Accurate and Efficient Multi-scale Flow Simulation by A Hybrid CFD-MD Approach with Its Runtime Environment

Soon-Heum Ko[1], Nayong Kim[1], Abhinav Thota[1,2] (?),
Dimitris Nikitopoulos[3], Dorel Moldovan[3] (?), Shantenu Jha[*1,2]

[1]*Center for Computation & Technology, Louisiana State University, USA*

[2]*Department of Computer Science, Louisiana State University, USA*

[3]*Department of Mechanical Engineering, Louisiana State University, USA*

[*]*Contact Author*

Abstract

****Jeff: To be included later. Abstracts from ECCOMAS and CCGrid are available; we can merge these.*

I. Introduction and Motivation

These days, with the emphasis on accurately solving the micro-scale fluid systems for the bio-fluidic product design, a modern numerical technique, called a hybrid computational fluid dynamics (CFD) - molecular dynamics (MD) approach [**?**],[**?**],[**?**],[**?**], is getting paid more attention to. In a word, a hybrid CFD-MD approach can be defined as adopting the continuum hypothesis in capturing macroscopic flow features while solving low-speed flow regions - whether it is near the stationary wall or not - by considering atomistic intermolecular interactions. CFD can accurately predict flow properties on conventional moderate/large size fluid domains, but is intrinsically impossible to reflect characteristics of surrounding solid materials. While MD can provide atomistic level resolution of interactions between particles, it becomes computationally demanding as the size of simulated system grows. The hybrid approach provides a good balance between computational cost and atomistic details/resolution.

An example of the macroscopic flow changes due to different atomistic interaction in the infinitesimal region is presented in Fig. **??**. Steady-state solutions of Couette flow by MD simulations are presented at different slip ratio between wall and fluid particles. In macroscopic point of view, molecular dynamic simulations describe the same flow physics as the continuum hypothesis, in that converged flow shows the same velocity gradient along the vertical direction. On the other hand, the consideration of molecular interaction leads to the change of the microscopic flow pattern near the wall, which results in the diversity of velocity gradient according to different slip ratio. This is what continuum simulations cannot present without special treatments on boundary conditions. This motivates the necessity to consider intermolecular effect on CFD solution procedure.

The cost of force calculation to evaluate the potential energy is the slowest part of MD simulation. The computational cost of a classical MD simulation is very expensive, $O(N^2)$ in $bigO$ notation with N particles, i.e. it will be demanded more and more CPU cost when the number of particle increase in MD simulation [**?**]. ***Jeff: I could not clearly understand the meaning of this paragraph. Let me rewrite the text after getting more information from Nayong. ***Nkim: This sentence is related with the explanation about MD particle size VS simulation time. In classical MD, simulation time will increase as ratio of itN^2 when number of atom N increase. The exact relationship b/w them has too many factors including number of CPU, number of atoms, type of computation, size of memory and so on. The point of this sentence is that MD's computation time dramatically increases as increasing number of atom. So, it is hard to simulate whole physical domain using pure MD. It will be one of the main purpose why we need the hybrid simulation.

The above benefit of hybrid CFD-MD approach over pure CFD or pure MD simulations attracted the interest of many CFD and MD researchers, which resulted in the fast progress of hybrid schemes in 15 years since the first work [**?**] was in press, as to be addressed in Section 2. Nevertheless, it is still very hard for an individual scientist to apply this approach to his/her domain of interest. We assume that the difficulty to determine the state-of-the-art coupling condition is one possible reason. Even with accurate numerical schemes for solving each domain and

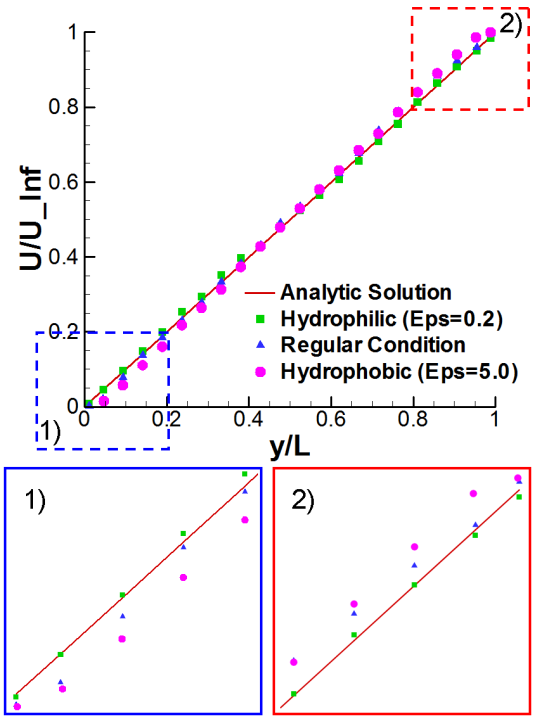

Fig. 1. Steady Couette Flow Solution by Pure Molecular Dynamic Simulation; In normal condition (with the unique potential well depth between flow particle and surface material, denoted by ϵ), the solution by MD is identical to the result of continuum approach. In hydrophobic case (smaller ϵ) the fluid shows a slight slip on the wall, while the fluid particle strongly attempts to stick to the wall in hydrophilic case (bigger ϵ).

coupling hybrid interface, the solution is strongly affected by the specification of coupling parameters. Layer size of data exchange zone with its position, sampling duration and the interval are major factors which affect the accuracy of coupled solution, and these conditions vary by the fluid and solid elements, flow condition and geometric characteristics. So far, there is no globally acceptable scheme for setting these coupling conditions. Thus, a number of experiments are required to empirically determine the coupling condition for specific problem.

In addition to the numerical endeavour of determining coupling condition between CFD and MD domains, there also exist computational challenges of integrating multiple application domains into a single problem set. Considering very different computational kernels (one could be mesh-based, the other unstructured particle simulations), it is nearly impossible to incorporate distinct CFD and MD codes under the umbrella of a single tightly-coupled application (i.e., binding two application codes within a single MPI communicator).

One possible alternative will be to implement coupling interface on individual code and control these logically separated codes as a virtually unified simulation package. However, confined to parallel execution on conventional production system with batch queue, it cannot be guaranteed that two separate jobs will execute concurrently. Considering the computational characteristics of current application, where CFD and MD codes conduct frequent information exchange via their file interfaces, the first job to run will inevitably experience the idle waiting for its counterpart without the explicit support for co-scheduling. Another important challenge is the need for efficient load-balancing which takes into account the individual application's performance. Even if the two simulations could run concurrently, without explicit load-management/balancing support, there is likely to be inefficient utilization of compute resources due to load imbalance. As the performance of each simulation component changes with computing resource and problem size, re-adjustment of allocated resources to each task according to their performance is required during the simulation.

Collectively, we are focusing on investigating numerical issues of applying the hybrid CFD-MD scheme to a nano-fluidic system, as well as designing and developing an efficient runtime framework for a coupled multi-component simulation. We consider the implementation of a 'generic' hybrid interface which can be easily attached to various kinds of incompressible CFD codes and MD packages, and the building of a 'portable' framework which is acceptable for most computer architectures. Regarding numerical simulations, we value our idea to determine coupling parameters is reasonable and easy to follow. Also we argue that our numerical experiments cover broader range of system scales and flow physics, than other individual works, though each simulation may not be the first trial for specific problem. Especially for physically unsteady flow simulation, our design of temporal coupling scheme can be one possible challenge to overcome the time-lagging in hybrid boundary. ***Jeff: Maybe the above sentence will be changed according to new numerical solution. In view of computational science, we believe our trial is the first documented coupled multi-physics simulation utilizing a virtually unified simulation package, called "Pilot-Job". We claim that there are several distinct advantages of using Pilot-Job: (i) obviates the need for a co-scheduler while preserving performance, (ii) enables dynamic resource allocation, which in turn is important for load-balancing across coupled simulations.

We begin the next section with an outline of the concept and numerical issues of coupled simulations. Numerical details of individual code and implementation of hybrid scheme are presented in the next Section. Another objective of current research, the construction of an efficient CFD-MD simulation framework, will be discussed in Sec-

tion 4. In Section 5, we will demonstrate our numerical results of a famous time-accurate validation problem (Couette flow simulation) and the physically unsteady flow simulation (oscillating boundary problem), as well as the performance of this coupled simulation, which will prove the accuracy and efficiency of our hybrid simulation framework. Finally, our next plan and the summary of current work are expressed in Section 6 and 7.

II. A Hybrid CFD-MD Approach - Numerical Technique for Multi-scale Flow Simulation

Hybrid continuum-molecular simulation approaches can be grouped into two according to the microscopic solution method coupled into macroscopic continuum domain. [**?**] One is the coarse-grained simulation by using Direct Simulation Monte Carlo (DSMC) for dilute gas [**?**],[**?**] or Dissipative Particle Dynamics (DPD) for liquid [**?**],citeFedosov2, where the simulation molecule represents a number of atoms/molecules. The coarse-grained simulation can expect better computational cost and enables the study of temporal evolution for longer time, while the state-of-the-art modeling is required to correctly impose the wall boundary condition.

The other group of researches, which directly couples traditional MD solvers with continuum solvers, can be classified to alternating Schwarz method [**?**],[**?**],[**?**],[**?**],[**?**], direct flux exchange [**?**],[**?**],[**?**],[**?**],[**?**], and constrained Lagrangian Dynamics [**?**],[**?**],[**?**],[**?**],[**?**],[**?**],[**?**]. Comparing initial formulations of above methods, in alternating Schwarz method, CFD and MD solvers iterate at decoupled time space and exchange the conserved properties until two solutions are converged (i.e., CFD and MD solutions in the overlap region becomes identical). Thus, in view of computational cost, this method is good for solving steady or quasi-steady flow in larger domain of interest. On the other hand, the direct flux exchange approach exchanges the flux properties from both solvers at the same physical time. Consideration of more properties is adequate for capturing the secondary flow, thus this method is expected to give better resolution in solving highly variational unsteady flowfield in nanoscale. Finally, constrained Lagrangian dynamics exchange the conserved properties in coupled time domain. This enables cost-effective simulation of unsteady flowfiled when secondary flow normal to principal direction is not strong. Meanwhile, debating on one's superiority over other models becomes meaningless these days as each technique is evolving to accept others' benefit. For example, external force function design by Werder et al [**?**] starts from the force function in constrained Lagrangian dynamics and direct flux exchange, and unsteady flow simulation by Liu et al [**?**] adopts the quasi-steadiness concept in alternating Schwarz method. ***Jeff: Maybe the history of each category will be included if we have time

In the current work, we adopt a constrained Lagrangian dynamics proposed by Nie et al [**?**], considering our numerical interest. We aim to have an efficient coupled simulation framework with a generic interface implemented on individual "blackbox" solver. Thus, getting abundant communication cost on convergence check is not preferred and the minimal change on each code is a plus.

Composition of computational domain in hybrid simulation is schematized in Fig. **??**. CFD approach solves the external zones with the moderate flow velocity and MD analyzes the microscopic flow feature near the stationary obstacle or in the low-speed region. These two domains overlap in the middle, where CFD and MD exchange their solutions. Hybrid region is placed sufficiently far from the solid obstacle to prevent the direct influence of strong fluctuation due to the intermolecular collision between solid and fluid particles. Also, this region is set up sufficiently large to include five layers with enough spacing. From the bottom, we have the boundary zone for CFD and MD simulations and have additional zone to exert external force. These three interesting zones are isolated by having buffer layer in between. In the boundary zone of continuum dynamics, particles' velocities are averaged over time and this is directly used as CFD boundary condition. The height of each layer is the same as the CFD cell height and averaged conservative properties in two consecutive layers are passed to continuum domain to impose viscous boundary condition in Navier-Stokes solver with non-staggered data structure. Next, the outer boundary zone of molecular dynamics domain is placed in the middle. In this region, the continuum solution is transferred to MD code and particles are guided to eventually follow this macroscopic flow velocity, by solving an equation called 'constrained MD equation'. The strong point of this equation is that, molecules are led to eventually follow the macroscopic flow property, while preserving the maximal degree-of-freedom of molecular motion. In the uppermost layer, artificial external force is exerted on particles to maintain numerical stability. This force function should be designed not too strong to influence the motion of particles in the domain of interest nor not too weak to break up the initial hypothesis of molecular dynamic simulation, termed the conservation of statistical ensemble. Finally, buffer layer is positioned in between each layer. This layer is set up to be bigger than the length scale of interacting particles (cutoff radius), not to have a solution in one layer being directly interfered by another solution.

***Jeff: It would be better to explain a little more detail on last sentence. Let's say the buffer between CFDtoMD and MDtoCFD boundary is smaller than cutoff length. Then, a solution at CFDtoMD (constrained MD) directly

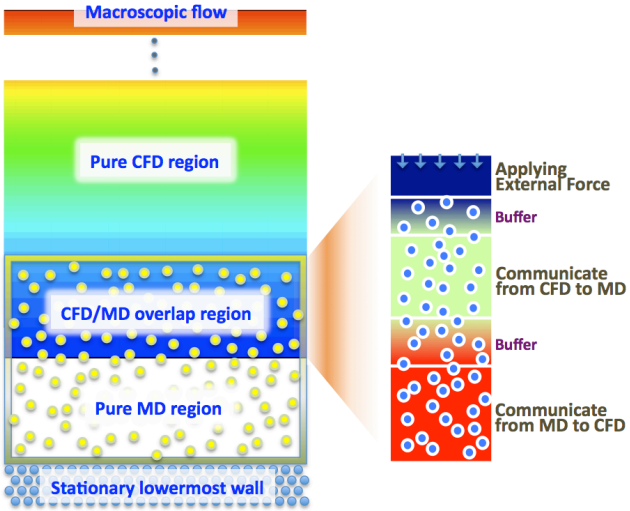

Fig. 2. Schematic Diagram of the Hybrid Domain with Detailed View of Overlapping Zone; Overall continuum/atomistic computational domain including overlap region is shown on left figure. Detailed layer by layer explanation of overlapping region is indicated by right figure.

affects the motion of particles at MDtoCFD. And this is used for CFD boundary condition. This results that, CFD solution at CFDtoMD is the result of flow variation in MDtoCFD boundary, which in turn is directly affected by the CFDtoMD solution. Thus, solution at CFDtoMD becomes the boundary condition of CFD domain.

The composition of layers in hybrid region changes to unify the external force layer and molecular dynamics boundary layer in alternating Schwarz method or direct flux exchange. Therefore, the design of external force is very important to exert the correct mean pressure on molecular dynamics domain without local disturbance in the density field, as is expressed by Werder et al [**?**]. On the other hand, in the domain composition of constrained Lagrangian dynamics, particles in the buffer layer between external force layer and molecular dynamic boundary layer act as decreasing the influence of non-uniform density distribution. Thus, a cost-effective classical external force model by Nie et al [**?**] looks still valid for acquiring the accurate solution.

For time-accurate unsteady solution by a hybrid method, temporal coupling scheme is another important issue. Temporal coupling scheme denotes the scheduling of data exchange routine in time domain between CFD and MD solvers, to synchronize both solutions at the same physical time. Two coupling parameters shall be addressed to set up coupling mechanism in time space, i.e., how often data exchange will take place (called 'sampling interval') and how long the averaging will be conducted in particle domain (called 'sampling duration'). Spatially averaged particle velocity at an instance tends to be very noisy due to the natural physics of molecular vibration, thus molecular dynamic solution should be averaged over enough sampling duration to reduce this statistical error. Following the stable condition for sampling duration in steady [**?**] and unsteady [**?**] flowfield or suppressing the noise by virtual damping terms [**?**] or dynamic parameter for coupling intensity [**?**] can be referenced. Sampling interval is usually set up to be longer than sampling duration.

Two conventionally used coupling approaches of synchronized coupling and sequential coupling strategy are depicted in Fig. **??**. In synchronized coupling, CFD and MD codes exchange the information in the overlapping region at the data exchange point (denoted by t) and independently advance by their sampling interval (Δt). On the other hand, one domain advances to the next exchange point and leads its counterpart to approach this point in sequential coupling. Comparing these two mechanisms, synchronized coupling clearly have a far better parallel efficiency compared to sequential coupling. [**?**] Parallel performance in synchronized coupling can ideally reach the single code running in case an appropriate load balancing between two domains is applied, while large overhead on waiting is inevitable in sequential way. On the other hand, sequential approach is expected to produce more time-accurate solution because the boundary condition on following domain (CFD domain in the figure) is implicit. Time-lagging phenomenon in CFD boundary region is observed in both strategies, because averaged molecular dynamic properties over backward sampling duration (from $t-\Delta t_S$ to t) are representing the solution at that instance (t). As was pointed out by Wang and He [**?**], additional numerical treatment is required to eliminate this time-lagging, either extrapolating the boundary condition from previous solutions or setting the sampling interval far longer than sampling duration (assuming quasi-steady).

III. Development of A Hybrid CFD-MD Simulation Toolkit

A. Features of Baseline CFD and MD Codes

The current in-house continuum hydrodynamics code solves the unsteady incompressible Navier-Stokes equations to demonstrate the isothermal nano-scale flow field:

$$\frac{\partial u_i}{\partial x_i} = 0$$
$$\frac{\partial u_i}{\partial t} + \frac{\partial}{\partial x_j}(u_i u_j) = -\frac{\partial p}{\partial x_j} + \nu \frac{\partial u_i}{\partial x_j \partial x_j} \quad (1)$$

where ν is the kinematic viscosity.

In this work, we adopted the pseudo-compressibility method [**?**] to form a hyperbolic system of equations

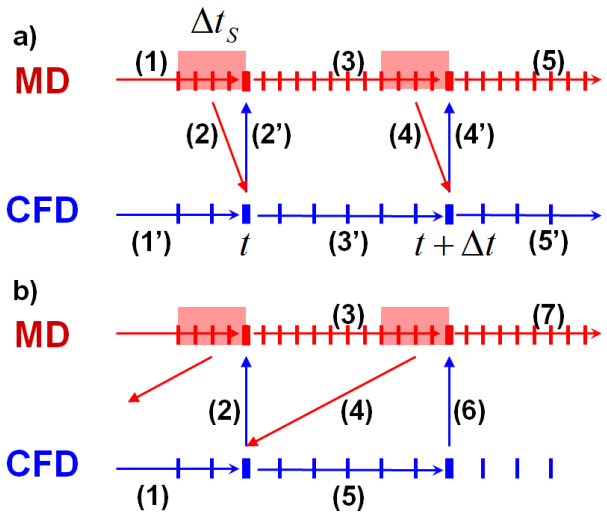

Fig. 3. Conventional Time Evolution Mechanisms of a Hybrid CFD-MD Approach; CFD and MD codes are scheduled to conduct data exchange in the overlapping region at every Δt sampling interval. CFD solution at t is directly applied as the boundary condition for MD simulation and backward time-averaged molecular dynamic properties over Δt_S sampling durations are imposed as CFD boundary conditions. a) Synchronized Coupling: Both codes communicate at the same time level and independently advance to next exchange point. b) Sequential Coupling: From the same time level, one solver advances to the next communication point and impose implicit boundary condition to its counterpart.

which can be marched in pseudo-time. A time derivative of pressure is added to the continuity equation resulting in

$$\frac{\partial(p/\rho)}{\partial\tau} = -\beta\frac{\partial u_i}{\partial x_i} \tag{2}$$

where β denotes a pseudo-compressibility parameter, currently set up as 2.5.

For time-accurate unsteady simulation, dual time stepping method is adopted and it is combined with the LU-SGS (Lower-Upper Symmetric Gauss-Seidel) scheme [**?**] for the implicit time integration. The inviscid fluxes are upwind-differenced using Osher's flux-difference splitting scheme [**?**]. For higher-order spatial accuracy, the MUSCL (Monotone Upstream-centered Schemes for Conservation Laws) [**?**] approach is used on inviscid flux calculation. Viscous fluxes are calculated using the conventional second-order central differencing.

Molecular dynamics computer simulation technique is a specified computer simulation method for molecular systems, including microscopic details of a system and macroscopic statistical properties, which are the properties of the atoms, the interactions between particles, molecular characteristics, structure of molecules, transport phenomena etc. [**?**],[**?**],[**?**] In molecular dynamics, an initial velocity is assigned to each atom, and Newton's laws are employed at the atomic level to propagate the system's motion through time evolution:

$$F_i = m_i a_i \tag{3}$$

here each atom i in a system constituted by N atoms, mi is the mass of ith atom, ai is the acceleration and denote by $d^2 r_i/dt^2$, and F_i is the force on ith atom, due to the atomic interaction which is calculated based on the potential energy between individual particles. The simplest choice for the potential energy $U(r)$ can be written as the sum of pairwise interactions of particles:

$$U(r_1, ..., r_N) = \sum_i \sum_{j>i} u(|r_i - r_j|) \tag{4}$$

where i and j particles are located at r_i and r_j, and only $j > i$ cases have been considering for each particles pair once. The most commonly used two-body interaction model is the Lennard-Jones (12-6) potential is applied to calculate pairwise interaction and is define as:

$$u(|r_i - r_j|) = 4\epsilon_{ij}[(\frac{\sigma_{ij}}{r_{ij}})^{12} - (\frac{\sigma_{ij}}{r_{ij}})^{6}] \tag{5}$$

where ϵ_{ij} and σ_{ij} denote the pairwise potential well depth and the atom size parameter respectively, and r_{ij} is the distance between the particle i and j. The term $1/r_{ij}^{12}$ dominating at short range distance repulsive behavior based on the Pauli principle to avoid overlapping the electronic clouds when particles are brought very close to each other. The term $1/r_{ij}^{6}$ dominates at long range attractive forces by van der Waals dispersion forces. ***Nkim: cut-off The cut-off distance σ_c is introduced here to reduce the computational cost and is set to be 2.2σ [**?**].

The time integration algorithm is required to integrate the equation of motion of the interacting particles and computing molecular trajectories, one of most common velocity Verlet algorithm is employed to compute the simulation. In this work, the MD simulations were performed by using the modified version of Large Atomic Molecular Massively Parallel Simulator(LAMMPS). It is the classical molecular dynamics open software written in C++ and developed by Sandia National Labs. [**?**]

B. Implementation of Hybrid Interface

For hybrid simulation, additional numerical treatments need to be implemented on each individual code. They include hybrid schemes, information exchange, and global change of data structure.

TABLE I. Implementation of Hybrid Interface on CFD and MD Codes. Both codes are equipped with the file-based information exchange routine, to update the hybrid boundary condition. CFD code experiences the global change of its data structure to store the information of whole fluid system. MD code adopts hybrid equations to impose the macroscopic information on microscopic domain and to ensure numerical stability.

	CFD	MD
Global Change	Overset Data Structure (*optional)*	-
External Force	-	External Force Equation (eqn. **??**)
CFDtoMD	File Interface: Sender	File Interface: Receiver Constrained MD Equation (eqn. **??**)
MDtoCFD	File Interface: Receiver	File Interface: Sender

Major changes on standalone CFD and MD codes are summarized in table **??**. With regard to the CFD code, it experiences the global change of data structure to be the same as that of overset mesh technique [**?**]. That is, CFD code stores the whole fluid domain with its geometric information and treat the pure MD region in Fig. **??** as 'Hole' in the terminology of overset technique. Likewise, MDtoCFD and CFDtoMD boundary cells are declared as 'Fringe' and 'Donor' cells, respectively. This change is "optional" but we strongly encourage because the implementation of overset technique will enable easy change of the hybrid condition (position and depth of hybrid layers) without mesh regeneration .

Information exchange between continuum and discrete particle descriptions takes place through the file interface. The CFD code has one file sending and one receiving interface, the former sending conserved properties on 'Donor' cells to MD site and the latter imposing the MD solution as boundary conditions on 'Fringe' cells. All exchanged data are written in MD unit: thus, CFD code is equipped with velocity unit conversion function and equation of state which changes the pressure solution from CFD site to equivalent density property in MD domain.

Besides the data exchange interface with CFD code, additional equations of motion need to be employed on MD code to accurately describe the influence of macroscopic flow variation on particle domain. First, the external force should be imposed to prevent leaving particles from the control domain and the force is applied its perpendicular relative position of uppermost MD layer, as was seen in Fig. **??**.

$$F_{ext,i} = -p_a \sigma \frac{y_i - Y_{n-1}}{1 - (y_i - Y_{n-1})/(Y_n - Y_{n-1})} \tag{6}$$

where p_a denote the average pressure in the MD region, $Y_n - Y_{n-1}$ is the thickness of the uppermost layer which is applied the force and F_{ext} is the external force acting on i_{th} particle located on position y_i.

Next, on CFDtoMD layer, the macroscopic flow properties at specific time shall be introduced to lead the motion of multiple particles in that layer. To satisfy mass conservation, a certain number of particles are inserted into or removed from this layer according to the mass flux by CFD solution,

$$n = -A\rho u_y \Delta t/m \tag{7}$$

where A is the horizontal area, u_y is the vertical velocity component by CFD solution and Δt is sampling interval.

Next, a very complicated numerical intervention is required to maintain momentum conservation. The average velocities of particles in J_{th} cell is equal to the velocity u_J in continuum cell.

$$u_J(t) = \frac{1}{N_J} \sum_i v_i \tag{8}$$

where v_i is velocity of i_{th} particle and N_J is the number of particles in the cell. With taking Lagrangian derivative of eq. **??**,

$$\frac{Du_J(t)}{Dt} = \sum_i \frac{\ddot{x_i}}{N_J} \tag{9}$$

The Classical MD equation of motion can be generalized to obtain constraint by adopting the fluctuation in acceleration of each particles, ζ_i

$$\frac{F_i}{m_i} = \ddot{x_i}(t) = \frac{Du_J(t)}{Dt} + \zeta_i = \frac{\sum_i F_i(t)}{\sum_i m_i} + \zeta_i \tag{10}$$

where F_i is the force on i_{th} particle based on the interactions between particles, m_i is mass of each atom and eq. **??** satisfies,

$$\sum_i \zeta_i m_i = 0 \tag{11}$$

The constrained particle dynamics with conventional equation of motion can be written as:

$$\ddot{x_i}(t) = \frac{F_i}{m_i} - \frac{\sum_i F_i(t)}{\sum_i m_i} - \frac{1}{\Delta t_{MD}} \{ \frac{\sum_i m_i \dot{x_i}}{\sum_i m_i} - u_J(t + \Delta t_{MD}) \} \tag{12}$$

The continuum velocity and the mean microscopic velocity from MD over control domain provide the synchronization of the mass and momentum consistent via eq. **??**.

In MDtoCFD layer, particles' properties are averaged over prescribed sampling duration and written on file

interface at every sampling interval. This suffices to impose microscopic profile on macroscopic domain.

The last thing to implement is the scheduling of data exchange routine in time domain, to synchronize both solutions at the same physical time. By default, two solvers open their send and receive channels on every sampling time step and independently advances to the next sampling time step, which is the design of synchronized coupling strategy in Fig. **??**-a). However, the time-lagging phenomenon of this strategy requires the modification for unsteady flow simulation.

A new approach named 'prediction-correction strategy' is depicted in Fig. **??**. After the data exchange at t, MD advances to the next communication point by extrapolating two previous boundary conditions at $t - \Delta t$ and t. On the other hand, CFD code loads the previous solution at $t - \Delta t_S/2$ after communicating with MD code at t and evolves to $t + \Delta t$ by imposing extrapolated boundary condition from data at $t - \Delta t - \Delta t_S/2$ and $t - \Delta t_S/2$, which makes the difference from the original strategy. In this approach, time interval between $t - \Delta t_S/2$ and t gets to be solved twice: first in prediction step and second in correction step. Clear benefit of current approach is that both solvers are provided with the "exact" boundary condition once in every sampling interval, compared to the original formulation in which boundary condition had to be extrapolated "everytime" or the quasi-steady assumption had to be satisfied. As the exact boundary condition is provided on every sampling interval, sampling duration can be increased as long as sampling interval, which will reduce statistical error. Furthermore, with longer prediction setting as seen at Fig. **??**-b) and c), more accurate boundary condition can be imposed by simulating over longer prediction time.

Though the current numerical approach looks a possible way to eliminate the time-lagging phenomenon of existing strategy, this approach can stand on two assumptions: (i) computational cost on CFD is quite smaller than that of MD, and (ii) driving force which determine the flow characteristics is generated on CFD side. Without the condition (i), additional computational overhead for prediction process will harm the simulation performance. If (ii) is not satisfied, the accuracy of the predicted solution is not guaranteed.

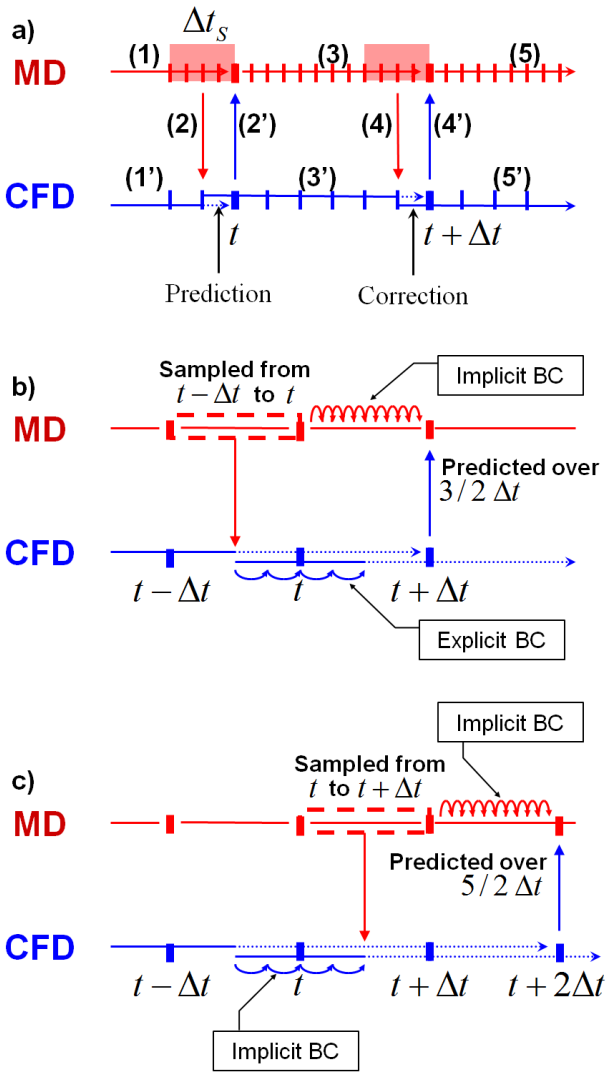

Fig. 4. A Prediction-Correction Approach; CFD evolves over longer time than the communication point and supports MD domain with predictive boundary condition. a) Default Formulation; CFD solver evolves $\Delta t_S/2$ time longer, which satisfies the explicit boundary condition of CFD and MD site at communication point. b) Implicit MD Boundary Condition; On exchange point, CFD provides predicted solutions until the next data exchange point by a predictive simulation of $3/2\Delta t$ time. c) Implicit CFD and MD Boundary Condition; In principle, both solvers are expected to get implicit boundary conditions with the predictive simulation of $5/2\Delta t$ time.

IV. Construction of a Coupled Multi-physics Simulation Framework

***Jeff: This Section is moved from introduction: needs rewriting to match this section. Conceptually, the adoption of a Pilot-Job concept can be an answer to resolve above co-scheduling and load balancing issues of running a coupled multi-task simulation. The Pilot-Job is just a container task where a number of sub-tasks can run in a pre-defined schedule with the specified number of processors whether or not they are coupled. We basically devise this solution to overcome the concurrent scheduling requirement/constraints of coupled jobs; interestingly, the dynamic resource allocation capabilities of the Pilot-Job prove useful for the effective load-balancing of two in-

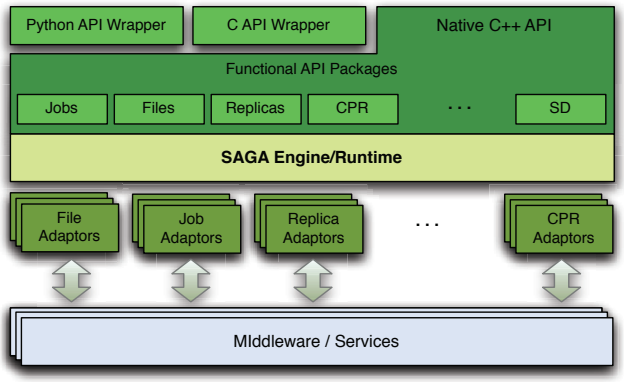

Fig. 5. Layered schematic of the different components of the SAGA landscape. At the topmost level is the simple integrated API which provides the basic functionality for distributed computing. Our BigJob abstraction is built upon this SAGA layer using Python API wrapper

dependent but concurrently running codes. In contrast, if simulations were submitted as independent jobs, changing resource (CPU) allocation to address these changes is challenging – as the change in resource assigned to one is correlated with a change in resource assigned to the other component.

A. SAGA and SAGA-based Frameworks - An Efficient Runtime Environment for Coupled Multi-component Computations

***Jeff: Introduction to SAGA: All details and amount dedicated to Prof. Jha :)

The Simple API for Grid Applications (SAGA) is an API standardization effort within the Open Grid Forum (OGF) [**?**], an international standards development body concerned primarily with standards for distributed computing. SAGA provides a simple, POSIX-style API to the most common Grid functions at a sufficiently high-level of abstraction so as to be independent of the diverse and dynamic Grid environments. The SAGA specification defines interfaces for the most common Grid-programming functions grouped as a set of functional packages (Fig. **??**). Some key packages are:

- File package - provides methods for accessing local and remote filesystems, browsing directories, moving, copying, and deleting files, setting access permissions, as well as zero-copy reading and writing
- Job package - provides methods for describing, submitting, monitoring, and controlling local and remote jobs. Many parts of this package were derived from the largely adopted DRMAA
- Stream package - provides methods for authenticated local and remote socket connections with hooks to support authorization and encryption schemes.
- Other Packages, such as the RPC (remote procedure call) and Replica package

Fig. **??** shows the structure of BigJob and its operation flow. When a BigJob is submitted to the remote resource, the application manager monitors the status of this pilot-job through the advert service. When resources are allocated to the BigJob, the application manager allots the obtained resources to its sub-jobs and a coupled simulation starts under the control of a multi-physics agent in the remote resource. Advert service keeps on getting the status of a pilot-job from the queuing system and the status of sub-jobs from multi-physics agent and also delivers this information to the application manager by a push-pull mechanism. The application manager watches the status of sub-jobs and decides the next event when the coupled simulation is finished. When one default BigJob is launched, sub-jobs keeps running until final solution is achieved and the manager quits the Pilot-Job at that time. In case multiple BigJobs are submitted for the same simulation or if a load balancing function is included, sub-jobs experience several restarts from their checkpointing data, reflecting changed processor allocation to each application. In the former case, resource allocation to each sub-job follows a pre-defined map according to the number of BigJobs allotted to the simulation; in the latter case, resource allocation to each sub-job becomes dynamic according to its performance, as discussed in the next section.

B. Load Balancing for Coupled Multi-Physics Simulation

***Jeff: I know.. I'll be shrinking this subsection.. When a rabbit gets a horn on his head :) The flexibility to re-distribute resources (processors) to the individual task does not imply efficient utilization. This is the responsibility of a load-balancer (LB). We will discuss the implementation and algorithm of this LB [**?**]; it is important to mention that the LB functions in the context of the SAGA-BigJob framework. Each application's load is determined by its elapsed time to run the evolution loop. Here, time for initialization or inter-domain data exchange are excluded from the counting, because they are one-time events or irrelevant to application's performance. The efficient functioning of the LB is predicated on application codes being able to restart from their checkpointing data effectively. Also, application codes should be equipped with generalized domain partitioning routine to run a simulation with any number of processors, without harming their parallel efficiency a lot. If above conditions are satisfied, it makes sense to load the LB within the pilot-job,

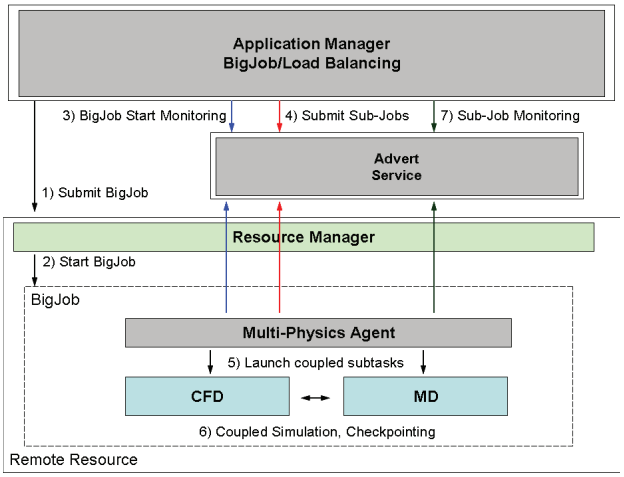

Fig. 6. Architecture of the Controller/Manager and Control Flow: Application manager is responsible for job management including BigJob and sub-job submission, their status monitoring functions. We implement a load balancing module, and migration service based on job information. Application agent system resides on each HPC resource and conducts job information gathering and also communicates with the application manager via the advert service

to implement dynamic resource allocation between tasks. Conceptually, the load-balancing algorithm assigns more processors to a sub-task with greater runtime, until the two codes take the same wall-clock time between points when they communicate. Interestingly, our approach is very simple and the algorithm is independent of applications upon the predications of, (1) each application code follows the ideal parallel efficiency. (2) all processors in one node are assigned to one single task.

Let the computation time (between exchanges) of the two sub-jobs be t_{CFD} and t_{MD}, and the number of processors assigned to each domain be PE_{CFD} and PE_{MD}, respectively. Subscripts I and F denotes initial and final states. Based on assumption (1), total problem size of each application remains the same after resource re-allocation,

$$\begin{aligned} W_{CFD} &= PE_{CFD,I} \times t_{CFD,I} = PE_{CFD,F} \times t_{CFD,F} \\ W_{MD} &= PE_{MD,I} \times t_{MD,I} = PE_{MD,F} \times t_{MD,F} \end{aligned} \quad (13)$$

In spite of the re-allocation, the total number of processors utilized remains the same:

$$PE_{TOT} = PE_{CFD,I} + PE_{MD,I} = PE_{CFD,F} + PE_{MD,F} \quad (14)$$

Our objective is to reduce the computation time of a sub-job to the point until the two application components show the same computation between the exchange points, i.e., $t_{CFD,F} = t_{MD,F}$. From Equation **??** and Equation **??** an optimal number of processors distributed for the CFD subtask would be:

$$PE_{CFD,F} = \frac{W_{CFD}}{(W_{CFD} + W_{MD})} \times PE_{TOT} \quad (15)$$

The MD simulation (sub-job) will follow a similar expression. The optimal processor distribution from above equation will return a non-integer value. Also, under the second assumption (which is the policy of many supercomputing centers), any load-balancing determined as above, will proceed in discrete values expressed as the multiples of the number of CPU cores in a node. We choose the nearest discrete number to our load balancing as the optimal number of processor on each application.

C. Implementation of an Execution Framework to Support Multi-physics Applications

***Jeff: Features of an application manager: Mention that it is written in PYTHON, load balancing function is implemented inside, 'codes experience several launch/re-launches (which we call as evolution loop) to get assigned with changed number of processors by the result of load balancing', etc. 1 paragraph. Also include how LB is searching for its final solution: first with pure computation performance, next with small variation of total CPUs, next with more variation, then get the final solution. Also, in case of internal disturbance, it would reference former values. It stores 4 last time data with different CPU numbers.

(More context) The application manager along with its load balancing functionality is implemented as follows.

***Jeff: follows will be rewritten to one paragraph No changes in the application codes are required in order to utilize the BigJob capability; however, in order to utilize load-balancing features the application codes need to be equipped with the ability to carry out application-level checkpointing. The application codes in addition to having application-level checkpointing need generalized domain partitioning method to work with a range of processor counts.

The use of a load balancing function requires one more change on application codes. As load balancer refers to the performance data from application codes, application codes need to return their actual runtime, which excludes idle waiting on inter-domain information exchange. This can be implemented by checking elapsed time on inter-domain data exchange and subtracting it from total iteration time. The use of a load balancing function requires one more change to application codes. As load balancer refers to the performance data from application codes, application codes need to return their actual runtime, which excludes idle waiting on inter-domain information exchange. This can be implemented by checking elapsed

time on inter-domain data exchange and subtracting it from total iteration time.

The generation of an application manager and the changes in application codes raise the possible simulation scenarios as given in Fig. **??**. The first (leftmost) shows the time evolution of a coupled simulation executing after using a conventional job submission (which we define to be scenario S0), and the other using a BigJob. For S0, individual tasks with resource requirements of PE_{CFD} and PE_{MD} respectively, are independently submitted to the conventional queuing system and job scheduler recognizes these coupled tasks as two distinct jobs. Thus, they are start at different times on average, except when coincidentally resources for both are available. In this case, both tasks wait on the queue when no job is allocated, the first allocated job idles to perform data exchange with its counterpart; the actual simulation executes only after both jobs are running/allocated. On the other hand, for scenario S1, a BigJob of size $PE_{CFD} + PE_{MD}$ is submitted to the queue, and coupled simulation directly starts when the resource is assigned to this BigJob. Because of co-scheduling of sub-jobs, a BigJob is free from long inactive mode which is frequent in conventional job submission, while total runtime is the same if the resource distribution to sub-jobs is identical. However, eliminating inactive mode in of itself does not guarantee a reduction in the total runtime, because a larger allocation may result in a greater queue waiting time than two simulations requesting smaller number of processors each (but the total being the same). The same situation can arise for the load-balanced case with one BigJob ($S1_{LB}$). From the comparison between S1 and S0, we can estimate the performance gain by concurrent start of distinct coupled codes; $S1_{LB}$ solution compared to other scenarios will demonstrate the benefit of a load balancing function on coupled simulation.

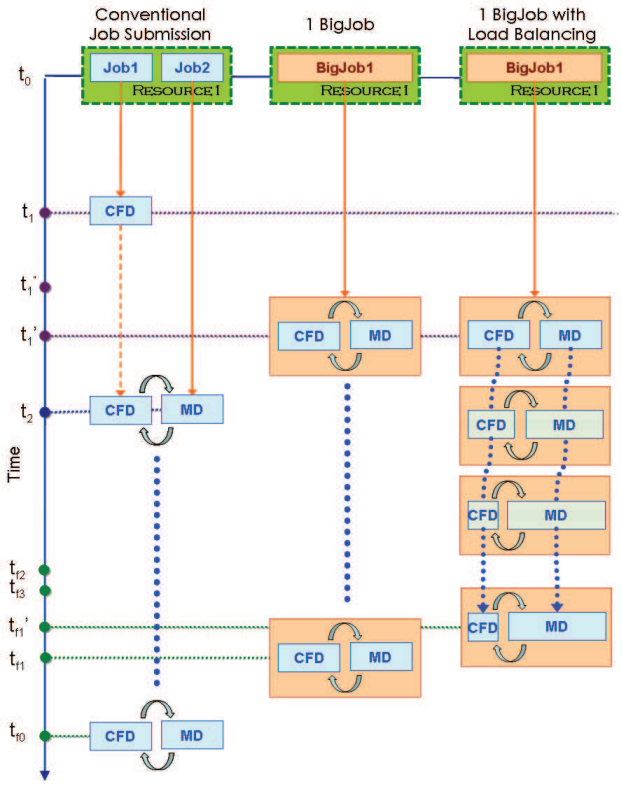

Fig. 7. Comparison of dependencies encountered for coupled simulations submitted as conventional jobs, to the scenario when using Pilot-Jobs. Here we use only 1 BigJob (S1). The conventional mode of submission experiences three phases based upon their queue state: (i) All jobs are waiting: $(t_1 - t_0)$; (ii) Inactive mode (where one job is waiting for another: $t_2 - t_1$), and (iii) Active mode (running a coupled simulation: $t_f - t_2$). The Inactive stage, disappears when a coupled simulation runs within a BigJob, as an allocated BigJob distributes its resource to both sub-jobs.

V. Multi-scale Flow Simulation and Its Performance

A. Nano-scale Couette Flow Simulations

The first problem is the Couette flow simulation, which have been in wide use for the validation of a hybrid CFD-MD solver. [**?**],[**?**] We start from the validation case, which has 52σ distance between two parallel plates and upper wall velocity is σ/τ. We assume liquid argon particles are filled in the domain and both walls have artificial properties which is the same as those of liquid argon. Characteristic length of liquid argon is $\sigma = 3.405\times10^{-10}$ and time scale is $\tau = 2.2\times10^{-12}$. Density is $0.81m\sigma^{-3}$, which means 0.81 atoms are included in the characteristic volume.

As we have argued earlier, MD solution inherently describes the fluctuating pattern of particles, which becomes a noise in CFD solution. Thus, the first thing to do for coupled simulation is to determine coupling conditions including layer size (layer width and height), sampling duration, sampling interval and MD timescale, which defines the scale and pattern of noise. For continuum approach, averaging more particles on bigger layers for a long time would be preferred to eliminate the systematic noise of molecular vibration. However, in view of MD, smaller layer is preferred to accurately apply the velocity gradient from CFD solution. Also, shorter sampling interval who updates boundary condition more frequently is required for the time-accurate solution. Thus, optimal coupling condition will be to set the smallest sampling interval (it is noteworthy that our new temporal coupling approach enables the sampling duration to be as long as the sampling interval.) whose statistical error is within the range of acceptable error.

TABLE II. Implementation of Hybrid Interface on CFD and MD Codes. Both codes are equipped with the file-based information exchange routine, to update the hybrid boundary condition. CFD code experiences the global change of its data structure to store the information of whole fluid system. MD code adopts hybrid equations to impose the macroscopic information on microscopic domain and to ensure numerical stability. The first one with x=35 σ

	1 tau	2 tau	4 tau	8 tau	16 tau	32 tau	64 tau
0.1 σ	35.184	27.888	22.167	18.043	14.819	12.820	12.835
0.2 σ	29.890	24.546	20.020	17.499	14.836	14.252	14.246
0.4 σ	26.610	22.440	18.793	17.053	14.464	13.238	13.234
0.8 σ	23.458	20.640	17.489	15.782	13.879	12.475	12.463
1.6 σ	18.902	17.514	15.837	14.448	12.813	11.459	11.035
3.2 σ	14.055	13.552	12.942	12.322	11.225	10.027	9.770
6.4 σ	10.040	9.873	9.640	9.270	8.937	8.141	7.961
12.8 σ	6.701	6.649	6.579	6.454	6.315	6.007	5.282

	1 tau	2 tau	4 tau	8 tau	16 tau	32 tau	64 tau
0.1 sigma	24.969	20.608	15.674	12.683	9.161	7.165	6.197
0.2 sigma	20.826	17.397	13.052	10.833	7.867	6.622	5.567
0.4 sigma	18.000	15.210	12.028	10.261	7.412	6.063	5.810
0.8 sigma	15.209	13.108	11.059	9.478	7.358	5.679	5.556
1.6 sigma	12.034	10.998	9.809	8.477	7.157	5.751	5.522
3.2 sigma	8.802	8.453	7.923	7.052	6.762	5.634	5.486
6.4 sigma	6.163	5.952	5.795	5.538	5.355	5.019	4.928
12.8 sigma	4.760	4.712	4.691	4.578	4.385	4.120	4.120

	1 tau	2 tau	4 tau	8 tau	16 tau	32 tau	64 tau
0.1 sigma	17.929	14.024	10.980	9.595	7.386	6.409	5.744
0.2 sigma	14.887	12.077	9.924	8.638	7.167	6.310	5.736
0.4 sigma	12.823	10.849	9.326	8.081	7.202	6.466	5.771
0.8 sigma	11.546	10.226	9.223	8.033	7.402	6.307	5.852
1.6 sigma	9.565	8.970	8.367	7.648	6.997	6.247	5.983
3.2 sigma	7.705	7.558	7.291	6.962	6.400	5.966	6.017
6.4 sigma	5.717	5.657	5.574	5.338	5.070	4.999	4.958
12.8 sigma	4.293	4.271	4.219	4.137	4.079	3.940	3.906

TABLE III. Implementation of Hybrid Interface on CFD and MD Codes. Both codes are equipped with the file-based information exchange routine, to update the hybrid boundary condition. CFD code experiences the global change of its data structure to store the information of whole fluid system. MD code adopts hybrid equations to impose the macroscopic information on microscopic domain and to ensure numerical stability.

	10 τ	20 τ	40 τ	80 τ	160 τ	320 τ	640 τ
1 σ	13.188	11.573	9.725	8.640	6.604	3.666	2.625
2 σ	11.594	10.522	9.329	8.285	6.690	3.715	3.129
4 σ	9.964	9.436	8.488	7.619	6.531	4.006	3.582
8 σ	7.887	7.571	6.948	6.439	5.292	3.681	3.492
16 σ	5.725	5.638	5.361	5.206	4.420	3.143	2.712
32 σ	3.910	3.854	3.752	3.748	3.599	3.186	2.247
64 σ	2.470	2.457	2.421	2.420	2.303	2.277	1.877
128 σ	1.640	1.631	1.622	1.613	1.613	1.595	1.557

Our idea of numerically detecting the strength of statistical noise is to solve the stationary flow of the full fluid domain by pure MD method. According to statistical error analyses on periodic MD system by Hadjiconstantinou et al [**?**] and Delgado-Buscalioni and Coveney [**?**], the statistical error is inversely proportional to ${u_{max}}^2$. This implies that the statistical error in stationary flow is going to be the upper bound of the fluctuation strength as long as this relation remains valid on wall-bounded flows. Computational cost on solving pure MD system is fairly acceptable as we can start gathering the data right after the minimization process is done.

Table **??** shows the averaged velocity on different layer size and sampling duration.

Figure **??** shows the fluctuation of averaged velocity in pure MD simulations of stationary flow. Domain size is the same as our validation problem and this domain is split to 6, 12, 24, 48 layers in Y-direction to compute the flow velocity. As the flow is stationary, all averaged velocities should be zero regardless of the number of layers. However, the L2 norm of averaged velocity tends to increase with more layers (from 6 to 48), meaning that bigger layer size is required to reduce the noise in hybrid boundary. The strength of fluctuation converges to $0.01\sigma/\tau$ as we increase the sampling interval, which implies that the fluctuation of ± 0.01 in hybrid boundary is unavoidable. The fluctuation did not decrease with smaller MD timescale and this concludes us that this fluctuation and its scale is physically natural phenomenon. From this test, we decide the coupling condition as $\Delta t_{MD} = 5 \times 10^3$ with each layer size to be 2σ (which is nearly equivalent to having 24 layer), sampling duration of 10τ and sampling interval is set up the double of sampling duration.

This simple and direct approach can be still valid on micro-scaled domains by reducing the domain size in periodic direction. Again from statistical error analyses [**?**],[**?**], statistical error is inversely proportional to the number of particles. increasing the system domain in periodic direction will reduce the statistical error proportionally. Also, for more accurate error computation

Multiple datasets with different layer sizes and sampling time are gathered by a single simulation run using *fix-ave-spatial function on LAMMPS package. Initial data up to 96 τ are dropped off to have enough time for minimization process and data at the central layer are averaged over 512 τ for the reliability.*

Resulting CFD and MD computational domains are depicted in Fig. **??**. *In CFD mesh, the cell size in Y-direction is 2σ. Pure MD region is specified to be 10σ, which was observed to be sufficient to prevent strong fluctuation between fluid particles and wall materials from directly transported to CFD domains by iterative testruns. Two layer boundary zones from particle to continuum domain is placed ahead of pure MD region, from 10 to 14σ. Continuum to particle boundary is positioned from 18 to*

22σ and external force region is place at the top of hybrid region, from 24 to 26σ. MD domain has sufficiently large length in the principal flow direction, to include enough number of particles in the hybrid data average region.

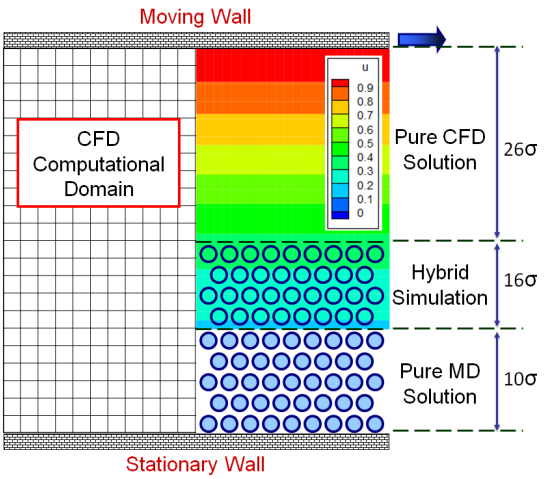

Fig. 8. Computational Domain of Couette Flow Simulation; The height of the fluid domain is 52σ ($\approx 177\mathring{A}$). CFD mesh size is 201×27 and CFD cells at the pure MD region are treated as holes. MD domain size is about 210σ in the X-direction and around 30σ in the Y-direction, including the bottom wall. Periodic boundary condition is applied on the principal flow direction.

Unsteady Couette flow profile by CFD, MD and hybrid simulations are presented in Fig. **??**. *Pure CFD produces identically the same result as analytic solution and MD simulations also describes the same flow physics. In MD simulation, fluctuating pattern is observed during the evolution. This causes the slight variation in hybrid solution, along with the time-lagging in current algorithm. After convergence, the solution follows the same flow pattern as steady profile, which proves that the hybrid approach can accurately analyze the steady flow profile in micro-scaled systems.*

The same domain with coupling condition is applied to solve the Stokes boundary layer problem, which is purely unsteady problem. In this case, the upper wall boundary condition changes from the fixed velocity to oscillatory wall, which is $u_{wall}(t) = (\sigma/\tau)\times sin(2\pi t/T)$. *Period* T *is set* 200τ. *Velocity in the hybrid region becomes far slower than the Couette flow profile, so the influence of noise from MD side is concerned to be more critical in the current flow simulation.*

Figure **??** *shows the oscillatory velocity profile by pure CFD and hybrid simulations. Unsteady profile in both simulations are globally the same, proving that the current approach can be directly applied to unsteady problems. In detail, fluctuation in MD domain is not so strong in hybrid region. Though minor time-lagging profile is seen from*

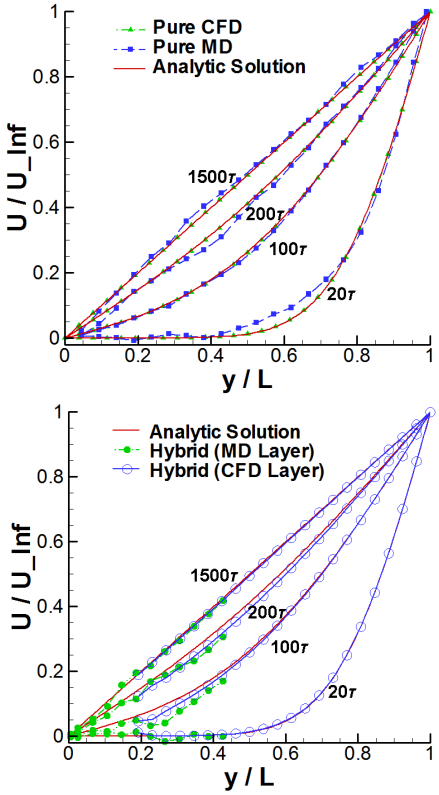

Fig. 9. Unsteady Couette Flow Profile; (Left) Pure CFD solution is exactly the same as analytic solution. MD solution shows the same flow pattern as analytic solution, though some fluctuation is observed. This verifies that CFD and MD represents the same flow physics. (Right) The steady result by hybrid approach produces the same numerical result as analytic solution, though the time-lagging in the hybrid boundary is observed during the evolution.

MD solution, this does not change the global flow profile dominantly because the driving force in this simulation is the oscillating velocity from CFD domain.

B. Numerical Simulation of a Physically Unsteady Problem: Stokes Boundary Layer Problem

****Jeff: Additional context after real-scale simulation*

C. Performance Analysis of a Multi-physics Simulation Framework

****Jeff: Queue Wait Time Analysis: Preliminary Test. Waiting Time with Different PX Requests, Wall Time Limit (Mention that BQP is not perfect, especially measuring inactive waiting time of TWO CONVENTIONAL JOBS)*

The benefit of a BigJob as a way of reducing the waiting time on the local queueing system can be

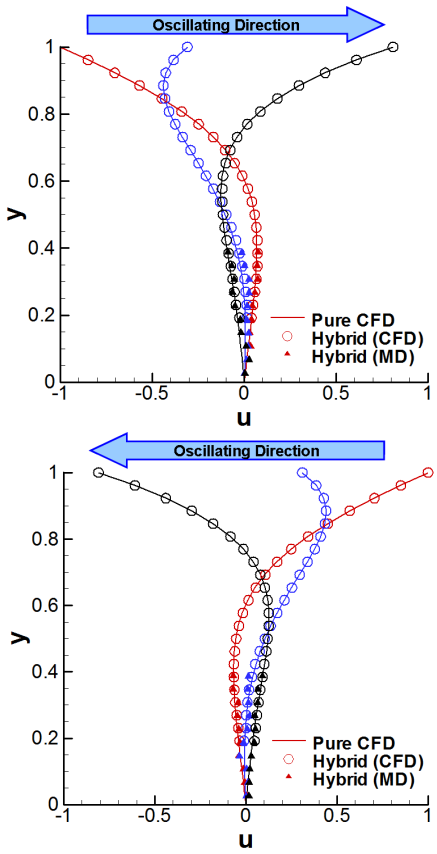

Fig. 10. Unsteady Flow Profile of Stokes Boundary Layer Problem; Solution by pure CFD simulation is denoted by a solid line and solution by hybrid simulation is presented by symbols. Hybrid flow profile globally matches well with CFD solution, though slight time-lagging in the MD side is observed. This evaluates that current hybrid approach can be applied to purely unsteady problem.

discussed from our preliminary test of the relationship between the requested number of processors and the waiting time.

****Jeff: Already tested waiting time on Ranger: data table and expression will be replaced.*

We designed experiments to determine if running larger and/or longer simulations effects the actual waiting time on the queue. We performed our experiments on machines spanning two orders of magnitude in size (and peak performance)– from approximately 500 px to 65000 px, and for a range of load-factors. We submitted jobs of different sizes, requiring different wall-time limits. Each time we submitted a job, we gathered the load-factor and the actual waiting time on the queue. Many factors effect the waiting times, arguably the most important of them are the load-factor at the instant of submission, requested wall-time limit and also the number of processors requested.

TABLE IV. Effect of various configurations on waiting time. The tables show the queue waiting times on Ranger and QueenBee with a changing number of processors and different requested resource wall-time limits. Analyzing the actual waiting time as a function of the number of processors at different wall-time limits, it can be said that better more often than not, the waiting time decrease as the requested number of processors increases. The relationship between the waiting time and wall-time limit is harder to quantify. However, obtained numbers provide a good case study for showing the variance of actual queue waiting times.

Number of processors	Requested wall-time at 92±6% load (Ranger)				
	2hr	3hr	4hr	6hr	12hr
	Waiting time on the queue [sec]				
16	9989	15984	39151	65	66
32	15371	4106	11376	54	55
48	13264	4392	37780	43	44
64	9944	1975	39855	31	32

Number of processors	Requested wall-time at 95±4% load (Queenbee)				
	2hr	3hr	4hr	6hr	12hr
	Waiting time on the queue [sec]				
16	14339	3578	39113	6	940
32	14312	3550	39238	5	6344
48	21555	3517	39207	4	6353
64	21541	3489	39179	3	6329

Two other factors that effect this are the backfilling that occurs when some other unrelated job finishes and the changes in the priority of the test job when a particular higher priority job joins the queue; however, unlike the first three factors, the last two are somewhat random and it is difficult to systematically account for them. Also, the internal queueing policy of supercomputing centers may effect the waiting time on the queue based on their queueing priority including credential, fairshare, resource and service priority. As can be seen from Table **??**, *the waiting time tends to decrease as the number of processors requested in a job increases.*

Results for machines with more than 5000 px, Ranger and Queenbee are presented in Table **??**, *and as can be seen, jobs with larger processor counts have typically lower wait times. The observation holds true for a range of values of requested wall-times over a range of "high" load-factors.*

It is however, difficult to discern a relationship between waiting time with the requested wall-time limit of jobs; from experience there is greater variation in backfilling probability with varying wall-time requests. However, as we are able to establish that a single large job on average has to wait less at the queue than a smaller job does, most definitely the maximum wait time of two small jobs will be even greater. In other words the sum total of the waiting time (of the first-to-run job) and the inactive time (defined

TABLE V. Waiting and inactive time for conventional job submissions, and a single BigJob submission. All measured times are in seconds and expressed as 'mean±SD'. In both cases, conventional job is submitted to use 2×32 px and a BigJob requests 64 px on small and less crowded LONI clusters. Conventional job submission mode showed faster time-to-start (i.e., the sum of waiting time and inactive mode) on small problem size with 6 hr of wall-time limit, while a BigJob gets allocated faster with longer wall-time limit. Conventional job submission showed 3 failures during the test due to the timeover of wall-time limit in the first-started job.

	Small simulation with 6 hr wall-time limit		
	Waiting time	Inactive mode	Failed runs
Conventional	12318±15649	7407±11375	2
1 BigJob	29452±31946	0	0

	Large simulation with 24 hr wall-time limit		
	Waiting time	Inactive mode	Failed runs
Conventional	83102±77134	47488±55647	1
1 BigJob	76645±55474	0	0

as the difference between the waiting time of the two jobs) will be larger than the wait time of a single large job.

****Jeff: Queue Wait Time Analysis: Waiting time of a BigJob and two conventional jobs. Emphasize the inactive time as well - it makes the conventional job submission to have longer wall time limit and it will even reduce the possibility of getting the backfilling capability. Include Table 2 here.*

A BigJob submission shows the saving of waiting time on the queue, not only the bigger job size has the higher priority in the queueing system but also one BigJob submission eliminates the inactive idling time of first allocated job in conventional job submission.

****Jeff: More test conducted on Ranger. Table needs replacement.*

Controlling runtime in these two scenarios, i.e., we take a BigJob of size 2X and 2 conventional jobs of size X each, we compare the waiting time of a 64 px BigJob (with wall-clock limits of 2, 3, and 4hrs) which is smaller than the wait for a 32 px conventional job for the same values of wall-clock limits. As the individual simulations are assigned the same number of processors, the runtime performance will be similar in the two scenarios, and thus the total time-to-solution will determined by the wait-times on queues – which we show statistically to be lower for the larger BigJob submission mode.

Results for smaller systems (≈ 500 px) are presented in a different fashion in Table **??**. *In addition to the fact that a 64 px BigJob is now around 16% of the machine resources, the load-factors of the smaller machines fluctuated much more than those of the larger machines. Consequently the data obtained for smaller resources is far more noisier and than the "clean" data for the larger machines – Ranger and QueenBee. As can be seen from Table* **??** *although the waiting time for the first-to-start job was smaller in the conventional job submission mode than the BigJob (S1), the second job (convention job) had a large subsequent wait time; thus for conventional job submission mode there is a non-negligible inactive mode (defined as the time difference between queue wait times between the two simulations). There is no inactive mode for BigJob submission as by defintion both simulations begin concurrently. Interestingly, the situation is somewhat worse for the conventional job submission; although the average wait time is lower than the BigJob, there exist 2 (out of 6) cases (for 6hr wall-clock limit), when the wait time of the second job is greater than the wall-clock limit for the first-to-start job. In other words, the second job fails to start in the duration that the first-to-start job is in active/run mode, but can't do anything useful as the second job is still waiting on the queue. This is typically alleviated with co-allocation of resources; however the number of production grid infrastucture that provide co-allocation (and co-scheduling) as a production service is very limited. This leads us to the heart of the advantage of the BigJob submission mode: circumventing the requirements of co-scheduling by providing an application level solution that preserves performance and guarantees non-failure due to large-fluctuations (see data for 24hr wall-time limit) in queue waiting times.*

****Jeff: Runtime Analysis: Comparison of LB with conventional simulation time: different system size (with different ratio between CFD and MD), different number of simulation loop and different interval of simulation loop. Table 3 with 2 graphs - change of CPU allocation to each subjob with iteration time in 2 simulation cases. Mention the problem size and setting a little more detail.*

Runtimes of the coupled simulation with a single BigJob is given on Table **??**. *For both small and large simulations, a default BigJob task takes about 1 percent longer than the conventional test. This is reasonable because a default BigJob has the same processor distribution between sub-jobs as the conventional job, while BigJob has the minor overhead of sub-jobs' status monitoring and communication with advert server. In cases of load-balanced BigJob simulations, there is a significant reduction in the runtime compared to successful conventional jobs – 13% and greater. For larger problem set, a load-balanced BigJob simulation relatively shows higher standard deviation (SD) due to the unexpected instability of a computing resource during one experiment, to be discussed in detail below.*

The validity of a load balancing function can be discussed by the change of processor distribution between

TABLE VI. Results of runtime for S1, $S1_{LB}$ and conventional submission. All measured times are in seconds and expressed as 'mean±SD'. 6 distinct experiments were accomplished for each simulation, all with 64 px. In both cases, S1 shows about 1% overhead due to the communication with advert server. On the other hand, $S1_{LB}$ tests show about 13% runtime save compared to conventional submission.

	Conventional	S1	$S1_{LB}$
Small sim.	757±1.89	764±1.41	661±4.41
Large sim.	39595±119.2	39906±206.3	34350±1302.7

subtasks throughout the simulation. For the result of a small simulation in Fig. **??***, both CFD and MD subtasks are assigned with 32 px initially. After two simulation loops, a load balancer converges to the processor distribution of 12 to 52 px between CFD and MD respectively; this processor distribution remains the same until the simulation completes. Runtime per loop is reduced from 153 sec for the first loop to 107 sec after the convergence. Total computation time is 596.19 sec, which is different from 663 sec counted from BigJob application manager. This time difference implies that the BigJob application manager spends about 13 sec per stage in executing its internal commands including the run of a load balancing function and sub-job re-launch.*

The result of computation time evolution for a large simulation is seen in Fig. **??***. For most experiments, which is given in the left side of Fig.* **??***, a load balancer directly goes to a converged solution of processor distribution, which is 24 to 40 between CFD and MD jobs. On the other, in one experiment, computing nodes assigned to MD simulation seem to have temporarily experienced the internal overhead as shown from the right side of Fig.* **??***. This overhead temporarily increased MD computation time a lot and a load balancer shows the fluctuating pattern of processor distribution in response to this temporary instability. A load balancer goes to a different steady solution after the system settled down, which is the processor distribution of 20 to 44 between two sub-jobs. Compared to the steady solution in stable cases, computation time for one simulation loop increases in this processor distribution increases from 1320 sec to 1380 sec.*

Plots on the right side of Fig. **??** *show a non-monotonically changing resource assignment by the LB, and thus demonstrating how the load balancer can be self-correcting and adapt to changing performance; after increasing the number of processors assigned to the MD, the load-balancer unassigns the additional processors.*

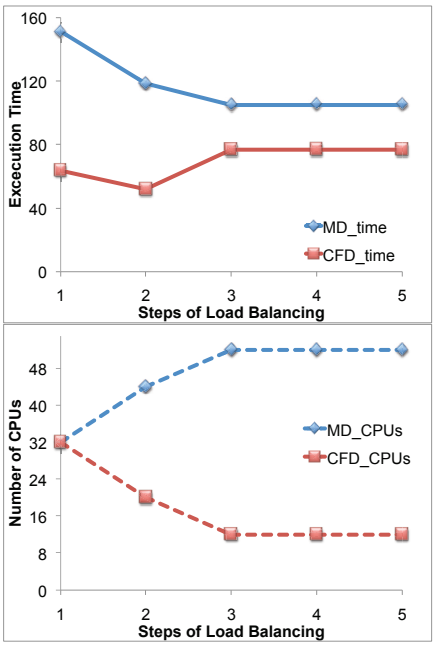

Fig. 11. Change of processor distribution between CFD and MD jobs and resultant computation time in the small simulation. A load balancer starts from the same number of processors assigned to both sub-jobs and detects 20 to 44 px between each sub-job as the optimal solution. The poor scalability of CFD job makes the load balancer to search for the optimal condition once again and the processor assignment finally reaches to a steady solution of 12 to 52 between two sub-jobs. Computation time for every simulation loop reduces from 153 sec to 107 sec after the balancing.

VI. Next Step: Further Refinement

VII. Conclusions

Acknowledgment

This work is part of the Cybertools (http://cybertools.loni.org) project and primarily funded by NSF/LEQSF (2007-10)-CyberRII-01. Important funding for SAGA has been provided by the UK EPSRC grant number GR/D0766171/1 (via OMII-UK). This work has also been made possible thanks to computer resources provided by LONI. We thank Andre Luckow for initial work on BigJob, Lukasz Lacinski for help with SAGA deployment (via HPCOPS NSF-OCI 0710874) and Joao Abecasis for his work on the SAGA Condor adaptors.

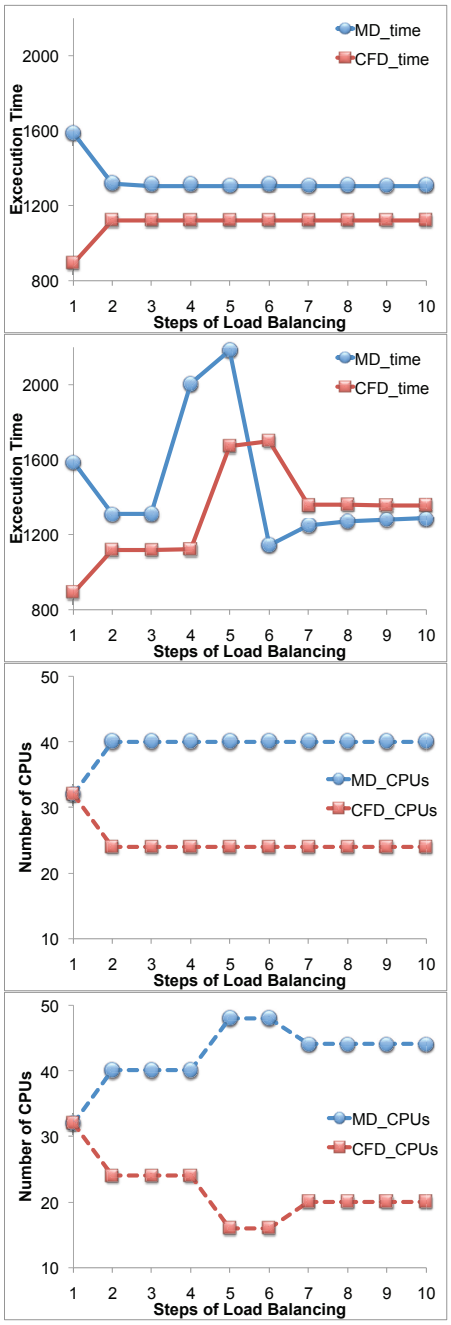

Fig. 12. (Left Column) Change of processor distribution between CFD and MD jobs and resultant computation time in the large simulation. A load balancer directly finds the processor distribution of 24 to 40 between CFD and MD jobs and remains the steady state until it completes after 25 simulation loops. Initial computation time of 1605 sec reduces to 1320 sec after the convergence. (Right Column) Plots showing non-monotonic resource assignment by the LB, and thus demonstrating how the load balancer can be self-correcting and adapt to changing performance; after increasing the number of processors assigned to the MD, the load-balancer unassigns the additional processors.