables independent models to be more readily linked into an integrated system with no changes to the models themselves. We examined a number of existing technologies for linking models, all from environmental modeling. While these systems have not been applied to systems with the range of scales found in materials, their basic structures provide an excellent starting point for creating integrated multiscale materials models. The models we examined include the Bespoke Framework Generator [5], Earth System Modeling Framework [6], OASIS (Ocean Atmosphere Sea Ice Soil) [7], and CSDMS (Community Surface Dynamics Modeling System) [8]. All of these systems require models to have initialize, run, finalize, get, and set functions (examples are described below) for basic control over models. Also generally required by these couplers is a description of how the models are connected together, which is done in the form of an XML file or some other type of configuration file. OASIS and CSDMS have an additional advantage of having GUI's that allow placing the models and their connections into the interface for visualization and automated configuration file generation. CSDMS also uses Babel [9] to allow model writing in different languages (C++, C, Fortran, XML, Python, and Java are supported) whereas the other couplers are limited to models all being in the same language.

After consideration of the attributes of the various approaches, we have chosen to adapt the basic methodology of the CSDMS approach, which is based on the use of a low-level model-to-model interface, for linking materials models together. The purpose of this paper is to give an example of how to construct such an interface within a multiscale problem, with a focus on the types of information that must be passed from one model to another.

The biggest challenge for computational materials science is the extreme range of length and time scales that govern materials behavior, ranging from the and sub-picoseconds of atomistic behavior to the meters and years of materials in engineered applications. In between those extremes lie complex sets of physical phenomena that depend on the overall material behavior of interest and the type of material. We show a simple example of this range of mechanisms in Figure 2.1, highlighting the various scales that govern the mechanical behavior of metallic materials [10]. Consideration of other types of materials or other properties would lead to a similar, but not identical, figure. Looking more closely at Figure 2.1, we see that at each scale there is a structural“unit” that dominates the physical processes at the given length and timescales. These units are the entities whose dynamics define the physics of interest at each scale. Typically, a set of models of materials behavior is created for each of the various scales [2][11]. These models are generally developed and used by different groups. In the traditional view of computational materials science, this hierarchy of models and simulations, each describing a specific scale and its associated phenomena, are linked to create a multiscale description of materials behavior. Typically, information is passed sequentially from scale to scale, an approach that is often referred to as information passing or sequential multiscale [1]. Inherent in the information-passing paradigm is the lack of inverse models that would allow us to predict the needed structures and properties at the small scale from desired properties at a larger scale – information flows only one way. It is common that information passing is done be separate groups of researchers, using codes specific to the scales of their particular interests.

While the information-passing approach has proved useful for many systems and properties, it is, at its core, suspect in a number of ways. Information passing assumes that there is a separation of scales, which is not always clear. It is inherently a coarse-graining procedure, which loses information at each step, especially about the distribution of properties other than their mean. Indeed, this latter statement reflects the fundamental flaw of this approach – it assumes that we can start from fundamental laws and build up to a description of a macroscopic system, which is not obviously true [12]. Another approach, which is often referred to as “concurrent multiscale” [1], links simulations at differing scales directly within a single integrated framework. This approach has the advantage of enabling the flow of information between scales, which eliminates many of the uncertainties inherent in the information-passing paradigm. For recent examples, see [13][14][15]. As in sequential multiscale simulations, models must be developed and applied at multiple scales, with the added complication of developing appropriate interfaces between the models that enable them to be used concurrently. As we will discuss in detail below, the integrated approach described here provides for an efficient construction of



Unit	Length Scale	Time Scale	Mechanics
Complex Structure	10^{3} m	10^{6} s	Structural Mechanics
Simple Structure	10^{1} m	10^{3} s	Fracture Mechanics
Component	10^{-1} m	10^{0} s	Continuum mechanics
Grain Microstructure	10^{-3} m	10^{-3} s	Crystal plasticity
Dislocation Microstructure	10^{-5} m	10^{-6} s	Micro-mechanics
Single Dislocation	10^{-7} m	10^{-9} s	Dislocation Dynamics
Atomic	10^{-9} m	10^{-12} s	Molecular Dynamics
Electron Orbitals	10^{-11} m	10^{-15} s	Quantum Mechanics

Figure 2.1: Length and time scales in materials science adapted from [10]. On the left, we indicate the important unit structure at each scale, in the middle, the approximate length and time scales, and at the right, the approach used to simulate the material's mechanical behavior. We note that the approaches used at each scale are generally distinct to that scale and, most often, are developed and used by different research groups.

integrated multiscale materials models. We will present these ideas in the context of a specific problem, involving the interaction between an atomistic scale and a continuum scale, to serve as an example of the types of problems inherent in multiscale modeling of materials. Specifically, we use molecular dynamics to examine the interactions between a fluid and an underlying solid to set the boundary condition between a continuum fluid (described using the Lattice Boltzmann method) and the solid. Our focus is not on fluid properties, per se, but rather on how one connects an atomistic scale to a continuum scale in the context of creating a model-to-model interface to link together independent multiscale models. We describe the basic methodologies in Section 2.2, followed by a description of our implementation of a basic model-to-model interface in Section 2.3, in which we discuss modifications to the CSDMS approach necessary for our test problem. Section 2.4 is a summary of results from an application of the model, with comments about the efficiency of the code with the model interface. Concluding remarks are given in Section 2.5.

2.2 Coupled Multiscale Model of Surface-Fluid Interactions in Couette Flow

The goal of this work is to provide a method for coupling multiscale simulations using a low-level model-to-model interface. We focus on an example problem in which we examine the role of surface interactions on Couette flow, which is the laminar flow of a viscous fluid held between two plates, with one plate moving at some velocity relative to the other, fixed, plate. In traditional computational solutions to Couette flow, the velocity of the fluid at the stationary plate (which we will refer to as the slip velocity) is assumed to be zero and a linear increase of velocity is seen between the two plates. In this sample problem, our goal is to examine the role of fluid-surface interactions on determining the slip velocity by incorporating realistic descriptions of the interactions between atoms in the fluid and those on the surface of the solid. We model the system using a lattice Boltzmann method for the continuum fluid flow coupled to a molecular dynamics simulation of the interaction between the atoms of the fluid and the underlying surface. The models operate at different scales and are based on different physics so provide a good case for understanding information mediation in multiscale simulations. We note that this problem has been used previously as an example of coupled multiscale simulations, though without the model integration strategy introduced here.[e.g., [16]]

2.2.1 Lattice Boltzmann (LB) method

The lattice Boltzmann (LB) method [17] was introduced in the 1990s to address deficiencies in the lattice-gas cellular automaton model for fluid flow [18]. In the LB method, the single particles that move around the lattice in the lattice-gas method are replaced with single particle distribution functions, $f(\mathbf{x}, \mathbf{e_i}, t)$, in which f is the probability of a fluid particle at position $\mathbf{x}$ at time t having velocity $\mathbf{e}$. The probability distributions are discretized on a lattice (with positions $\mathbf{x}$) and represented as $f_i(\mathbf{x}, t) = f(\mathbf{x}, \mathbf{e_i}, t)$, in which i indicates a direction to the nearest points in the lattice (as described below), $\mathbf{e}\mathit{\Delta} t$ is the displacement vector along direction i, and $\mathit{\Delta} t$ is the time step.

The macroscopic quantities of density ρ and velocity $\mathbf{u}$ are found by evaluating the moments of f at each lattice site. These are given by

$$\begin{aligned}\rho(\mathbf{x}, \mathbf{e_i}) &= \rho_0 \sum_{i=0}^{18} f_i(\mathbf{x}, \mathbf{e_i}) \\ \rho(\mathbf{x}, \mathbf{e_i})\mathbf{u}(\mathbf{x}, \mathbf{e_i}) &= \rho_0 \sum_{i=0}^{18} f_i(\mathbf{x}, \mathbf{e_i})\mathbf{e}_i\end{aligned} \tag{2.1}$$

in which the sum is over the directions in the lattice and ρ_0 is the density of the system (an input parameter). The equation of motion for the probability distributions can be shown to be [17]

$$f_i(x + \mathbf{e}_i \Delta t, t + \Delta t) = f_i(x, t) + \frac{1}{\tau}(f_i^{eq} - f_i(x, t)) \tag{2.2}$$

where f_i^{eq} is a local equilibrium distribution resulting from collisions, $\mathit{\Delta} t$ is the time step and τ is a relaxation time. The equilibrium distributions are given by

$$f_i^{eq} = w_i \frac{\rho}{\rho_0}(1 + 3\mathbf{e}_i \cdot \mathbf{u} + \frac{9}{4}(\mathbf{e}_i \cdot \mathbf{u})^2 - \frac{3}{2}(\mathbf{u} \cdot \mathbf{u})) \tag{2.3}$$

in which w_i are a set of directional weights normalized to unity and c_s is the speed of sound. The w_i are effectively quadrature weights in the numerical integration over a continuous distribution [19].

In Figure 2.2 we show the grid used in this work. This lattice is generally referred to as the D3Q19 lattice (3 dimensions and 19 directions) [20]. Shown in the figure are the lattice directions and velocity vectors, with the velocity vectors given by

$$\mathbf{e}_i = \begin{cases} (0,0,0), & \text{i} = 0 \\ (\pm c,0,0),(0,\pm c,0),(0,0,\pm c), & \text{i} = \text{1-6} \\ (\pm c,\pm,0),(0,\pm c,\pm c),(\pm c,0,\pm c), & \text{i} = \text{7-18} \end{cases} \tag{2.4}$$

in which $c = \mathit{\Delta} x/\mathit{\Delta} t$ and $\mathit{\Delta} x$ is the grid size [21]. The speed of sound is given by $c_s = c/\sqrt{3}$ for the D3Q19 lattice [20]. The weights are

$$w_i = \begin{cases} 1/3, & \text{i} = 0 \\ 1/18 & \text{i} = \text{1-6} \\ 1/36, & \text{i} = \text{7-18} \end{cases} \tag{2.5}$$

such that $\sum_{i=0}^{18} w_i = 1$. To account for constraints on the particle densities introduced by using content-velocity boundary conditions at the top and bottom of the simulation, an artificial force, $\mathbf{Q}$, is applied to the unknown components of the particle density function [22].

The solution to the LB equations has two steps, advection and collision. The advection equation

$$f_i^{in}(x + \mathbf{e}_i\Delta t, t + \Delta t) = f_i^{out}(x,t) \tag{2.6}$$

and the collision equation is

$$f_i^{out}(x,t) = f_i^{in}(x,t) + \frac{1}{\tau}(f_i^{eq} - f_i^{in}(x,t)) \tag{2.7}$$

in which f_i^{out} denotes the outgoing (after collision) distribution functions while f_i^{in} denotes the incoming (before collision) distribution functions. Note that the LB calculations are all done in reduced units with unit densities, unit time steps, and unit distances. The calculations are scaled to specific units as specified below.

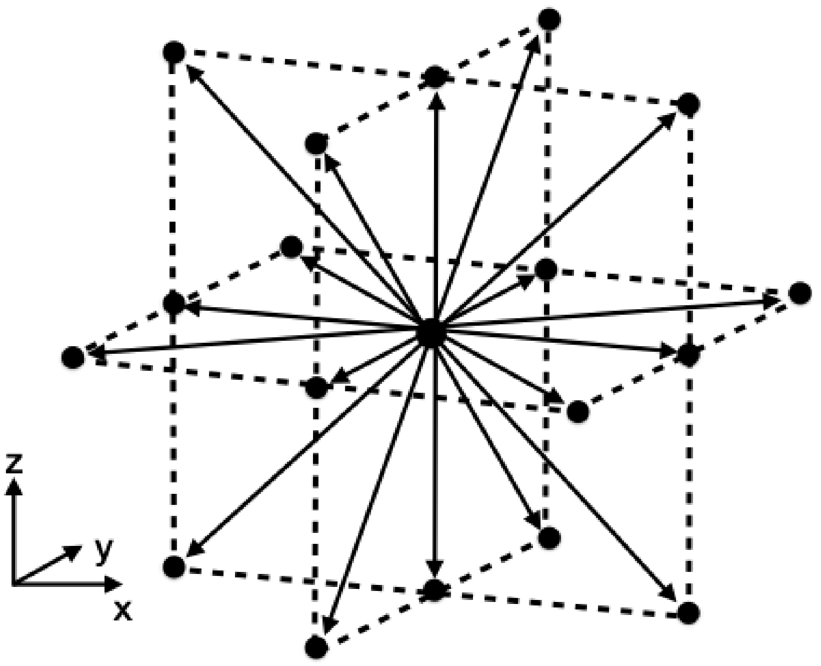


Figure 2.2: D3Q19 lattice showing the lattice directions. The numbering scheme is described in the text. [18]

2.2.2 Molecular dynamics (MD)

We employed a constant-volume molecular dynamics (MD) simulation to model the interaction of a fluid and a solid. We solved the equations of motion using the velocity Verlet algorithm [23] and controlled the temperature using the Nosé-Hoover thermostat [24][25]. Details of such simulations are given elsewhere [10][26]. Lennard-Jones potentials were used to describe the interactions between the atoms in the fluid and between the atoms in the fluid and the solid. Specifically the potential between atoms in the fluid takes the form [27]

$$U = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{2.8}$$

Details of the choice of potential parameters are described below. A cutoff distance of is used. The MD simulations were run in standard reduced units, with the reduced total, kinetic and potential energies given by $E^* = E/\epsilon$, $K^* = K/\epsilon$, and $U^* = U/\epsilon$, respectively. Other reduced quantities are: distance: $r^* = r/\sigma$; pressure: $P^* = P\sigma^3/\epsilon$; temperature: $T^* = T/\epsilon$; density: $\rho^* = \rho\sigma^3$; and time: $t^* = t/t_0$, where $t_0 = \sigma\sqrt{m/\epsilon}$, where m is the mass of the atom.

The constant velocity boundary condition at the top of the system continuously adds energy to the system, which, if left unchecked, would lead to viscous heating and a steady increase in the temperature. The resultant non-uniform temperature profile can be studied either by solving the heat equation with a source term representing viscous forces [28] or by using molecular dynamics simulations with a thermostat applied to the wall atoms [29]. A complete treatment of the viscous heating would require the incorporation of heat flow into the fluid flow simulation, either through a thermal lattice Boltzmann model [30] or by coupling an additional model for the heat flow to both the lattice Boltzmann and molecular dynamics models. For this paper, in which the goal is to examine the challenges inherent in linking models across scales, we chose to simplify the problem and include only a Nosé-Hoover thermostat on the fluid atoms, which we apply to the x-components of the velocities, perpendicular to both the flow direction and the height of the simulation cell.

2.2.3 Interface between lattice Boltzmann (LB) and molecular dynamics (MD)

As noted above, our goal is to couple a lattice Boltzmann simulation of continuum Couette flow with a molecular dynamics simulation used to model the interactions of the fluid with the underlying surface. These simulations are done in three dimensions, using the basic grid shown schematically in Figure 2.3. The system is divided into vertical cells with flow in the cells being calculated with the LB method. In a traditional simulation of Couette flow, the driving velocity in the y-direction would be set at the upper surface (z = 1 in Figure 2.3) and a zero-velocity boundary condition would be used at the bottom (z = 0) boundary. In the present simulation, we employ a MD simulation in the bottom cell to set the velocity boundary condition for the LB calculation. The MD simulation cell is shown schematically in Figure 2.4. The boundary condition at the bottom edge arises from the influence of the interaction of the fluid atoms with a set of stationary atoms in a face-centered cubic (fcc) structure. The strength of the interaction between the fluid atoms and the solid atoms, the well depth ϵ_s, was varied to examine the role it plays on the slip velocity. The top atoms (light gray) have a density and interactions appropriate for the fluid but are kept in a fixed fcc structure and are given a velocity in the y direction that comes from the LB calculation. The basic procedure is as follows: an LB calculation is run assuming a zero-velocity boundary condition at the bottom; the calculated velocity from the LB calculation at the top of the bottom cell is used to set the velocity at the top boundary of the MD calculation; the velocity at the bottom edge of the MD cell (the slip velocity) is determined (as described below); and the slip velocity sets the bottom boundary conditions for an LB calculation. The system is iterated to self-consistency.

2.2.4 Information transfer between models

In our approach, the LB and MD models have been written as autonomous codes that need input to run and that provide output. For the LB method, the needed input are the distance and time scales (Δx and Δt), the actual density, and the relaxation time τ. For MD, the needed input are the potential parameters (σ, ϵ, and ϵ_s) as well as atomic positions, temperature, cell dimensions, number of atoms, and time step, which are given in the code in reduced units. In addition to these parameters, the boundary conditions must be set. The LB method requires the velocity at the top surface (the driving velocity) and bottom surface (the slip velocity). The driving velocity is set as an input, while the MD calculations provide the slip velocity. The MD calculations require the average velocity at its top surface, which is provided by the LB calculation. One of the challenges in any multiscale simulation based on methods that describe different physical processes is to ensure a match between the fundamental thermodynamic, mechanical and transport properties described by each method. In general, the methods are based on different physical models and thus the information is calculated in very different ways. In the present case, we must ensure that the LB and MD calculations are modeling the same fluid, i.e., that each fluid has the same density, effective temperature, and viscosity. Details on making that connection are presented in the Appendix.

2.3 The Basic Model Inteface (BMI)

Our goal is to create a standard model-to-model interface by which we can link autonomous models into an integrated modeling system for multiscale simulations. A detailed discussion of the reasons behind such an approach and more details of how to construct these interfaces is given elsewhere by the CSDMS group [8]. The basic goal of the CSDMS system is to create a

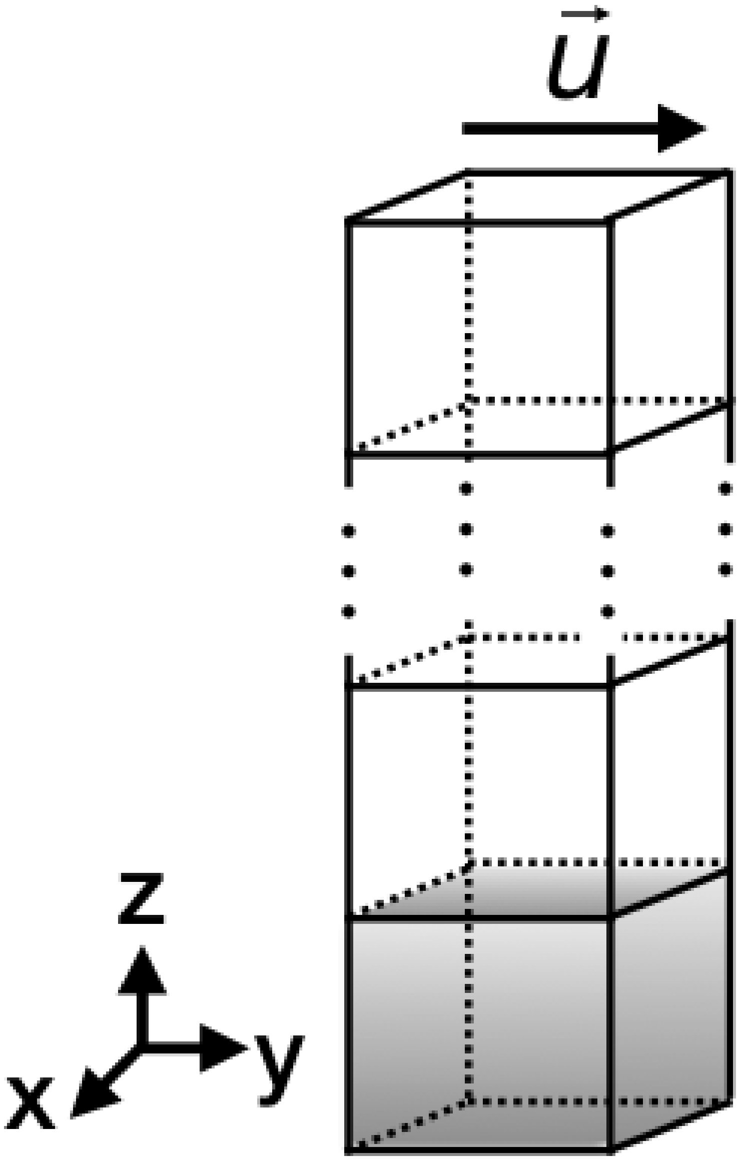


Figure 2.3: Schematic view of the Lattice Boltzmann grid, with the region in which molecular dynamics is used shown in gray.

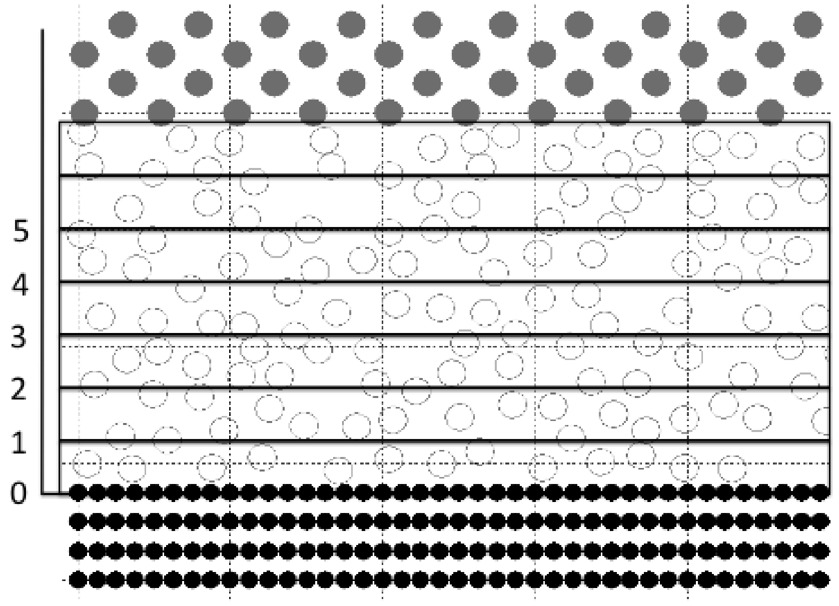


Figure 2.4: Schematic two-dimensional view of the molecular dynamics cell (a 2D projection). The dark colored atoms at the bottom represent the solid. The shaded atoms at the top boundary are given a velocity set by the results of the LB calculation. The rectangles indicate volumes over which averages are taken to determine the velocities as a function of z. Details are in the text.

common interface to link computer codes that have been developed by disparate members of their team to describe different aspects of the earth-surface system into an integrated simulation. They refer to these codes as “components,” which are equivalent to what we have been referring to as models. We will continue with the word model in the following discussion. We emphasize that we are not discussing an interface between a user and a computer program, such as through a GUI. Rather, we are constructing an interface between models. The interface has no data associated with it, but rather is a specific set of common functions that define how information is transferred to and from various models. This interface thus allows the models to be connected into an integrated modeling system. We use the word “model” in a very broad way. Each model is a computer code, which is written in some computer language and requires some input and yields some output, i.e., a “model” is an “information processor” that takes information in and sends new information out. The model may involve calculations that will be solved on a grid (or not) and will be written in some units (e.g., meters), which may or may not be consistent across all the models. The model could be a simple algebraic equation or it could be a complex computer simulation or a database – each model acts as a source of information. The model interface makes communication between models independent of the models themselves, making integration of the models easier. We believe that such an approach is essential for the efficient construction of integrated multiscale models in a environment in which models are widely different (e.g., lattice Boltzmann and molecular dynamics) and are likely to be developed and maintained by different groups. The benefits to this approach are that these groups can develop their models in any computer language; the models can be written in any sets of units; and the models can be calculated on any grid (or no grid). Each model can have multiple interfaces, so that we could link the same model in different ways to achieve multiple functionalities. For example, we could provide a set of interfaces to a molecular dynamics code that has many functionalities (constant volume and energy versus constant temperature versus constant stress, etc.). Thus this one model could be used in multiple ways by multiple people with no change in the fundamental computer code and can be reused in multiple integrated models without any special adaptations. The most important benefit is that constructing such autonomous models facilitates effective cooperation between groups, allowing each group to do what it does best

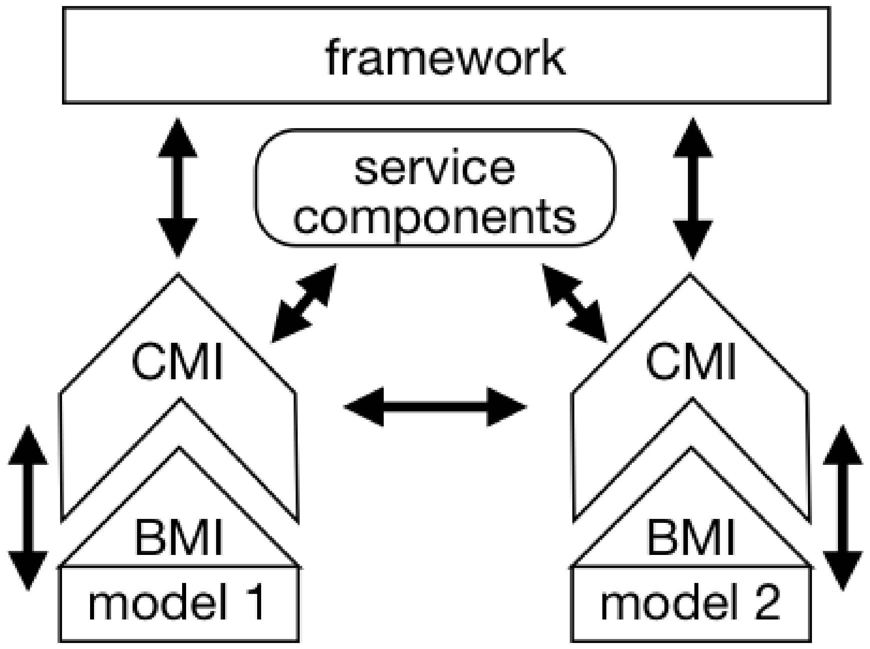


Figure 2.5: Relationship between the components of the CSDMS schema. Adapted from [7].

– creating new models without having to focus on developing a new framework. Our starting point is the CSDMS approach [8], which enables a model to communicate with other models in a way that is accessible to the developers of those models. The CSDMS scheme is to do this in a structured way, as shown schematically in Figure 2.5. The basic model interface (BMI) is a set of functions that is attached to each model to enable it to fit into a second-level interface called the component model interface (CMI). The BMI uses very simple data types and is easy to implement, which facilitates its use by the developer community [31]. The BMI provides all the information that enable the model to be "plugged into" the integrated modeling application. The functions in the CMI allow the models to communicate and share data with other models regardless of computer language [8]. Service components are background packages that all models use to provide basic compatibility between languages, etc. The framework provides the means by which the modeler can assemble the models into a connected application. We note that in this paper we are using the same programming language for all models and so are focused on creating a low-level basic model interface.

Modeling materials has a number of unique challenges, including a tight coupling between scales as described in Figure 2.1. Our work is thus centered in part on developing a better understanding of model-to-model communication in multiscale materials simulations, i.e., on the mediation of information transfer between models across scales. Our approach has been to start with a relatively simple hybrid system of multiscale models (described in Section 2) to examine key aspects of information mediation for linked (concurrent) multiscale simulations, including

- compatibility of information transfer between models (i.e., does information have the right units, attributes, etc.)
- boundaries between models (how do we match boundary conditions)
- convergence of the overall solution, especially with highly disparate models
- stability of the solution

The Basic Model Interface (BMI) consists of a standardized set of functions that allow access to a model's input and output variables in a standard way [31]. Communication between models is handled mainly by the use of get and set functions that return and change the value of variable respectively. The full name of these functions will include the data type and rank of the variable. Appropriate use of these functions allows access to any of the models's input or output variables. While the models are allowed to do their internal calculations in whatever systems of units is most convenient, the values returned by get or input into set should be in some accepted set of units. In our case, we use SI units. A list of functions and their descriptions is found in Table (2.1). We note that get and set functions exist for variables of every combination of data type and rank and that we only list a few of them in Table (2.1). While many of these functions are similar or identical to those from the BMI [8][31], new functions have been added for information about the potential boundary conditions that can be applied to a model and what input and output variables to use with a specific boundary condition. Specifically, the function `get_boundary_condition_names` returns types of boundary conditions that a model is capable of enforcing and `get_boundary_var_names(string)` returns a list of names of variables associated with a given boundary condition. The function `match_units(model *)` has also been added and is used to ensure that two models are simulating the same material under the same conditions. This function enables the model calling it to examine the state variables of the input model and modify its own state variables to match them, as discussed in the Appendix.

The framework in Figure 2.5 provides access to linking autonomous models within a run-time environment. To demonstrate the benefits of this approach, however, we are using a simpler system, in which there is a "controlling model" that serves the role as both the CMI and framework. The MD code and LB code are implemented as subclasses of the controlling model, so they inherit all of the functions, while the implementation for each function is specific to a given model. The BMI enables the models to communicate and share data with each other regardless of the language, though in our case the two models are written in a common language (C++). We emphasize again that since the models communicate through the functions in Table (2.1) (and others), it is immaterial that they are on a different grid (or no grid at all), use different units, or use a different time stepping scheme. The boundaries between models are controlled through the model-to-model interface. The controlling model creates and initializes a object of each model class, md and lb, and uses match_units to set the density, temperature, and viscosity to be the same in both models. It then connects the LB/MD simulations as described above.

2.4 Results of the Model

We developed and implemented a BMI-like interface to link the LB and the MD models, with the goal of (1) establishing a structure for connecting multiscale materials models and (2) demonstrating that this structure can be used to solve a specific problem, which, in this case, is to evaluate the effects of surface interactions at a fluid-solid interface. Here we describe the simulations in more detail and present some results. We note that detailed comparisons were made between an original monolithic code and the BMI-linked application, with no loss in computational speed or accuracy arising from use of the BMI-linked code.

The basic parameters in the modeled system are given in Table (2.2). The interaction potential between the fluid atoms and the underlying solid atoms (as shown in Figure 2.4), ϵ_s, is varied. The velocity profile in the y direction from an MD simulation is found by determining

Table 2.1: List of BMI-like functions

Function	Purpose
`void initialize(string input_file, string identifier)`	allocates memory for model and sets input variables

`void run(int time_steps)`
 	
`void finalize()`
	runs model for number of time steps based on value of time_steps
	deallocates memory for model and prints output to a file

`vector<string> get_input_var_names()`	returns list of input variables
`vector<string> get_output_var_names()`	returns list of output variables
`vector<string> get_boundary_condition_names()`	returns list of usable boundary conditions
`vector<string> get_boundary_condition_var_names(string boundary_condition)`	returns list of variables to use to enforce given boundary condition
`string get_var_type(string variable)`	returns data type of variable
`string get_var_units(string variable)`	returns units of variable
`int get_var_rank(string variable)`	returns rank of variable
`double get_0d_double(string)`	returns value of a zeroth rank floating point variable
`vector <double> get_1d_double(string)`	returns a first rank floating point variable
`void set_2d_double_at_index(string, double, int, int)`	set the value of a second rank floating point variable at a specified index
`int get_3d_int_at_index(string, int, int, int)`	return the value of a third rank integer variable at a specified index
`vector < vector < vector < vector < string> > > > get_4d_string(string)`	return a fourth rank string variable
`void match_units(model *)`	matches the values of variables in two models to put them into the same state

Table 2.2: Physical parameters input to each model
Δx, Δt, ρ_0, and τ are defined in Eq.(2.1), (2.2) and (2.4). N is the number of freely flowing fluid atoms in the molecular dynamics cell, m_a is the mass per atom, ϵ and σ are the potential parameters defined in Eq.(2.8), $a_{MD}$is the size of molecular dynamics cell ($= \Delta x$), V_{cell} is the volume of the molecular dynamics cell (and the LB grid volume), T is the temperature, and δt is the time step in the solution to the equations of motion. t_0 is the reduced unit of time in the calculation, as described in the text.

Lattice Boltzmann	Molecular dynamics
Δx = 5.78 nm	N = 2400
Δt = 4.363 x 10^{-11} s	m_a = 6.63 x 10^{-26} kg
ρ_0 = 1374 kg/m^3	ϵ = 120 K = 1.656 x 10^{-21}J
$\tau = 1$	σ = 0.34 nm
	a_{MD} = 17 σ = Δx
	V_{cell} = 1.93 x $10^{-25} m^3$
	T = 1.3 ϵ/k_b = 156 K
	$\delta t = 0.001 t_0$ = 2.2 x 10^{-15} s

the average velocity in small volume at different z values in the MD cell (indicated in Figure 2.4). These are then fitted with a line and the slip velocity (the value at the surface) is determined, which is passed to the LB model as the boundary condition at the bottom surface. The LB simulations were run for 1000 time steps before passing information to the MD calculations, which were then run for 20000 time steps to equilibrate the fluid with the new boundary condition and 600,000 time steps to create the velocity profile and to determine the slip velocity, which is passed to the LB calculation. This process continues for 20 cycles. The average of the slip velocity from the last 10 cycles is taken as the slip velocity and its standard deviation is taken as a measure of the error of the method.

We examined two types of substrate surfaces. In Figure 2.6a we show a “smooth” surface, consisting of the [100] surface of a face-centered cubic lattice. In Figure 2.6b, a “rough surface” is created by removing 1/2 of the surface atoms from Figure 2.6a, keeping a fourfold symmetry. We note that the interaction energy between these atomic-scale surfaces and the fluid is not sufficient, in general, to lead to zero slip velocity, which is why we are using a solid density at an unphysical value of four times that of the fluid [16]. To obtain non-zero slip velocities for the atomically rough surface we had to increase the driving velocity at the top of the Couette

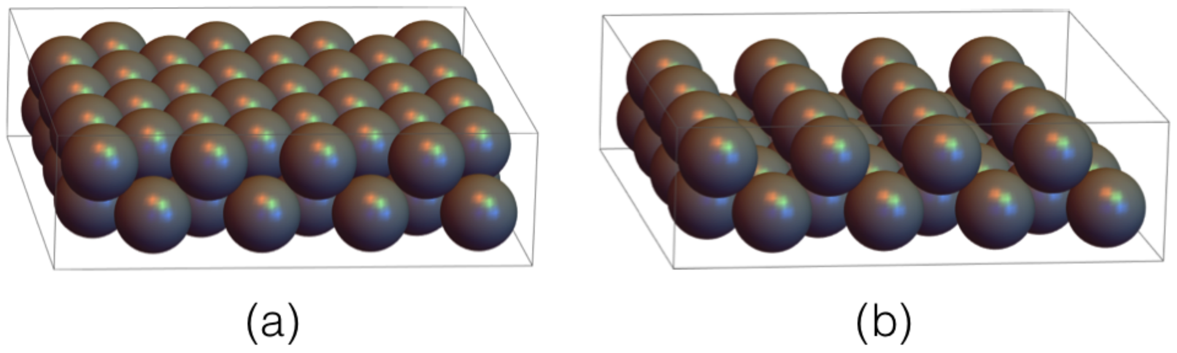


Figure 2.6: Atomic-level structures for the two surfaces used in our study. (a) The [100] surface of a face-centered cubic lattice. Note that it is smooth at an atomistic scale. (b) The surface from (a) with 1/2 of the atoms removed. Note that the surface structure has a fourfold symmetry.

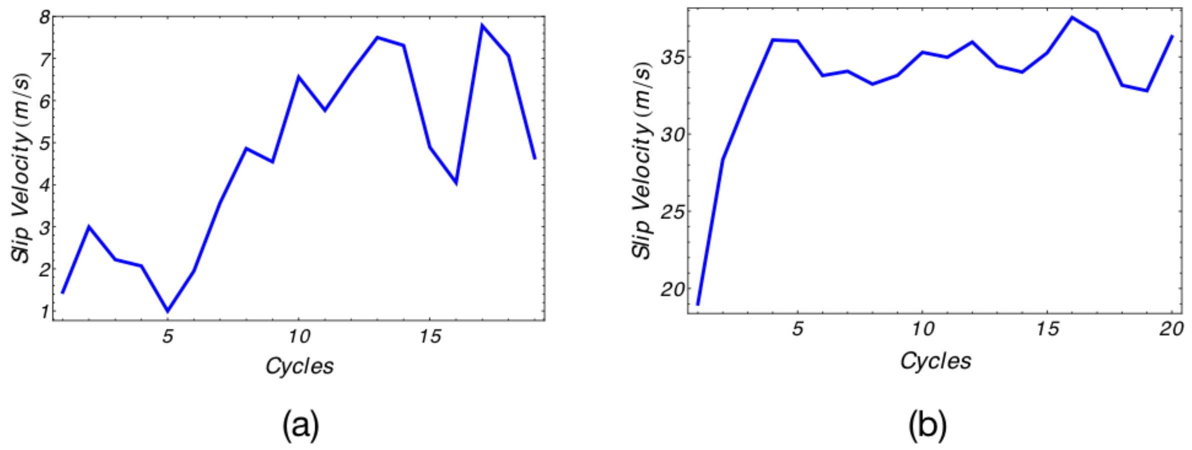


Figure 2.7: The slip velocities as a function of iterative cycle for (a) the atomically-smooth surface of Figure 2.6a and (b) the atomically-rough surface of Figure 2.6b.

flow relative to that used for a smooth surface (30 LB units, or 3974 m/s for the rough surface and 0.75 LB units or 99.3 m/s for the smooth surface).

In Figure 2.7, we show the calculated slip velocity as a function of the number of cycles in the iterative procedure. Calculations were done with $\epsilon_s/\epsilon = 0.2$ for the rough surface and 0.6 for the smooth surface. As noted above, the velocities with the smooth surface are small enough that numerical noise is relatively larger than that for the rough surface. That said, the results converge reasonably well for both, with larger relative error for the smooth surface. Note that the surface velocities are much smaller than the driving velocities (at the top of the LB model). For example, for the smooth surface, the ratio of the slip velocity to the driving

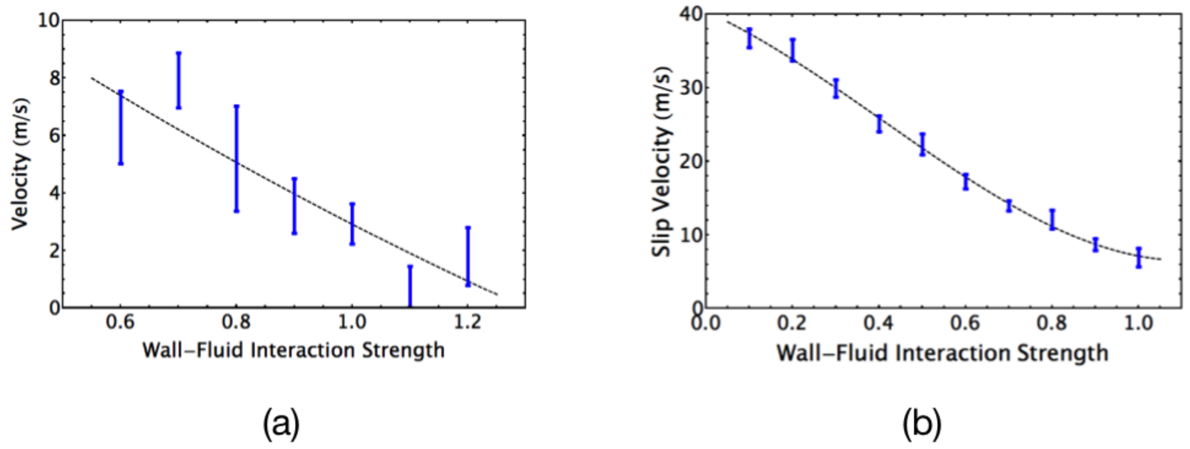


Figure 2.8: The slip velocities v_s (the average fluid velocity at the substrate surface) as a function of the strength of the interatomic interaction (ϵ_s/ϵ) between the atoms in the fluid and those in the solid substrate. (a) The atomically-smooth surface of Figure 2.6a. (b) The atomically-rough surface of Figure 2.6b.

velocity in Figure 2.7a is about $6/99 \approx 6\%$ and for the rough surface in Figure 2.7b, the ratio is about 0.8%. From a continuum perspective the assumption of a zero slip velocity may be a reasonable boundary condition, depending on the driving velocity and the strength of the fluid solid interaction.

In Figure 2.8, we show the converged slip velocity as a function of ϵ_s for the smooth and rough surfaces. In both cases, the stronger the interaction energy between the fluid atoms and the solid atoms, the lower the slip velocity, which is certainly as one would expect For the smooth surface (Figure 2.8a), the net interactions between the substrate surface and the atoms in the fluid were quite small, and very small net fluid velocities were sufficient to drive the atoms past the surface. Moving to an atomically rough surface changed the dynamics of the solid-liquid interface dramatically (Figure 2.8b), greatly increasing the effects of the surface on the fluid flow at the interface.

2.5 Summary and Conclusion

The results from the simulations of Couette flow are consistent with expectations and with previous work [16], with the stronger the interaction energy between the fluid atoms and the solid atoms, the lower the slip velocity (see Figure 2.6 and Figure 2.8). We found that an surface with roughness at an atomic scale enhanced the interactions between atoms in a fluid and the surface than that found with an atomically-smooth surface. The iterative approach taken to link the models was satisfactory, though one could imagine a number of other coupling schemes. The important part of this paper is that we established a basic model-to-model interface for linking autonomous multiscale computer models. We specifically examined key aspects of the mediation of the information between the models. A challenge for materials modeling is that the physical behavior at a specific set of length and time scales generally depends on physical processes at smaller scales, which can lead to complex boundary conditions between models. Because of this complexity, we focused on the key issues of

- compatibility of information transfer between models (i.e., does information have the right units, attributes, etc.)
- boundaries between models (how do we match boundary conditions for widely different length and time scales)
- convergence of the overall solution, especially with highly disparate models
- stability of the solution, especially with highly disparate models

We introduced new functionality to the Basic Model Interface of the CSDMS group [8] to handle these issues. Specifically, we introduced new functions to force compatibility between the LB and MD models to consistently represent fluids in the same state (detailed in the Appendix) and to set the boundaries between the models (detailed in Section 3). Convergence and stability were managed in the "controlling model" and were a product of the choice of an iterative link between scales. More sophisticated calculations will most likely require more sophisticated boundary conditions, which are under investigation. We are currently adding the model-to-model interface to other models (e.g., heat flow and a phase-field model) for application to problems of more relevance to materials (e.g., solidification). The goal is to create a library of models that we can link together as needed to form an integrated model system for a specific application that requires no further modification of the models in the library. We note that a more effective approach to developing this type of integrated model system would be to develop autonomous models with ontological and semantic independence [32]. Autonomy refers to models existing and running independently of other models. Ontological and semantic independence mean that each model's description of its own data and how the code is written is independent of how other models work and how they describe their data. Models could then be developed independently of one another and a way to link them together as necessary would be employed.

2.A Appendix

The parameters of the two models were chosen so that the fluid is in the same thermodynamic state, i.e., with the same temperature, density, and viscosity, in both simulations. The equivalence between units as calculated with the two methods is given in Table (2.A.1). After both models are initialized, the function `match_units(model *)` is called, which sets the value of the parameters dx and dt in the lattice Boltzmann simulation so that one cell in the lattice Boltzmann simulation is the same size as the entire molecular dynamics simulation and to match the viscosity of the two simulations. Temperature and unit cell size are input parameters to the molecular dynamics simulations, from which the density and viscosity can be determined. The viscosity for the molecular dynamics simulations are based on previous calculations for the Lennard-Jones fluid and tabulated as a function of pressure and temperature [33]. The temperature, simulation cell size, density, and viscosity are then passed to the lattice Boltzmann simulation. The viscosity, lattice spacing, density, and time step are related, so once the first three of these variables are known (passed from the molecular dynamics simulation) the time step is determined. All of the matching of variables and units is handled through the function `match_units(model *)` function.

Table 2.A.1: Equivalence of quantities between lattice Boltzmann and molecular dynamics

Property	Lattice Boltzmann	Molecular dynamics
Density	$\rho = \rho_0 \sum_{i=0}^{18} f_i = 1374 \frac{kg}{m^3}$	$\rho = \frac{4m_a}{(1.7\sigma)^3} = 1374 \frac{kg}{m^3}$
Temperature	$T = 156$K (arbitrary)	$T = 156$K (thermostat)
Viscocity	$\eta = \frac{\rho}{3} \frac{\Delta x^2}{\Delta t}(\tau - .5) = 1.75\text{x}10^{-4} \frac{kg}{ms}$	$\eta^* = 1.934$ $\eta = \eta^* \frac{\sqrt{m_a \epsilon}}{\sigma^2} = 1.75\text{x}10^{-4} \frac{kg}{ms}$
Velocity Conversion Factor	$v_{LB} = \gamma_{MD->LB} v_{MD}$ $\gamma_{MD->LB} = \frac{\Delta x}{\Delta t} \frac{\delta t}{a_{MD}} = .0493$	$v_{MD} = \gamma_{LB->MD} v_{LB}$ $\gamma_{LB->LMD} = 1/\gamma_{MD->LB} = 20.3$

CHAPTER 3. ADDING THE BASIC MODEL INTERFACE (BMI) TO LARGE-SCALE ATOMIC/MOLECULAR MASSIVELY PARALLEL SIMULATOR (LAMMPS)

3.1 Introduction

One of the long term goals of this project is to build a library of models that can be linked together via the BMI. To test the ability to add new models to this library, we would like to add the BMI to an existing code that has been developed independently by someone else and without its compatibility with the BMI in mind. Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) was chosen for this purpose. LAMMPS is an open source molecular dynamics code written in C++ developed at Sandia National Laboratories. We will be adding the BMI to LAMMPS and comparing it with our own molecular dynamics code by repeating the calculations performed in the previous chapter. Section 2 of this chapter will be a discussion of the implementation of the BMI with LAMMPS, Section 3 will show the results of the calculations, and Section 4 will be a summary of the results along with our conclusions.

3.2 Interface Implementation

As mentioned in the previous chapter, models are implemented as subclasses of the class *model*. This means that all objects of subclasses of *model* can act as members of *model*, such as in a list of *models*. Additionally, all functions defined for the class *model* can be used by its subclasses, a process known as inheritance [34]. The BMI functions are defined for the class *model* and all of its subclasses inherit these functions with each individual model having its own specific implementation of the BMI functions.

LAMMPS works by creating an object of the class *LAMMPS*, reading an input file, and translates each line into code to execute. Since *LAMMPS* is already a class, it can be used with the BMI by simply declaring it a subclass of *model* and adding the BMI functions to it.

The BMI `get_` and `set_` functions are implemented by determining an equivalent *LAMMPS* command, generating a `string` for that command, then passing that `string` to the function `one`() which executes the same code as if that string had used as a line in an input file. For example, setting the velocity on the top boundary uses the `velocity` command from LAMMPS to set the velocity on a specified group of atoms to the desired value, the relevant section of code is

```
void LAMMPS::set_1d_double(string variable, vector<double> value)
{
    std::stringstream ss;
    ss.str("");
    double units;

    if(variable == "velocity")
    {
        // conversion factor from real units to LAMMPS units
        units = _time/_length;
        value[0] *= units;
        value[1] *= units;
        value[2] *= units;

        // creates a string for the command to set
        // the velocity of group BMI_upper_boundary
        // to the value specified by value
        ss << "velocity␣BMI_upper_boundary␣set␣" << value[0]
        << "␣" << value[1] << "␣" << value[2] << "␣units␣box";

        // creates a char array with the same contents as ss
        const char *s = new char[100];
        s = &(ss.str())[0];

        // runs the command specified by the string
        input->one(s);
    }
}
```

With similar code existing setting the temperature and pressure and for calculating the slip velocity. For this to become a constant velocity boundary condition, two commands in the LAMMPS input file are necessary. One which sets the force on the group of atoms to always

Table 3.1: Physical parameters input to each model
$\mathit{\Delta x}$, $\mathit{\Delta t}$, ρ_0, and τ are defined in Eq.(2.1), (2.2) and (2.4). N is the number of freely flowing fluid atoms in the molecular dynamics cell, m_a is the mass per atom, ϵ and σ are the potential parameters defined in Eq.(2.8), $a_{MD}$is the size of molecular dynamics cell (= $\mathit{\Delta x}$), V_{cell} is the volume of the molecular dynamics cell (and the LB grid volume), T is the temperature, and δt is the time step in the solution to the equations of motion. t_0 is the reduced unit of time in the calculation, as described in the text.

Lattice Boltzmann	Molecular dynamics
$\mathit{\Delta x}$ = 5.78 nm	N = 2400
$\mathit{\Delta t}$ = 4.363 x 10^{-11} s	m_a = 6.63 x 10^{-26} kg
ρ_0 = 1374 kg/m^3	ϵ = 120 K = 1.656 x 10^{-21}J
$\tau = 1$	σ = 0.34 nm
	a_{MD} = 17 σ = $\mathit{\Delta x}$
	V_{cell} = 1.93 x $10^{-25} m^3$
	T = 1.3 ϵ/k_b = 156 K
	$\delta t = 0.001 t_0$ = 2.2 x 10^{-15} s

be zero and one which updates the positions of the atoms at every time step. Together these look like

```
fix          3 BMI_boundary setforce 0.0 0.0 0.0
fix          4 BMI_boundary nve
```

Units are handled within LAMMPS by defining a number of variables which give the conversion factor between real constants and the unit system used. The BMI looks at two of these variables, `angstrom` and `nanosecond`, which define the size of an angstrom and a nanosecond, respectively, to define the values for `_length` and `_time` which are used for unit conversion.

3.3 Results of the Model

The simulations performed in Chapter 2 are run again, this time using LAMMPS for the molecular dynamics portion of the simulations instead of our own molecular dynamics code. For reference, the simulation parameters are repeated here in Table (3.1).

The results of both sets of simulations are shown in Figure 3.1 and Figure 3.2. As seen in Figure 3.1, the convergence behavior for both appear to be similar, the slip velocity approaches the true value after about 10 cycles then fluctuates around its mean for the rest of the sim-

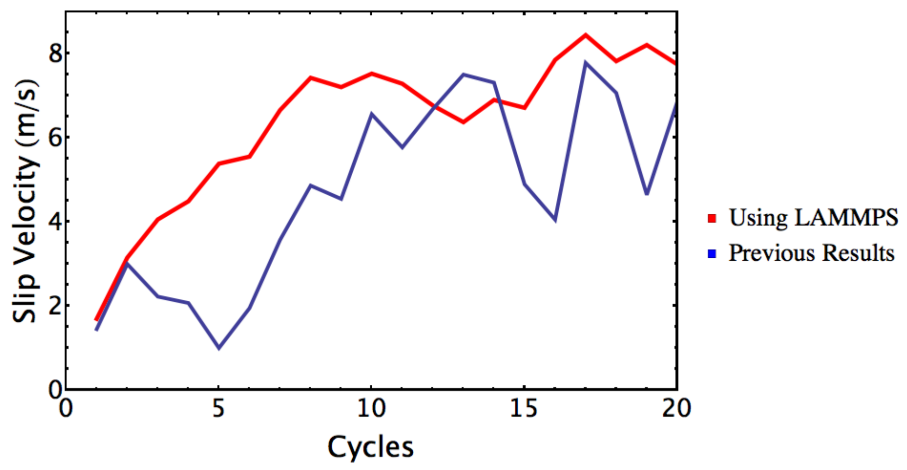


Figure 3.1: Comparison of the slip velocity as a function of iterative cycle for the smooth surface with $\epsilon_s/\epsilon = .6$.

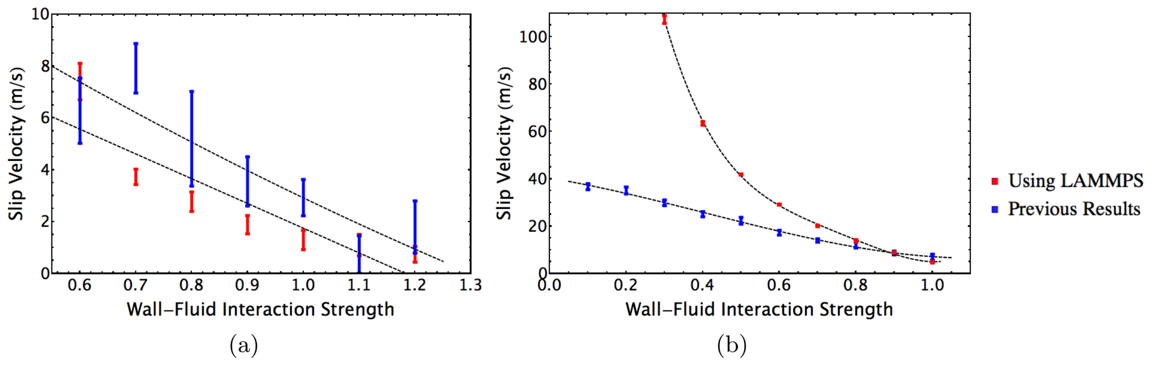


Figure 3.2: Comparison of the slip velocity as a function of ϵ_s/ϵ. (a) The atomically smooth surface. (b) The atomically rough surface.

ulations. For the smooth surface Figure 3.2a shows that calculations using LAMMPS give a higher value for the slip velocity for the $\epsilon_s/\epsilon = .6$ and $\epsilon_s/\epsilon = 1.1$ cases and smaller values for the slip velocity in the other cases. Additionally the standard deviation of the slip velocity over the second half of the simulation, shown as half the height of the vertical bars for each data point, is much smaller, less than half, in all cases. For the rough surface Figure 3.2b shows that the calculations using LAMMPS tend to give larger values for the slip velocity, especially at lower values of ϵ_s/ϵ. Though difficult to see from Figure 3.2b, the standard deviation of slip velocity is smaller when using LAMMPS for all cases except $\epsilon_s/\epsilon = .3$.

3.4 Summary and Conclusion

By adding the BMI to LAMMPS we demonstrated the ability to add the BMI to an existing code. This modified version of LAMMPS was then able to communicate with our previously written lattice Boltzmann model and was used to repeat previous simulations of Couette flow. The results using LAMMPS are similar to the results from the previous chapter. The differences could be do to a number of possible factors: A simulation parameter that has accidentally not been made identical in the two simulations, slight differences between the algorithms used by the two molecular dynamics codes, an unknown bug in either code, etc.

It has been mentioned previously that one of the advantages of the BMI is that it can be added to models without modifying their code. Unfortunately, this may not be the case for LAMMPS. While creating a function to calculate the slip velocity based on the velocity profile, it was discovered that the velocity profile values are not directly accessible so a simple function which outputs these values had to be added to the `fix` and `fix_ave_chunk classes`. This is regarded as a temporary solution to this problem. Different methods of accessing this information and a solution which does not require the modification of LAMMPS code are still being sought.

REFERENCES

[1] Jacob Fish. “Bridging the scales in nano engineering and science”. In: *Journal of Nanoparticle Research* 8.5 (2006), pp. 577–594.

[2] Jitesh H Panchal, Surya R Kalidindi, and David L McDowell. “Key computational modeling issues in integrated computational materials engineering”. In: *Computer-Aided Design* 45.1 (2013), pp. 4–25.

[3] Gerard F Laniak et al. “Integrated environmental modeling: a vision and roadmap for the future”. In: *Environmental Modelling &amp; Software* 39 (2013), pp. 3–23.

[4] DJ Muth and Kenneth Mark Bryden. “An integrated model for assessment of sustainable agricultural residue removal limits for bioenergy systems”. In: *Environmental modelling &amp; software* 39 (2013), pp. 50–69.

[5] C. W. Armstrong, R. W. Ford, and G. D. Riley. “Coupling integrated Earth System Model components with BFG2”. In: *Concurrency and Computation: Practice and Experience* 21 (2009), pp. 767–791.

[6] V. Balaji. *ESMF Reference Manual for Fotran*. 2014.

[7] R. Redler, S. Valcke, and H. Ritzdorf. “OASIS4 - a coupling software for next generation earth system modelling”. In: *Geoscientifici Model Development* 3 (2010), pp. 87–104.

[8] Scott D. Peckham, Eric W.H. Hutton, and Boyana Norris. “A component-based approach to integrated modeling in the geoscience: The design of CSDMS”. In: *Computers & Geosciences* 53 (2013), pp. 3–12.

[9] *Babel Homepage*. https://computation.llnl.gov/casc/components/index.html.

[10] Mike F Ashby. “Physical modelling of materials problems”. In: *Materials Science and Technology* 8.2 (1992), pp. 102–111.

[11] Richard LeSar. *Introduction to computational materials science: fundamentals to applications*. Cambridge University Press, 2013.

[12] Philip W Anderson et al. “More is different”. In: *Science* 177.4047 (1972), pp. 393–396.

[13] Somnath Ghosh. “Adaptive Hierarchical-Concurrent Multiscale Modeling of Ductile Failure in Heterogeneous Metallic Materials”. In: *JOM* 67.1 (2015), pp. 129–142.

[14] Qi Tong and Shaofan Li. "From molecular systems to continuum solids: A multiscale structure and dynamics". In: *The Journal of chemical physics* 143.6 (2015), p. 064101.

[15] Franck J Vernerey and Mirmohammadreza Kabiri. "An adaptive concurrent multiscale method for microstructured elastic solids". In: *Computer Methods in Applied Mechanics and Engineering* 241 (2012), pp. 52–64.

[16] Nikolaos Asproulis, Marco Kalweit, and Dimitris Drikakis. "A hybrid molecular continuum method using point wise coupling". In: *Advances in Engineering Software* 46.1 (2012), pp. 85–92.

[17] Shiyi Chen and Gary D Doolen. "Lattice Boltzmann method for fluid flows". In: *Annual review of fluid mechanics* 30.1 (1998), pp. 329–364.

[18] Uriel Frisch, Brosl Hasslacher, and Yves Pomeau. "Lattice-gas automata for the Navier-Stokes equation". In: *Physical review letters* 56.14 (1986), p. 1505.

[19] Bastien Chopard. "Cellular automata modeling of physical systems". In: *Encyclopedia of Complexity and Systems Science.* Springer, 2009, pp. 865–892.

[20] Dominique d'Humières. "Multiple–relaxation–time lattice Boltzmann models in three dimensions". In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 360.1792 (2002), pp. 437–451.

[21] Carolin Körner, Elham Attar, and Peter Heinl. "Mesoscopic simulation of selective beam melting processes". In: *Journal of Materials Processing Technology* 211.6 (2011), pp. 978–987.

[22] Chih-Fung Ho et al. "Consistent Boundary Conditions for 2D and 3D Lattice Boltzmann Simulations". In: *Computing Modeling in Engineering and Science* 44,2 (2009), pp. 137–155.

[23] Loup Verlet. "Computer" experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules". In: *Physical review* 159.1 (1967), p. 98.

[24] Shūichi Nosé. "A molecular dynamics method for simulations in the canonical ensemble". In: *Molecular physics* 52.2 (1984), pp. 255–268.

[25] William G Hoover. "Canonical dynamics: equilibrium phase-space distributions". In: *Physical review A* 31.3 (1985), p. 1695.

[26] Daan Frenkel and Berend Smit. *Understanding molecular simulation: from algorithms to applications.* Vol. 1. Elsevier (formerly published by Academic Press), 2002, pp. 1–638.

[27] John Edward Jones. "On the determination of molecular fields. II. From the equation of state of a gas". In: *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences.* Vol. 106(738). The Royal Society. 1924, pp. 463–477.

[28] J. N. Reddy. *An introduction to continuum mechanics with applications*. Cambridge University Press, 2008.

[29] Xin Yong and Lucy T Zhang. “Thermostats and thermostat strategies for molecular dynamics simulations of nanofluidics”. In: *The Journal of chemical physics* 138.8 (2013), p. 084503.

[30] Xiaoyi He, Shiyi Chen, and Gary D Doolen. “A novel thermal model for the lattice Boltzmann method in incompressible limit”. In: *Journal of Computational Physics* 146.1 (1998), pp. 282–300.

[31] *CSDMS Basic Model Interface*. http://csdms.colorado.edu/wiki/BMI_Description.

[32] Kenneth Bryden. “A Proposed Approach to the Development of Federated Model Sets”. In: *International Congress on Environmental Modelling and Software* (2014).

[33] RL Rowley and MM Painter. “Diffusion and viscosity equations of state for a Lennard-Jones fluid obtained from molecular dynamics simulations”. In: *International journal of thermophysics* 18.5 (1997), pp. 1109–1121.

[34] Bjarne Stroustrup. *The C++ programming language*. Pearson Education, 2013.