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 [1],[2],[3],[4], 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. 1. Steady-state solutions of Couette flow by molecular dynamic simulations are presented at different hydrophile 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 evaluates the necessity to consider intermolecular effect on CFD solution procedure.

***Jeff: 1 paragraph; Benefit over MD: efficient computational cost by CFD on global flow solution. Needs to be filled by Nayong.

***Nkim: CPU cost The force calculation to evaluate of the potential energy is the most slowest part of molecular dynamics simulation. The CPU 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 [5].

Despite the above benefit of hybrid CFD-MD approach over pure CFD or pure MD simulations, not many numerical applications have been conducted using this approach. We assume that the hardness to determine the appropriate coupling condition is one possible reason. Even with accurate numerical schemes for solving each domain and 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

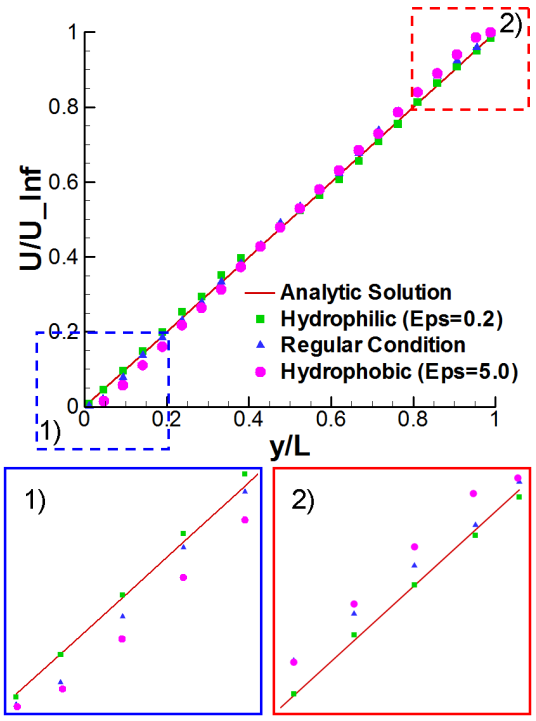

Fig. 1. Steady Couette Flow Solution by Pure Molecular Dynamic Simulation; In normal condition (with the unique potential well depth between flow particle and surface material, denoted by ϵ), the solution by MD is identical to the result of continuum approach. In hydrophobic case (smaller ϵ) the fluid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

no globally acceptable scheme for setting these coupling conditions. Thus, a number of experiments are required to empirically quantify (or mathematize) 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) with very different computational time-scales, 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.

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 independent 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.

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, flow conditions, and flow physics, than any other published works, though each simulation may not be the first trial for specific problem. ***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 utilization of Container-Job/Pilot-Job abstractions to perform coupled

Multi-Physics simulations. We claim that there several distinct advantages that arise as a consequence of using Pilot-Jobs for Coupled Simulations: (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 basic motivation and concept 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 Section 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 (Stokes boundary layer 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.

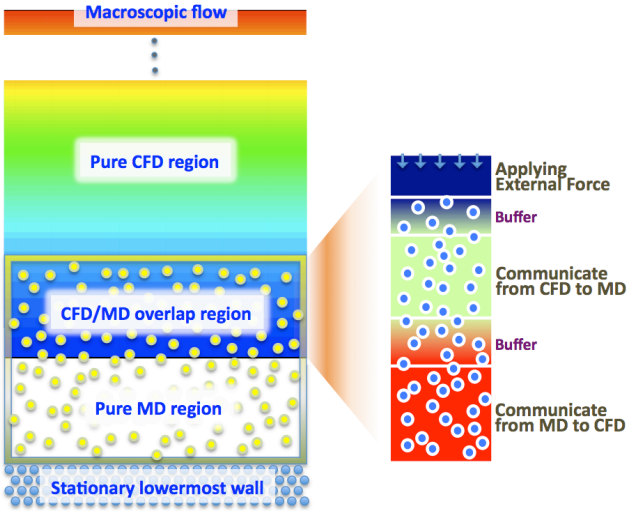

Fig. 2. Schematic Diagram of the Hybrid Domain with Detailed View of Overlapping Zone; Overall continuum/atomistic computational domain including overlap region is shown on left figure. Detailed layer by layer explanation of overlapping region is indicated by right figure.

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

***Jeff: About previous works: 2 paragraphs (More[5] details!)

The study of microfluidic mechanism including from nanoscale fluidic system to continuum one has been recently developed using hybrid CFD-MD method. The hybrid method is taking the advantage of each atomistic and continuum descriptions adopting information exchange based on constrained equation of motion over the overlap region [1],[2],[3],[4]. Modification for mutual flux exchange across the domain boundary has been proposed [6],[7] and more studies about the flux exchange scheme [8],[9], coupling parameters [10] have been accomplished. However, more challenging point of view are still remaining such as extreme physical domain condition, explicit time duration of passing information, the region for taking the statistics of atomistic averaging and others.

***Jeff: Their limitations: 1 paragraph. For example, lack of application problems, unclear setup of coupling conditions, etc.

Context here...

***Jeff: How to Couple: Schematic of CFD and MD zones. 1 paragraph.

(More details) The hybrid CFD-MD approach basically starts from the concept that the inertial force from macroscopic flow domain dominates the system while intermolecular effect cannot be negligible in the low-speed flow region. As given in Fig. 2, CFD approach analyzes the macroscopic flow characteristics with the moderate flow velocity, while MD independently solves low-speed flow region. These two domains overlap in the middle, where CFD and MD exchange their solutions.

***Jeff: Meaning of each layer within overlap region. 1 paragraph.

(Mention the meaning of the buffer and decision of their sizes as well: at least it should be longer than Van der Waals cutoff length. Don't mention 'Van der Waals' here! Change the expression like, 'information from CFD and MD should not conflict each other at the same time') In the CFD-MD hybrid overlap region, computational domain can be decomposed into three interesting regions. Applying external force region where located uppermost boundary of overlap region and in order to the momentum continuity between two approaches. The hybrid schemes have to exchange their information at the overlap region between the different approaches. The way passing information from MD to CFD is more clear since the averaging of particle information applies to a macroscopic state of CFD. While passing information from macroscopic CFD to MD has difficulty and it demands a constrained particle dynamics to achieve consistence between two different multi-scale approaches.

III. Development of A Hybrid CFD-MD Simulation Toolkit

A. Features of Baseline CFD and MD Codes

Two-dimensional unsteady incompressible Navier-Stokes equations are chosen for governing equations to solve the nano-scale flow:

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

where ν is the kinematic viscosity.

In this work, we adopted the pseudo-compressibility method [11] to form a hyperbolic system of equations 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 [12] for the implicit time integration. The inviscid fluxes are upwind-differenced using Osher's flux-difference splitting scheme [13]. For higher-order spatial accuracy, the MUSCL (Monotone Upstream-centered Schemes for Conservation Laws) [14] approach is used on inviscid flux calculation. Viscous fluxes are calculated using the conventional second-order central differencing.

Molecular dynamics (MD) 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. [5],[15],[16] 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 [17].

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. [18]

B. Implementation of Hybrid Schemes

CFD-MD hybrid method has numerical strategy for file-based information exchange between continuum description for macroscopic scale and discrete particle description for microscopic scale. For this information exchange, both codes have the file interface, where each writes the velocity profile of overlapping region, waits for its counterpart's velocity data and receive the data as a boundary condition in the overlap region.

Additional change made on CFD code is the implementation of overlapping scheme, in the same way as Chimera overset mesh [19]. That is, pure MD region is declared as holes and MD boundary layers are treated as fringe cells. Averaged particle velocity over time interval at MDtoCFD region (see Fig. 2) from molecular dynamics domain is stated as fringe boundary condition to the CFD code.

On the other hands, MD code experiences a lot of change to implement hybrid scheme. 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. 2.

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

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.

Communicate MD information to CFD can be achieved uncomplicated since the coarse grained microscopic information in time is imposed to macroscopic resolution of the continuum domain. The reverse procedure, to implement CFD to MD communication, is more difficult and requires artificial intervention to maintain mass and momentum conservation. The average velocities of particles in J_{th} cell is equal to the velocity u_J in continuum cell.

$$u_J(t) = \frac{1}{N_J} \sum_i v_i \quad (7)$$

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. 7,

$$\frac{Du_J(t)}{Dt} = \sum_i \frac{\ddot{x_i}}{N_J} \quad (8)$$

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

$$\frac{F_i}{m_i} = \ddot{x_i}(t) = \frac{Du_J(t)}{Dt} + \zeta_i = \frac{\sum_i F_i(t)}{\sum_i m_i} + \zeta_i \quad (9)$$

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. 10 satisfies,

$$\sum_i \zeta_i m_i = 0 \quad (10)$$

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

$$\ddot{x_i}(t) = \frac{F_i}{m_i} - \frac{\sum_i F_i(t)}{\sum_i m_i} - \frac{1}{\Delta t_{MD}} \{ \frac{\sum_i m_i \dot{x_i}}{\sum_i m_i} - u_J(t + \Delta t_{MD}) \} \quad (11)$$

The continuum velocity and the mean microscopic velocity from MD over control domain provide the synchronization of the mass and momentum consistent via eq. 11.

Figure 3 shows the schematic view of synchronization mechanism between CFD and MD domains. When both solvers approach the data exchange point (denoted by $t - \Delta t$), they exchange the information in the overlapping region and advance to the next exchange point (t). In the data exchange point, CFD sends the velocity profile at that instance. On the other hand, MD sends averaged velocity over a finite time duration ($S_{MD} \times \Delta t$). In MD simulation, spatially averaged velocity at an instance contains very high noise, thus it is inevitable to average particles' velocities over time. Data exchange interval (Δt) is set sufficiently large compared to the averaging duration.

This mechanism gives us a better parallel efficiency compared to conventional sequential coupling mechanisms where one domain advances to the next exchange point and leads its counterpart to approach this point. [20] As they send and receive their flow data at the same point, both codes can independently advance to the next time level without a large computing cost on waiting. So, parallel performance can ideally reach the single code running only if an appropriate load balancing between two domains can be applied. However, we should agree that this approach inherently includes the time lagging in CFD-MD boundary zone, because averaged molecular dynamic velocity over backward time scale is represented as the solution at that instance.

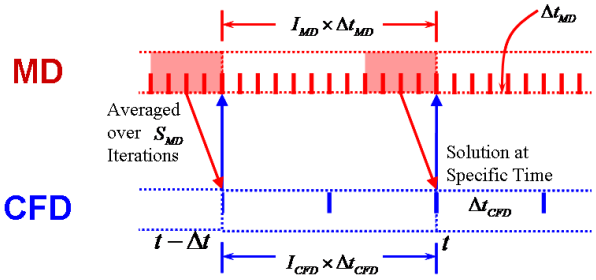

Fig. 3. Time Evolution Mechanism of Application Codes; CFD and MD codes are scheduled to conduct data exchange in the overlapping region at every Δt time. CFD and MD solvers do sub-iterations until approaching the next data exchange point, by I_{CFD} and I_{MD}. CFD solution at $t - \Delta t$ is directly applied as the boundary condition for MD simulation from $t - \Delta t$ to t, while averaged molecular dynamic velocity on $S_{MD} \times \Delta t_{MD}$ durations are imposed as CFD boundary conditions at this time instance.

IV. Construction of a Coupled Multi-physics Simulation Framework

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 :)

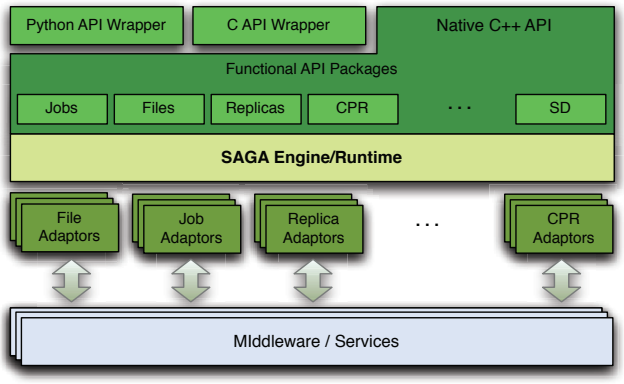

Fig. 4. Layered schematic of the different components of the SAGA landscape. At the topmost level is the simple integrated API which provides the basic functionality for distributed computing. Our BigJob abstraction is built upon this SAGA layer using Python API wrapper

The Simple API for Grid Applications (SAGA) is an API standardization effort within the Open Grid Forum (OGF) [21], 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. 4). 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. 5 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

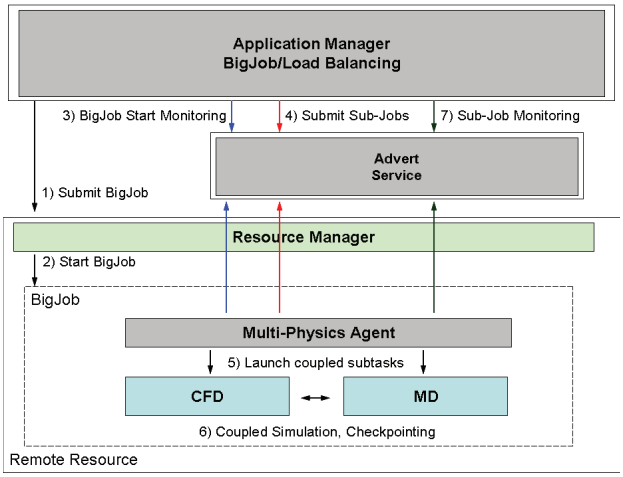

Fig. 5. Architecture of the Controller/Manager and Control Flow: Application manager is responsible for job management including BigJob and sub-job submission, their status monitoring functions. We implement a load balancing module, and migration service based on job information. Application agent system resides on each HPC resource and conducts job information gathering and also communicates with the application manager via the advert service

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 [22]; 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, 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,

$$\begin{aligned} W_{CFD} &= PE_{CFD,I} \times t_{CFD,I} = PE_{CFD,F} \times t_{CFD,F} \\ W_{MD} &= PE_{MD,I} \times t_{MD,I} = PE_{MD,F} \times t_{MD,F} \end{aligned} \quad (12)$$

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

$$PE_{TOT} = PE_{CFD,I} + PE_{MD,I} = PE_{CFD,F} + PE_{MD,F} \quad (13)$$

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 12 and Equation 13 an optimal number of processors distributed for the CFD subtask would be:

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

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. 6. 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.

V. Multi-scale Flow Simulation and Its Performance

A. Numerical Result of a Nano-scale Couette Flow Simulation

The first problem is the Couette flow simulation, which have been in wide use for the validation of a hybrid CFD-MD solver. [2],[3] 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 eariler, 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 decide coupling conditions including layer size, 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, longer sampling duration and the resulting longer sampling interval will increase the time-lagging in hybrid

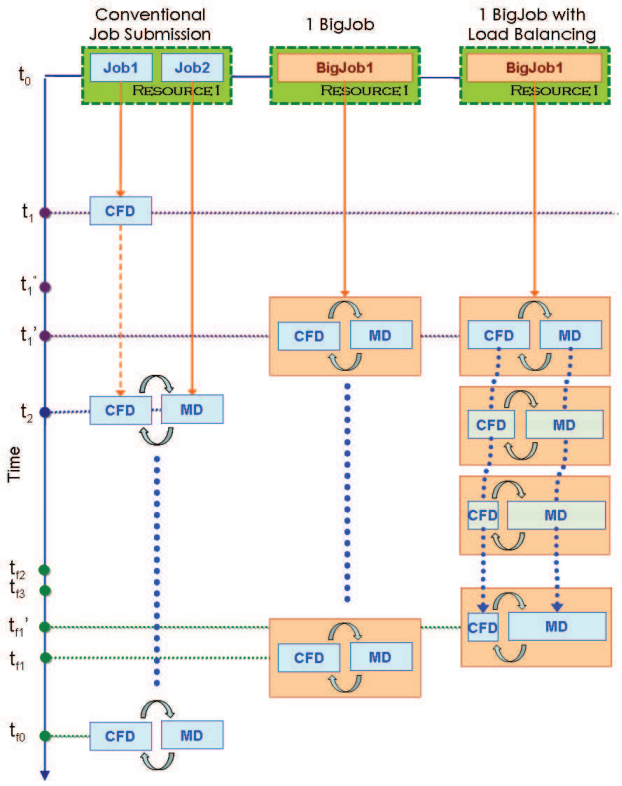

Fig. 6. Comparison of dependencies encountered for coupled simulations submitted as conventional jobs, to the scenario when using Pilot-Jobs. Here we use only 1 BigJob (S1). The conventional mode of submission experiences three phases based upon their queue state: (i) All jobs are waiting: $(t_1 - t_0)$; (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.

boundary. Thus, quantitative consideration on optimal scaling of these conditions are highly required.

Figure 7 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.

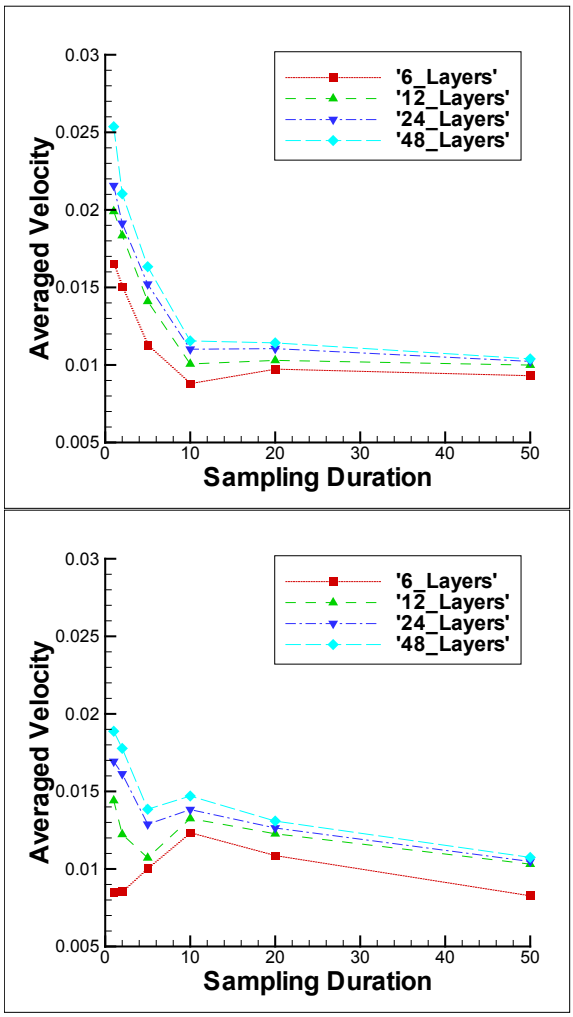

Fig. 7. L2 Norm of Averaged Velocity with Different Layer Sizes and Sampling Duration, at $\Delta t_{MD} = 5 \times 10^3$ (Left) and $\Delta t_{MD} = 1 \times 10^3$ (Right); Solution tends to produce less noise with bigger layer size and smaller MD timescale in short sampling duration. As sampling interval is increased, all solutions converges to the same fluctuation strength, which is $0.01\sigma/\tau$.

Resulting CFD and MD computational domains are depicted in Fig. 8. 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.

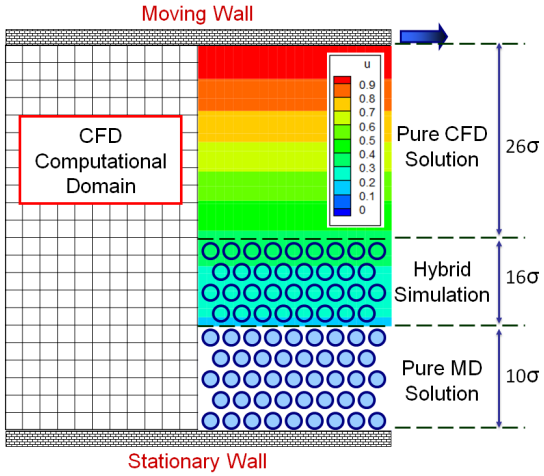

Fig. 8. Computational Domain of Couette Flow Simulation; The height of the fluid domain is 52σ ($\approx 177\mathring{A}$). CFD mesh size is 201×27 and CFD cells at the pure MD region are treated as holes. MD domain size is about 210σ in the X-direction and around 30σ in the Y-direction, including the 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. 9. 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 10 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

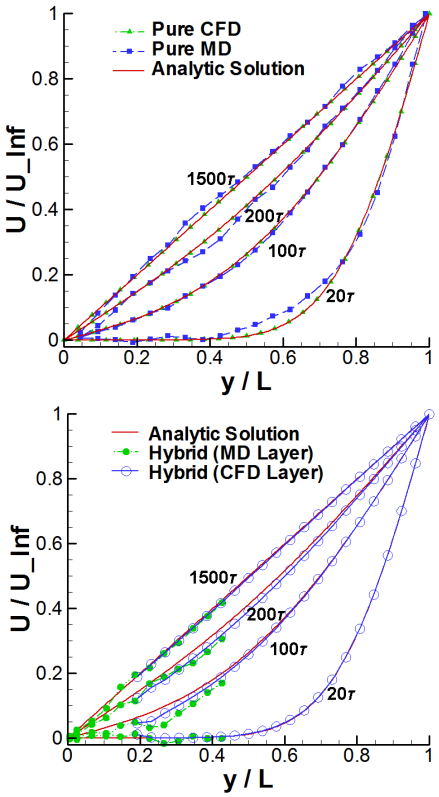

Fig. 9. Unsteady Couette Flow Profile; (Left) Pure CFD solution is exactly the same as analytic solution. MD solution shows the same flow pattern as analytic solution, though some fluctuation is observed. This verifies that CFD and MD represents the same flow physics. (Right) The steady result by hybrid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

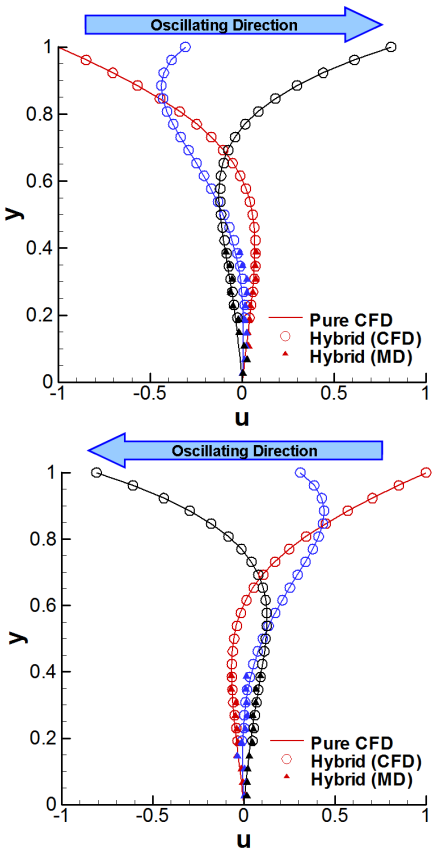

Fig. 10. Unsteady Flow Profile of Stokes Boundary Layer Problem; Solution by pure CFD simulation is denoted by a solid line and solution by hybrid simulation is presented by symbols. Hybrid flow profile globally matches well with CFD solution, though slight time-lagging in the MD side is observed. This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

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

TABLE II. 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

***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 ($\approx$ 500 px) are presented in a different fashion in Table II. 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 II 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 III. 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 subtasks throughout the simulation. For the result of a small simulation in Fig. 11, 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.

TABLE III. 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

The result of computation time evolution for a large simulation is seen in Fig. 12. For most experiments, which is given in the left side of Fig. 12, 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. 12. 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. 12 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.

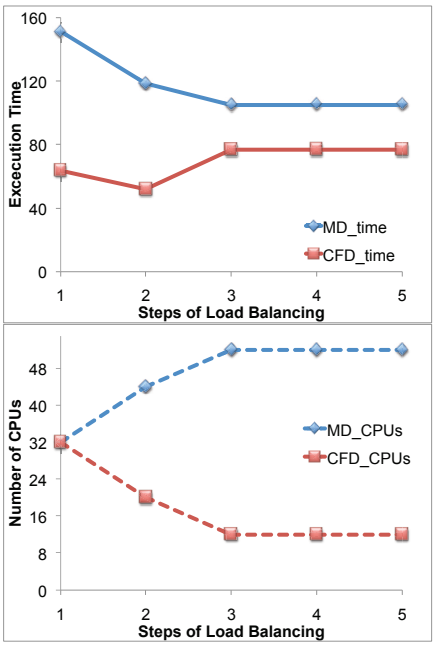

Fig. 11. Change of processor distribution between CFD and MD jobs and resultant computation time in the small simulation. A load balancer starts from the same number of processors assigned to both sub-jobs and detects 20 to 44 px between each sub-job as the optimal solution. The poor scalability of CFD 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.

References

[1] S. O'Connell and P. Thompson, "Molecular dynamics continuum hybrid computations: a tool for studying complex fuid flows," *Phys. Rev. E*, vol. 52, pp. R5792–R5795, 1995.

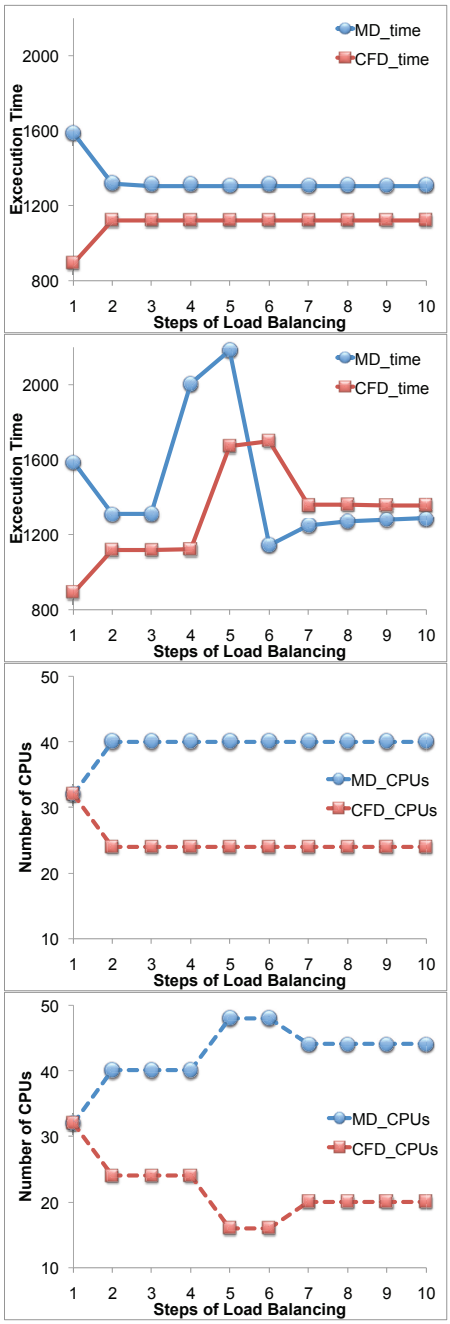

Fig. 12. (Left Column) Change of processor distribution between CFD and MD jobs and resultant computation time in the large simulation. A load balancer directly finds the processor distribution of 24 to 40 between CFD and MD jobs and remains the steady state until it completes after 25 simulation loops. 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.

[2] X. B. Nie, W. N. E. S. Y. Chen, and M. O. Robbins, "A continuum and molecular dynamics hybrid method for micro- and nano-fluid flow," *J. Fluid Mech.*, vol. 500, pp. 55–64, 2004.
[3] T. H. Yen, C. Y. Soong, and P. Y. Tzeng, "Hybrid molecular dynamics-continuum simulation for nano/mesoscale channel flow," *Microfluid Nanofluid*, vol. 3, pp. 665–675, 2007.
[4] R. Steijl and G. N. Barakos, "Coupled navier-stokes-molecular dynamics simulations using a multi-physics flow simulation framework," *Int. J. Numer. Meth. Fluids*, vol. In print: available online, 2009.
[5] M. Allen and D. Tildesley, *Computer Simulation of Liquids*. Oxford Science Publications, 1987.
[6] E. G. Flekkϕy, G. Wagner, and J. Feder, "Hybrid model for combined particle and continuum dynamics," *Europhys Lett.*, vol. 52, pp. 271–276, 2000.
[7] G. Wagner, E. G. Flekkϕy, J. Feder, and T. Jossang, "Coupling molecular dynamics and continuum dynamics," *Comput. Phys. Commun.*, vol. 147, pp. 670–673, 2002.
[8] N. G. Hadjiconstantinou and A. T. Patera, "Heterogeneous atomistic continuum representations for dense fluid systems," *Comput. Phys. Commun.*, vol. 4, pp. 967–976, 1997.
[9] N. G. Hadjiconstantinou, A. L. Garcia, M. Z. Bazant, and G. He, "Statistical error in particle simulations of hydrodynamic phenomena," *J. Comput. Phys.*, vol. 187, pp. 274–297, 2003.
[10] Y. C. Wang and G. He, "A dynamic coupling model for hybrid atomistic-continuum computations," *J. Comput. Phys.*, vol. 62, pp. 3574–3579, 2007.
[11] S. E. Rosers and D. Kwak, "An upwind differecing scheme for the time-accurate incompressible navier-stokes equations," *AIAA J.*, vol. 28, pp. 253–262, 1990.
[12] S. Yoon and A. Jameson, "Lower-upper symmetric-gauss-seidel method for the euler and navier-stokes equations," *AIAA J.*, vol. 26, pp. 1025–1026, 1988.
[13] M. M. Rai and S. R. Chakaravarthy, "An implicit form of the osher upwind scheme," *AIAA J.*, vol. 24, pp. 735–743, 1986.
[14] B. V. Leer, "Towards the ultimate conservative difference scheme. v. a second order sequel to godunov's methods," *J. Comput. Phys.*, vol. 32, pp. 101–136, 1979.
[15] J. M. Haile, *Molecular Dynamics Simulation. Elementary Methods*. Wiley, Chichester, 1992.
[16] D. C. Rapaport, *The Art of Motecutar Dynamics Simulation*. Cambridge University Press, 1995.
[17] K. Travis and K. Gubbins, "Poiseuille flow of lennard-jones fluids in narrow slit pores," *J. Chem. Phys.*, vol. 112, pp. 1984–1994, 2000.
[18] [Online]. Available: http://lammps.sandia.gov
[19] J. L. Steger, F. C. Dougherty, and J. A. Benek, "A chimera grid scheme," *Advances in Grid Generation*, vol. FED Vol. 5, pp. 59–69, 1983.
[20] R. Delgado-Buscalioni and P. V. Coveney, "Hybrid molecular-continuum fluid dynamics," *Phil. Trans. R. Soc. Lond. A*, vol. 362, pp. 1639–1654, 2004.
[21] Open Grid Forum. [Online]. Available: http://www.ogf.org/
[22] S.-H. Ko, C. Kim, O.-H. Rho, and K. W. Cho, "Load balancing for cfd applications in grid computing environment," *KSAS International Journal*, vol. 5, pp. 77–87, 2004.