A Massively Parallel Infrastructure for Adaptive Multiscale Simulations: Modeling RAS Initiation Pathway for Cancer

Francesco Di Natale[*]
dinatale3@llnl.gov

Harsh Bhatia[†]
hbhatia@llnl.gov

Timothy S. Carpenter[‡]
carpenter36@llnl.gov

Chris Neale[§]
cneale@lanl.gov

Sara Kokkila Schumacher[¶]
saraks@ibm.com

Tomas Oppelstrup[‡]
oppelstrup2@llnl.gov

Liam Stanton[‖]
liam.stanton@sjsu.edu

Xiaohua Zhang[‡]
zhang30@llnl.gov

Shiv Sundram[***]
shivsundram@gmail.com

Thomas R. W. Scogland[†]
scogland1@llnl.gov

Gautham Dharuman[‡]
dharuman1@llnl.gov

Michael P. Surh[‡]
surh1@llnl.gov

Yue Yang[‡]
yang30@llnl.gov

Claudia Misale[¶]
c.misale@ibm.com

Lars Schneidenbach[¶]
schneidenbach@us.ibm.com

Carlos Costa[¶]
chcost@us.ibm.com

Changhoan Kim[¶**]
ck624@hotmail.com

Bruce D'Amora[¶]
damora@us.ibm.com

Sandrasegaram Gnanakaran[§]
gnana@lanl.gov

Dwight V. Nissley[††]
nissleyd@mail.nih.gov

Fred Streitz[‡‡]
streitz1@llnl.gov

Felice C. Lightstone[‡]
lightstone1@llnl.gov

Peer-Timo Bremer[†]
bremer5@llnl.gov

James N. Glosli[‡]
glosli1@llnl.gov

Helgi I. Ingólfsson[‡]
ingolfsson1@llnl.gov

[*]Applications, Simulations, and Quality, Lawrence Livermore National Laboratory, Livermore, California, 94550
[†]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, California, 94550
[‡]Physical and Life Sciences, Lawrence Livermore National Laboratory, Livermore, California, 94550
[§]Theoretical Biology and Biophysics, Los Alamos National Laboratory, Los Alamos, New Mexico, 87545
[¶]IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 10598
[‖]Department of Mathematics and Statistics, San Jose State University, San Jose, California, 95192
[**]S. Sundaram and C. Kim are no longer affiliated with the respective institutions.
[††]Frederick National Laboratory, Frederick, Maryland, 21701
[‡‡]Lawrence Livermore National Laboratory, Livermore, California, 94550

SC '19, November 17–22, 2019, Denver, CO, USA

ACM ISBN 978-1-4503-6229-0/19/11…$15.00
https://doi.org/10.1145/3295500.3356197

ABSTRACT

Computational models can define the functional dynamics of complex systems in exceptional detail. However, many modeling studies face seemingly incommensurate requirements: to gain meaningful insights into some phenomena requires models with high resolution (microscopic) detail that must nevertheless evolve over large (macroscopic) length- and time-scales. Multiscale modeling has become increasingly important to bridge this gap. Executing complex multiscale models on current petascale computers with high levels of parallelism and heterogeneous architectures is challenging. Many distinct types of resources need to be simultaneously managed, such as GPUs and CPUs, memory size and latencies, communication bottlenecks, and filesystem bandwidth. In addition, robustness to failure of compute nodes, network, and filesystems is critical.

We introduce a first-of-its-kind, massively parallel Multiscale Machine-Learned Modeling Infrastructure (MuMMI), which couples a macro scale model spanning micrometer length- and millisecond time-scales with a micro scale model employing high-fidelity molecular dynamics (MD) simulations. MuMMI is a cohesive and transferable infrastructure designed for scalability and efficient

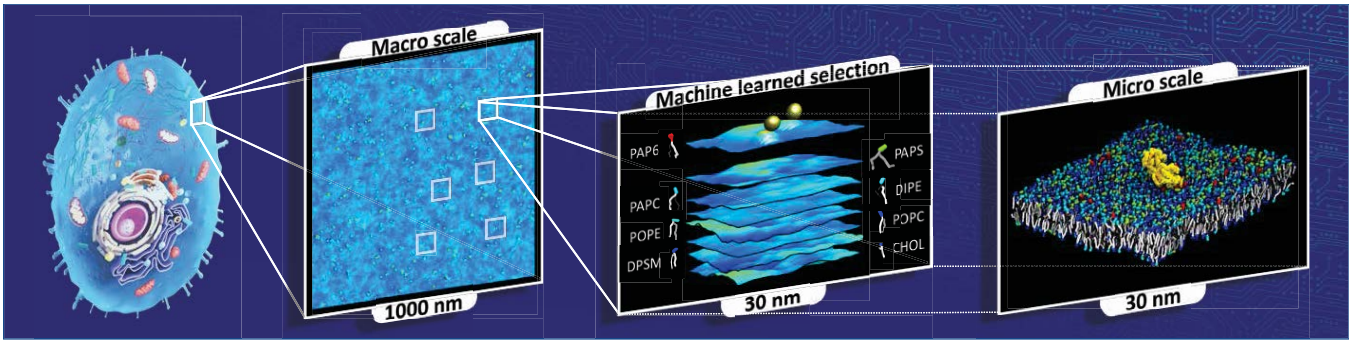


Figure 1: Addressing many important biological questions requires large length- and time-scales, yet at the same time molecular level details. Here we showcase the *Multiscale Machine-Learned Modeling Infrastructure* (MuMMI) by simulating protein-lipid dynamics for a 1 µm x 1 µm membrane subsection at near-atomistic resolution.

execution on heterogeneous resources. A central workflow manager simultaneously allocates GPUs and CPUs while robustly handling failures in compute nodes, communication networks, and filesystems. A hierarchical scheduler controls GPU-accelerated MD simulations and in situ analysis.

We present the various MuMMI components, including the macro model, GPU-accelerated MD, in situ analysis of MD data, machine learning selection module, a highly scalable hierarchical scheduler, and detail the central workflow manager that ties these modules together. In addition, we present performance data from our runs on *Sierra*, in which we validated MuMMI by investigating an experimentally intractable biological system: the dynamic interaction between RAS proteins and a plasma membrane. We used up to 4000 nodes of the *Sierra* supercomputer, concurrently utilizing over 16,000 GPUs and 176,000 CPU cores, and running up to 36,000 different tasks. This multiscale simulation includes about 120,000 MD simulations aggregating over 200 milliseconds, which is orders of magnitude greater than comparable studies.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; **Multiscale systems**; **Massively parallel and high-performance simulations**; *Simulation tools*; • **Applied computing → Computational biology**.

KEYWORDS

multiscale simulations, adaptive simulations, massively parallel, heterogenous architecture, machine learning, cancer research

ACM Reference Format:
Francesco Di Natale, Harsh Bhatia, Timothy S. Carpenter, Chris Neale, Sara Kokkila Schumacher, Tomas Oppelstrup, Liam Stanton, Xiaohua Zhang, Shiv Sundram, Thomas R. W. Scogland, Gautham Dharuman, Michael P. Surh, Yue Yang, Claudia Misale, Lars Schneidenbach, Carlos Costa, Changhoan Kim, Bruce D'Amora, Sandrasegaram Gnanakaran, Dwight V. Nissley, Fred Streitz, Felice C. Lightstone, Peer-Timo Bremer, James N. Glosli, and Helgi I. Ingólfsson. 2019. A Massively Parallel Infrastructure for Adaptive Multiscale Simulations: Modeling RAS Initiation Pathway for Cancer. In *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '19), November 17–22, 2019, Denver, CO, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3295500.3356197

1 INTRODUCTION

While supercomputers continue to provide more raw compute power, it is becoming increasingly difficult for applications to fully exploit these resources. The challenge of building a multiscale modeling capability utilizing modern supercomputing architecture — heterogeneous computing elements, deep memory hierarchies, and complex network interconnects — can be decomposed along two thematic axes: (1) the algorithmic challenges in managing increasing levels of parallelism within an application, and (2) the logistic challenges of scheduling and coordinating the execution of multiple applications across such diverse resources.

The prototypical approach to the first challenge are monolithic parallel applications able to simulate problems of unprecedented size and scale using full-system runs [30, 34, 67]. Conversely, the workflow challenge is often approached through massively parallel ensembles [56], which execute a large number of small- or medium-scale instances simultaneously. Here, we describe the creation of a novel simulation infrastructure that addresses both challenges and enables, as an example, the execution of a massively parallel, multiscale simulation steered by a machine learning (ML) approach, and orchestrated through a sophisticated workflow governing thousands of simultaneous tasks.

The scientific challenge to which we apply our novel infrastructure is an investigation of the interaction of RAS proteins with the cell membrane. Mutations of RAS contribute to a wide range of cancers as RAS modulates the signaling pathways that control cell division and growth. RAS activates signaling only when bound to lipid bilayers that form cellular membranes. This membrane association is an under-explored area of cancer biology that may be relevant to therapeutic intervention against cancer. We use MuMMI to facilitate the better understanding of the mechanism and dynamics of interaction between RAS, lipids, and other signaling proteins, which requires molecular-level detail and cannot be obtained experimentally with current technologies. MD simulations can simulate such interactions with the appropriate detail, but only for microscopic length- and time-scales (even on the largest computers). However, lipid concentration gradients and protein aggregation evolve over length- and time-scales hard to access through high-resolution MD.

Our approach to address this challenge of scale is to create a macro model for a cell membrane with RAS proteins and couple it to a ML-driven ensemble of MD simulations (Figure 1). The macro scale membrane model can reach experimentally observable time- and length-scales (μm and ms) with only moderate computing resources, but captures none of the molecular-level details of protein-protein and protein-lipid interaction. To provide molecular details of interest, we continuously monitor all RAS proteins and their local environments in the macro simulation and compare against configurations that have been previously seen, spawning MD simulations for configurations not previously sampled. The sampling process that determines which simulations are created is driven by a novel ML framework aimed at exploring the phase space of all possible RAS-membrane configurations. Limited only by available computing resources, this approach samples increasingly unusual configurations, effectively providing molecular-level detail on a macro scale. Automated feedback from MD simulations is used to continuously improve the macro model.

To achieve this automated multiscale simulation that can explore both the micro and macro scale, a sophisticated workflow is needed. MuMMI takes advantage of the heterogeneous compute architectures, and coordinates and links existing tools in a variety of programming languages. Our workflow is built upon the Maestro Workflow Conductor [15, 22], which provides an interface for scheduling the core simulations, as well as a number of supporting tasks, such as simulation setup, in situ analysis, etc. Maestro along with the Flux scheduler [3], provides a scalable scheduling solution.

Our *Multiscale Machine-Learned Modeling Infrastructure* (MuMMI) represents a new type of simulation framework that couples a diverse set of components through ML into a massively parallel application to address pressing scientific inquiries. Our key accomplishments are:

- A large-scale workflow manager that connects diverse software components into a single coherent, massively parallel, multiscale simulation;
- Innovations in scheduling and coordinating thousands of interconnected tasks and managing the resulting TBs of data;
- A demonstration on *Sierra* [45], the next-generation leadership supercomputer at Lawrence Livermore National Laboratory (LLNL), where we efficiently utilize the entire machine (176,000 CPUs and 16,000 GPUs) for several days aggregating about 120,000 MD simulations detailing a square μm membrane macro simulation with 300 RAS molecules over 150 μs.

2 RELATED WORK

As computational frameworks become more complex, scientific workflows are moving away from monolithic simulation codes and toward a complex web of interconnected tools, e.g., to pre- or post-process data, to execute ensembles for parameter studies, or to couple various different physics solvers. To manage such complex applications, a variety of workflow tools have been proposed to address different aspects of the overall challenge. One class of workflow solutions, such as Pegasus [21], Fireworks [38], or Kepler [5], grew out of the need to assemble complex post-processing capabilities, e.g., for large-scale experiments like the Large Hadron Collider. These workflows offer mature programming interfaces, distributed workflows, and data management solutions. A similar class of systems, such as Merlin [56], the UQ Pipeline [19], or SAW [27], focus on creating large ensembles, especially in the context of uncertainty quantification. Furthermore, these tools often contain various statistics and analysis capabilities or integrate packages, such as PSUADE [52] or Dakota [2], which provide their own job management capabilities. Collectively, these tools are primarily designed to operate in a batch-based, capacity-focused environment and do not provide the tight coupling needed for MuMMI. For example, many tools provide computational steering to enable adaptive sampling but enabling the tight feedback loop between the macro and micro models as well as the inter-job task placement and scheduling in these systems can be challenging.

Alternatively, there are frameworks designed to directly couple different physics solvers together as opposed to ochestrating tasks at a higher level. Examples for these approaches are preCICE [13], OpenPALM [24], or the OASIS coupler [18]. These approaches directly manage the exchange of information between solvers, communicating entire grids and/or boundary conditions in a massively parallel environment. However, they require access and modifications to the source code of the corresponding libraries and compatibility between the coupled software systems. In the case of MuMMI, we are coupling completely distinct simulation codes and infrastructure, many of which are written in different programming languages. Furthermore, our workloads are highly variable making the scheduling and job placement more complex than required for traditional multi-physics applications.

In summary, we require the system-facing aspects of the workflow and ensemble management systems to handle job scheduling, placement, and management, with a tighter coupling than existing solutions, and without the overhead of creating a joint multi-physics executable. To this end, we have created a sophisticated workflow to provide a highly integrated yet flexible solution that combines aspects of ensemble workflows with the integration of a joint multiscale application.

3 RAS-MEMBRANE DYNAMICS

Nearly a third of all cancers are driven by constitutively active (oncogenic) mutations in RAS proteins accounting for a high percentage of pancreatic (∽95%), colorectal (∽45%), and lung (∽35%) cancers [62]. KRAS is the most frequently mutated RAS isoform [78], and normally controls signaling through the MAP kinase pathway, a critical regulator of cellular growth, migration, and survival. Despite tremendous progress in RAS biology over the last three decades, no therapies are currently available. RAS proteins localize to the plasma membrane (PM) where they function as molecular switches by cycling between active (GTP-bound) and inactive (GDP-bound) states. Only active RAS binds and activates proximal effector proteins (i.e., RAF kinase) at the PM to propagate growth signaling. Specific localization of RAS to different regions of the PM may be important for the activation of signaling. However, it remains unknown how membrane composition (e.g., charged vs. neutral lipids), membrane dynamics (e.g., undulations and domain formation), and other physicochemical membrane properties affect the activity of RAS either directly or by regulating RAS orientation, localization, or clustering.

RAS-membrane dynamics and RAS-RAS interaction at the membrane are inherently a multiscale process because the protein-lipid and protein-protein interfaces are uniquely molecular, yet the long waiting times for association, which are largely dependent on diffusion, are inefficient to model in particle-based approaches. Therefore, MuMMI is an ideal platform for characterizing RAS-membrane dynamics. Here, we apply our multiscale approach to characterize the key events that trigger oncogenic signaling.

Many groundbreaking MD simulation studies have probed the relationship between structure, dynamics, and function in biological macromolecules. These studies include both all-atom (AA) and coarse-grained (CG) simulations containing millions to hundreds of millions of atoms/particles. Modeled systems include: the ribosome [79], viral capsids [32, 55, 57], the MexAB-OprM Efflux Pump [46], cytoplasm [80], hydrodynamic effects on lipid diffusion [74], protein crowding [25] and protein clustering [16]. Other studies include long continuous simulations extending into the high μs to ms range detailing: protein folding [61], RAS/RAF complexation [72], and plasma membrane organization [37]. In a different approach, ensembles of hundreds to thousands of shorter simulations have been combined to accumulate multiple ms of total sampling: for a membrane-embedded protein [42], to study protein folding and unfolding using an implicit solvent model [73], MscL gating [50], helix-helix interactions [6], and protein dimer and trimer assemblies [77]. However, simulations that represent the state-of-the-art in either size or duration are typically deficient in the other component (i.e., large but short or long but small). To reach larger systems over longer time-scales, additional coarse-graining can be done, combining whole molecule and/or proteins into single or a few interaction sites [75]. A number of ultra-CG models have been used to simulate systems of large length- and time-scales; such as viral particles budding [54], or protein-induced membrane vesiculation [58]. Ultra-CG models, however, are lacking in molecular level details and are not suited for all types of problems.

Although there have been many simulations of impressive duration and/or size, the real value of a simulation is more accurately quantified by the proportion of relevant phase space that has been sampled at sufficient resolution. Multiscale simulations have often been used to access large length- and time-scales at a coarser resolution and provide needed accuracy for select cases of interest. Multiscale approaches have been developed to couple different resolutions including quantum mechanics/molecular mechanics, AA/CG, AA/ultra-CG, and micro/macro [8, 9, 12, 14, 26, 36, 41, 60, 63, 65, 72] including recent work using both macro and micro scale simulations to model PM protein clustering [16].

4 MUMMI: AN OVERVIEW

MuMMI enables a new genre of multiscale simulation by coupling macro and micro scales using ML. Figure 2 illustrates how MuMMI integrates diverse software components that seamlessly work together, coordinated by a sophisticated workflow. Our framework drives a large-scale parallel simulation of the *macro model*, which is based upon dynamic density functional theory (DDFT) alongside a particle-based MD model. The macro model spans biologically-relevant length- and time-scales (μm and ms) that are intractable

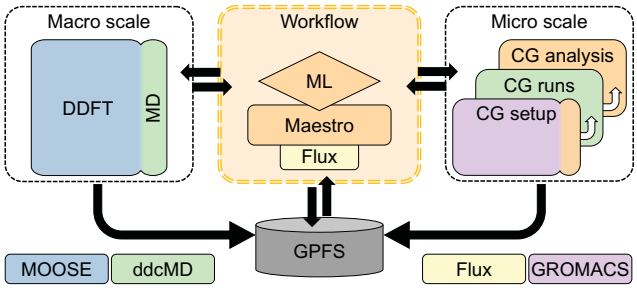


Figure 2: MuMMI couples the macro scale (DDFT and MD models) with the micro scale (CG model) using a ML-based sampling framework. Data resulting from the macro scale simulation is analyzed by ML, and interesting subregions are simulated at the micro scale. CG simulations are analyzed in situ and used to improve the macro model via on-the-fly feedback. The central workflow uses Flux as the resource manager, as abstracted using Maestro, and coordinates each of the software components using in-memory and on-disk communication. Modules in orange are the core, specially-developed components of our framework, and other colors represent external software that are extended for MuMMI.

for any single MD simulation; but, the macro model can not resolve fine spatial and temporal scales that are needed to understand protein-membrane and protein-protein interactions. A new *ML-driven importance-sampling framework* bridges the resolution gap by analyzing the resulting macro scale data and selecting the subregions of scientific importance. These selected macro scale *"patches"* are candidates for the higher-fidelity micro scale simulations. MuMMI launches micro scale CG simulations corresponding to the selected patches based on the availability of computational resources. A CG simulation consists of three phases: setup, simulation, and in situ analysis. The setup phase maps a patch to a molecular configuration that is then evolved, using MD simulations (ddcMD), to study the dynamics of the system. Since MuMMI is designed to support many thousands of CG simulations concurrently, storing the resulting data at the desired frequency is not feasible. Instead, the MD data is analyzed in situ, and the resulting analysis is saved with full-system coordinate data to disk infrequently. Finally, a key enabling technology in MuMMI is a self-healing feedback loop, in which the results of the more-accurate CG simulations are used to improve the parameters of the macro model.

5 WORKFLOW MANAGEMENT IN MUMMI

The workflow management in MuMMI has several roles while conducting thousands of concurrent simulations and enabling on-the-fly feedback (Section 5.1). New functionality was developed within Flux to support MuMMI job scheduling requirements (Section 5.2) and new strategies employed to manage the several PB of raw data resulting from our simulation campaign (Section 5.3).

Installation and deployment. With many diverse components, one of the first challenges in the development of MuMMI was a streamlined process of installation and deployment. To this end,

we use Spack [29] to install the entire infrastructure as a single package, accounting for differences in platforms, compilers, etc. We have developed MuMMI with portability in mind and deployed on three HPC clusters at LLNL with varying architectures. This paper discusses the deployment and testing on *Sierra*.

Initialization. MuMMI is initialized by requesting a full-size job allocation on the target machine. (e.g., 4000 computational nodes on *Sierra*). One node is assigned as the Flux master for job scheduling and resource management activities. A portion (24 CPU cores) of a second node is dedicated to our workflow to perform higher-level tasks, e.g., such as deciding which simulations to run. All remaining resources are used for performing simulations. In order to leverage the heterogenous architecture of the target machine, MuMMI distinguishes between different tasks based on their resource requirements and uses appropriate job placement strategies to schedule jobs according to the resources needed. In our current application, the CG simulations (using ddcMD) require GPU access, whereas all other components (CG setup, CG analysis, and the macro simulation) work entirely on CPU cores.

5.1 The Central Workflow Manager

The *workflow manager* (WM) is responsible for steering the framework toward the target multiscale simulation based on scientific interest. The WM is designed to be highly configurable and performs several key tasks, e.g., continuously polling the macro model for new patches, scheduling new CG simulations, and performing periodic feedback. Almost all details of these tasks, e.g., the configuration of the underlying machine and the frequency of a certain task, are controlled via configuration files. The WM is written in Python and uses Maestro [22] — an open-source workflow conductor with a simple API — to abstract the interface with the underlying job scheduler, resulting in a portable WM. Upon initialization, the WM loads a pre-trained ML model that guides the selection of patches for CG simulation.

Generation of patches from the macro model. As the macro model simulation executes, snapshots of the resulting data are saved to an instance of the IBM® Spectrum Scale™ (GPFS). The WM continuously polls for new snapshots, and generates several patches (spatial regions of scientific interest) per snapshot. At full scale, the macro simulation delivers new snapshots with 300 patches every 150 seconds as binary data files, and the WM processes them immediately. Patches are stored as serialized Python (pickle) objects in a tar archive file (described in Section 5.3) and indexed by identifiers for rapid access.

Ranking and selection of patches using ML. As new patches are generated, they are analyzed for their "importance" in real-time. This importance metric is used to create an in-memory priority queue of all patches seen thus far. However, the importance metric of patches is dynamic and changes frequently. Therefore, maintaining and re-evaluating an ever-growing list of patches is computationally prohibitive. By design, the importance of a given patch cannot increase over time. Therefore, we truncate the queue to a computationally-feasible and scientifically-relevant length because the discarded patches are deemed too uninteresting.

Tracking system-wide computational resources. A key responsibility of the WM is to track the available computational resources, which is done indirectly by tracking the tasks currently running and their known allocation size. The WM uses Maestro to query the status of running jobs (previously started by the WM) from which the available resources (with and without GPU) are calculated.

Scheduling of CG simulations. When new resources are available (during the loading phase of the workflow or when a previously-running job concludes), the WM launches new jobs with matching resource requirements. Patches are selected from the priority queue to start new CG setup jobs based on need and to match available resources. Completed CG setup systems are selected to run as new CG simulations, minimizing the wait time between running modules. The WM allows staggering scheduling of new jobs to reduce the load on the underlying scheduler, which is useful when executing large simulations on several thousands of nodes.

Managing the feedback from micro to macro model. Periodically, the WM triggers the feedback mechanism, which aggregates properties of interest captured from the MD simulations via in situ analysis and uses them to update the parameters of the macro model. The current design of our feedback mechanism uses the filesystem. With 120,000 CG simulations, the scan of latest data from running and completed simulations can take over 30 minutes. As a result, for the application at hand, MuMMI's feedback frequency is set to 2 hours. The limitations of the filesystem poses scalability challenges. We plan to mitigate this limitation by bypassing I/O operations and using faster communication between CG analyses and the WM (discussed in Section 5.3).

Checkpointing and restarting. To account for systems errors due to node failures, file corruption, GPFS failures, etc. that invariably occur, the WM monitors all running jobs. Failed jobs are automatically restarted at the last available checkpoint. For redundancy and protection against control data being corrupted, all status files are duplicated. As a result, MuMMI is capable of running simulations for hundreds of hours of wall time at full scale. More specifically, the WM uses several checkpoint files to save the current state of the simulation in a coordinated manner, which can be used to restore the simulation, potentially with different configurations or even on a different machine.

5.2 Job Scheduling

MuMMI distinguishes jobs based on their resource requirements. In particular, the CG simulations are the only component of MuMMI making direct use of GPUs, while using relatively little CPU resources, even accounting for its attendant in situ analysis job. Therefore, given that each node on *Sierra* consists of four GPUs and 44 CPU cores, four CG simulations (one per GPU) on each node is executed. Each CG simulation is bound to two CPU cores that share cache, to maximize the cache available to ddcMD. The corresponding analysis jobs occupy 3 cores for each CG simulation, totaling to 20 cores utilized by the CG simulations. In order to reduce latency in communication with GPUs, this 20-core partition is bound exclusively to the cores closest to the PCIe buses. The remaining 24 CPU cores per node are dedicated to CPU-only jobs and are either the macro simulation, a CG setup job, or the workflow manager.

Although this job arrangement works well for machine utilization, a number of scheduling challenges arise. In particular, even scheduling the CG simulation jobs in groups of four, several thousand jobs still run concurrently. Avoiding oversubscription, or placing too many jobs on the same node, requires the ability to co-schedule multiple jobs on the same node based on heterogeneous resource requirements and is an uncommon practice in HPC. Most importantly, jobs have to be dynamically scheduled so that jobs that finish early are backfilled and efficiently utilize the available resources.

Our initial attempts aimed at using LSF® [64] and *jsrun* [33] job schedulers available natively on *Sierra*. However, considering the requirements of the workflow, these options proved insufficient. The large number of jobs required relatively high throughput to quickly reach the steady state of machine utilization. Although *jsrun* could deliver the required speed, it lacked dynamic scheduling at the time. LSF, on the other hand, provides dynamic scheduling and scheduling by heterogeneous resources but is configured (on *Sierra*) to not allow multiple jobs to be scheduled onto a single node. Given these limitations, we use Flux [3], a resource manager in active development at LLNL. Flux demonstrates the features required to support our scheduling requirements. Since Flux is designed to be configured and run directly by the user inside of jobs, we configured it for co-scheduling and heterogeneous resource scheduling. More specifically, in order for MuMMI to start up to 36,000 concurrent tasks, we exploit Flux's ability to hierarchically schedule sub-instances and to specify explicit bindings of resources. In this case, we first schedule jobs to nodes and enforce the optimal placement of jobs on each node as discussed above. Flux can be launched within the framework of Slurm [39] and LSF, which lends portability to our framework across different platforms. To ensure an additional separation of concerns MuMMI uses a Maestro plugin for Flux, allowing the WM's interface to remain virtually independent to the ongoing development within Flux and the option to switch schedulers in the future.

5.3 Data Management

MuMMI is designed to run on supercomputers and address applications on an enormous scale. But, these applications pose significant data management challenges. For example, our simulation of RAS on the PM produced over 200 ms of CG trajectory data. Recording snapshots at 0.5 ns intervals would generate over 400 million files for snapshots alone, occupying over a PB of disk space. Including files for stored analysis, logs, restart, macro model, and ML would increase the number of files by a factor of 10. Managing and utilizing the data at this scale would require a sustained filesystem throughput on the order of 10,000 I/O operations per second for the entire duration of the simulation. During our campaign on *Sierra*, GPFS was considerably upgraded and ended up consisting of 154 PB of usable storage with 1.5 TB/s peak bandwidth.

In order to reduce the demand on the filesystem, we used a combination of local in situ analysis and data aggregation strategies to minimize bandwidth and I/O operations, especially expensive metadata operations related to creation and deletion of files. In particular, each CG simulation saves snapshots every 0.5 ns to a local on-node filesystem (RAM disk). All the snapshots are analyzed using a corresponding in situ module that extracts information of interest, prunes snapshots to every 2 ns, and periodically (every 20 ns) saves local data to parallel (global) filesystem, appending the snapshots and analysis to archive files. Together, these choices reduce the number of files by 3 orders of magnitude, and the total amount of data by a factor of 4.

For data aggregation, we chose the tar file format [35], augmented with an index file (stored separately) to allow quick retrieval of archive member files. Other more-advanced file formats, e.g., HDF5 [69], were considered, but not chosen because they modify file headers on every update, which could lead to data corruption in cases of untimely termination, e.g., due to node failure or time out. Instead, the tar format allowed us to use the files in append-only mode, so that previously stored data would never be overwritten or corrupted, even due to bugs in the code. Additionally, the tar format is portable, and the corresponding archives can be inspected and unpacked using standard tools. This aggregation approach provided robustness against node failure, filesystem time outs, and unexpected job terminations.

To further improve our data management strategy, we have explored the incorporation of the IBM®Data Broker (DBR) [59] into MuMMI. The DBR implements an in-memory key-value store for fast data storage and retrieval with database level fault tolerance. Furthermore, the corresponding backend Redis™ [43] server, configured as a cluster, can be tuned for the specific machine and allocation size. Initial experiments suggest that using 10–100 nodes on *Sierra*, CPU only and shared with running CG simulations, would be sufficient to hold all necessary feedback and control data. For 50 Byte key and 257 KB value pairs, we observe average latencies of 0.4 and 0.5 ms per read and write operations respectively. Unfortunately, at the time of our simulation on *Sierra*, the network and filesystem as well as the DBR were not fully mature, and the combination proved unstable. However, going forward, the expected performance of the DBR could replace many of the frequent I/O operations in MuMMI and significantly improve the overall performance of the workflow especially the on-the-fly feedback.

6 MULTISCALE SIMULATION USING MUMMI

Having presented the central WM in MuMMI, we next discuss the innovations in the simulation of the macro and the micro models needed to enable our multiscale framework. We also discuss the ML-based sampling and the on-the-fly feedback modules that were developed to couple the two scales together. We note that all ranges presented throughout this paper are formatted as a mean ± standard deviation, unless otherwise stated.

6.1 The Macro Model

In order to rapidly explore long time- and length-scale behavior of RAS protein membrane dynamics, we developed a *macro model* based on the classical approximation theory for liquids. Specifically, our model uses DDFT to describe the lipid-lipid behavior [47]. In this model, the lipid bilayer is represented as a two-dimensional surface. Each protein and its conformational state on the bilayer is represented by a single particle, which interacts with the lipids through a potential of mean force (PMF). The proteins are modeled

using Langevin equations [44] and interact with each other through a radially symmetric pair potential.

The input parameters to the macro model are lipid-lipid direct correlation functions and self-diffusion constants, RAS-lipid and RAS-RAS potentials, and RAS diffusion constants. These parameters were derived from several hundred CG PM and RAS PM simulations. The direct correlation functions were calculated using lipid-lipid radial distribution functions (RDFs) in conjunction with the Ornstein-Zernike equations [53]. The RAS-lipid PMFs were derived from RAS-lipid RDFs using the Hypernetted-chain closure relation [49]. The RAS-RAS potential was estimated from RAS-RAS PMF simulations and expected association strength.

Within the framework of DDFT, the governing equations are dictated by an appropriate free-energy functional, which yields a chemical potential for the i^{th} species, μ_i, by taking the variational derivative of the free energy with respect to the lipid density of that species. In the macro model, we can decompose each chemical potential as $\mu_i = \mu_i^{LL} + \mu_i^{LP}$, where the lipid-lipid chemical potential μ_i^{LL} and the lipid-protein chemical potential μ_i^{LP} are taken as:

$$\mu_i^{LL} = k_B T \left(\ln(n_i) - \sum_{j=1}^{N} n_j * c_{ij} \right), \quad \mu_i^{LP}(\mathbf{r}) = \sum_{k=1}^{P} u_i \left(|\mathbf{r} - \mathbf{r_k}| \right).$$

Here, $n_i = n_i(\mathbf{r}, t)$ is the local density for lipid i at time t in position $\mathbf{r}$, and $u_i = u_i(r)$ is the circularly symmetric protein-lipid PMF as a function of distance r, for each of the i^{th} lipid type. The direct correlation function between lipid types i and j is given by $c_{ij} = c_{ij}(r)$ for the distance r, and $\mathbf{r}_k$ is the position of protein particle k on the membrane. Finally, k_B is the Boltzmann constant, T is the absolute temperature, and the operation $(*)$ denotes a convolution: $(n * c)(\mathbf{r}) = \int n(\mathbf{r'}) c(|\mathbf{r} - \mathbf{r'}|) d\mathbf{r'}$. The evolution equations of DDFT are given as

$$\frac{\partial n_i}{\partial t} = \frac{D_i}{k_B T} \nabla \cdot n_i \nabla \left(\mu_i^{LL} + \mu_i^{LP} \right) + \xi_i,$$

where D_i is the self-diffusion coefficient for lipid i and $\xi_i = \xi_i(\mathbf{r}, t)$ is a conservative noise term that represents density fluctuations due to hidden degrees of freedom.

We implemented our macro model into the MOOSE finite element framework [70]. This implementation consisted of modules to compute the nonlocal correlation function convolutions, evaluate the lipid-lipid evolution equations, compute the protein-lipid interactions, and export the current lipid distribution to the protein particle integrator. For integrating the protein equations of motion, we used ddcMD (see Section 6.4). In this campaign, we used the macro model to simulate a 1 μm × 1 μm bilayer, at a resolution of 1200 × 1200 cubic-order elements, with 300 RAS molecules. We observed a rate of 6.3±0.12 and 12.0±0.25 μs per day for 900 and 2400 MPI tasks, respectively.

6.2 ML-based Importance Sampling

As the macro model is running, it explores the phase space of local lipid fluctuations. In this work, we are interested in the lipid configurations underneath RAS to understand how RAS affects lipid behavior and vice versa. Simulating all possible local neighborhoods of RAS (*"patches"*) would be computationally infeasible. Instead, we sample the corresponding phase space of patches.

Because certain lipid configurations are much more common than others, and simulating similar lipid configurations is wasteful as they do not deviate far enough from previous simulations, random selection of patches would be inefficient because selections would mimic the distribution of the phase space, i.e., commonly-occurring configurations will be selected often, whereas rare events will likely be ignored. Instead, we use a deep neural network to create a latent representation that captures the lipid configurations, and use farthest-point sampling in the latent space to identify "important" (dissimilar to previously simulated) patches.

Specifically, for each timestep of the macro model, we extract a 30 nm × 30 nm patch underneath each RAS molecule. Each patch consists of 14 lipid densities (8 lipid species in the inner leaflet and 6 in the outer), and is represented as a $5 \times 5 \times 14$ grid of lipid concentrations, which can be used to initialize a corresponding CG simulation. We use a deep neural network to construct a variational autoencoder (VAE) [23] that maps the $5 \times 5 \times 14$ patch into a 15-dimensional latent space, where each dimension represents a complex, nonlinear degree of variation in the behavior of the input data. We choose a VAE because it provides several favorable mathematical properties, such as a continuous distribution in the latent space, which are important for statistical analysis on the importance sampling. We developed several VAE models using Keras [17] and Theano [11] frameworks, varying the number, widths, and types of layers in the VAE as well as different sizes of the output latent space. Different latent spaces were evaluated using reconstruction loss of the corresponding VAE; a detailed discussion on the evaluation of ML models is beyond the scope of this paper. The final 15-dimensional latent space model was chosen due to its superior balance between preserving the relationships between lipid concentrations, and the computational advantages of smaller dimensionality. The chosen VAE uses a combination of fully-connected and convolutional layers, and makes use of batch normalization as well as dropouts to minimize reconstruction error.

As macro model simulation creates new data, we add the new patches into a priority queue, which is sorted based on the distance (in latent space) from patches corresponding to the previously-executed CG simulations. We use Faiss [40] to create an efficient data structure that performs fast, approximate-nearest-neighbor queries in the latent space, allowing for almost-real-time evaluations of importance metrics. With availability of new computational resources, the highest ranked patches are used to initiate CG simulations. In this manner, we progressively sample the phase space of all patches at a uniform and continually increasing density. Given sufficient computational resources, MuMMI will ultimately cover the phase space of all possible lipid configurations underneath RAS molecules densely enough to perform an analysis of the entire macro model at the scale of the CG model. In particular, for any RAS molecule at any time step of the macro model we can find a CG simulation that represents a lipid configuration close enough to the patch in question to inform the given analysis query.

6.3 Setup of the CG Simulations

The CG MD simulation setup module (*CG setup*) transforms a continuum macro model representation into a particle-based micro representation (see Figure 3). A selected 30 nm × 30 nm patch from the

macro model is instantiated and equilibrated for a Martini [48] CG simulation. Within the patch, the macro model dictates the number, states, and locations of RAS proteins, as well as the concentration and asymmetry of all membrane lipids, which are resolved down to a 5×5 subgrid. The proteins, lipids, ions, and water molecules are constructed using a modified version of the *insane* membrane building tool [76]. The modified *insane* tool allows for specifying lipid concentration with a subgrid resolution in each membrane leaflet. The proteins' initial conformations are sampled from pre-constructed libraries based on their conformational state, randomly rotated with respect to the membrane plane and packed at their specified coordinates. The GROMACS MD package [1] (CPU-only version) is used for energy minimization, equilibration, and to pull the proteins to the bilayer. Using only the CPU cores allows the workflow to set up new CG simulations without competing with currently running CG simulations. All simulations were run using the new-rf Martini parameter set [20], with a final time step of 20 fs, at 310 K and 1 bar semiisotropic pressure coupling. The setup process includes particle creation, 1500 energy minimization steps, and a total of 425,000 steps of equilibration for $\sim$140,000 particles.

6.4 GPU-accelerated CG Simulations

Once the particle systems are equilibrated they are queued for MD simulations using ddcMD [66]. ddcMD is a highly scalable general-purpose MD application with a flexible domain decomposition capability. ddcMD has previously been used to study a variety of problems in the areas of material science, fluid flows, and plasma physics. Its performance has been acknowledged twice with Gordon Bell Prize awarded to teams using ddcMD [30, 67]. In order to support the multiscale simulations targeted in this paper, we made significant extensions to ddcMD. In particular, new GPU capabilities were added to accelerate the Martini CG force field.

Given that there already exist several GPU-enabled bio-MD codes, our decision to extend ddcMD was motivated by the need for not only a GPU-enabled high-throughput MD using the Martini force field, but also one that minimizes CPU utilization. Minimizing CPU usage is critical when working on architectures with low CPU

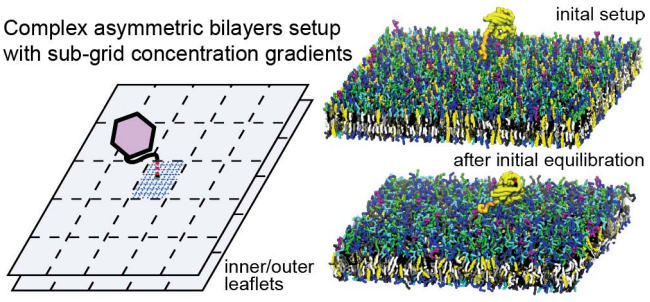


Figure 3: Particle-based micro simulations are created based on input from the macro model. CG MD simulations are instantiated based on protein location and state as well as sub-grid lipid concentration specificity. Snapshots of a selected patch (5×5) with one RAS molecule is shown after construction and after initial equilibration.

to GPU resources or when executing frameworks, like ours, with high CPU demand from other tasks. As such, no existing MD code in the computational biology community meets these constraints.

The GPU implementation in ddcMD to support biomolecular simulations places the entire MD loop on the GPU. This includes both bonded and nonbonded energy terms, neighbor table construction, barostat (Berendsen) [10], thermostat (Langevin) [4], pair constraints [7], and integrator (velocity Verlet) [68]. Therefore, compared to other codes ddcMD only needs to copy particle state (position, velocity, forces, box size, etc) from GPU to CPU infrequently; only when output or analysis is needed. Only one CPU core is used in ddcMD, primarily to handle the setup of the simulations and output the data. After the simulation is initialized on the CPU, all of the data is copied to the GPU memory, and all computations are performed on the GPU. We applied several techniques to improve the performance of GPU kernels:

- Improved thread scheduling in the nonbonded kernel by assigning a single thread per particle. Processing a particle's neighbor list would result in enough threads to fill the GPU; however, memory locality within a particle's neighbor list is better than the locality between two particles' neighbor lists. We determined that 8 threads per particle for the Martini force field yields the best performance.
- Enforced coalesced memory accesses. Since each particle has a set of 8 threads, ensuring these threads access contiguous memory results in the better locality and fewer bank conflicts.
- Refactored data structures containing separate arrays of energies, forces, and virials of each particle to be interleaved within a single array. By interleaving the arrays, we optimize locality for write access patterns.
- Use of shuffle-sync based reductions in lieu of shared memory reductions. We found that the need for shared memory was eliminated by switching to warp-level shuffle intrinsic reductions. This optimization is beneficial for large systems which may run out of shared memory.

A series of different test cases have been performed on CPU (Intel® Xeon® E5) and GPU (Nvidia® Tesla™ V100). The speedup of GPU over CPU is $\sim$300 fold.

The community standard MD simulations with the Martini force field is GROMACS [1]. GROMACS is designed to distribute the calculations between CPU and GPU with an automated CPU-GPU load balance scheme. To compare the performance of ddcMD and GROMACS, we chose a typical MD simulation used in our framework: a 135,973-bead Martini simulation of a single KRAS protein on an asymmetric 8-component, 3077 lipid bilayer. Figure 4 summarizes our comparison. We note that although GROMACS has a better single-simulation per node performance than ddcMD when the usage of the CPU cores is increased, ddcMD outperforms GROMACS on the more relevant benchmark of four-simulations per node and single-core per simulation by a factor of about 2.3. One of the biggest bottlenecks for GPU acceleration is the bandwidth between CPU and GPU, and frequent movement of data back and forth between CPU and GPU is inefficient. The design of GROMACS requires copying and synchronizing the data between the CPU and GPU at every time step, which leads to performance degradation

when the node is more completely occupied, even by separate simulations.

Additionally, ddcMD's implementation of the MD loop on GPU uses double-precision arithmetic, as compared to the single-precision calculations done by GROMACS. We are currently evaluating optimization associated with the reduced precision, and expect to find further performance improvement. Overall, for a typical MD simulation for our system, we observe an idealized performance of 1.08±0.01 µs per day when executing ddcMD on a single core and GPU.

6.5 In situ Analysis of CG Simulations

An integral part of MuMMI is the ability to carry out in situ analysis of MD simulations. This feature is essential for dealing with such vast numbers of CG simulations, in particular, to limit storage and I/O requirements, as well as to provide on-the-fly feedback. On each node, custom Python analysis modules are run on the CPUs. For each simulation, newly generated snapshots are saved locally using a fast RAM disk and consumed locally by running analysis modules. The molecular structure is read using an extended version of the MDAnalysis package [31] [51], that can parse the native ddcMD binary and ASCII data formats. The online analyses to be performed are designed and chosen based on parameters of interest from preliminary simulations and those needed for re-optimization of the macro model. Examples of features of interest are RAS-RAS contacts, RAS-lipid contacts, RAS orientation, and lipid distributions. The result of these analyses are gathered locally and intermittently written to GPFS. All analysis routines are optimized to be completed within the time frequency of new frames being written. Having this analysis instantly available during the simulation allows for efficient exploration of the data while the simulation is running as well as constant improvements of the fidelity of the macro model through online feedback.

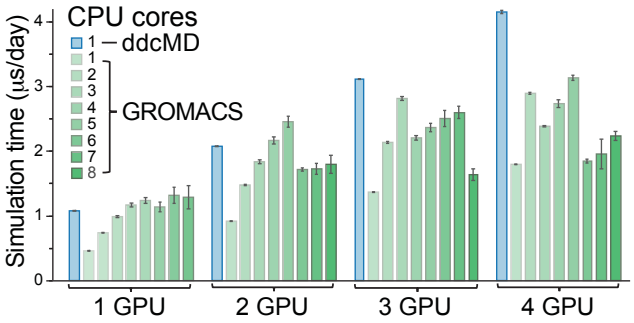


Figure 4: Comparison of job performance using GROMACS and ddCMD for one to four simulations running on a single node of *Sierra* using typical MD simulation from the workflow as the benchmark. All simulations use a single GPU per simulation and are averaged over 10 runs (with standard deviations shown as error bars). All ddcMD simulations use a single CPU core, whereas GROMACS simulations are multicore with one to eight cores per simulation (shown in different shades of green).

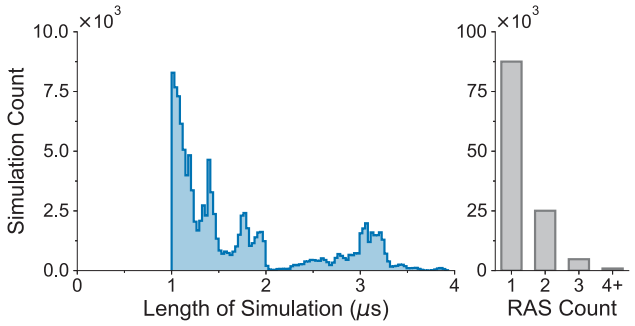


Figure 5: MuMMI enabled a scientific campaign at an unprecedented scale, generating about 120,000 (119,686) CG simulations. These simulations ran from 1 to 4 µs, aggregating over 200 ms of MD trajectories, and were distributed with respect to RAS counts according to the required scientific criteria.

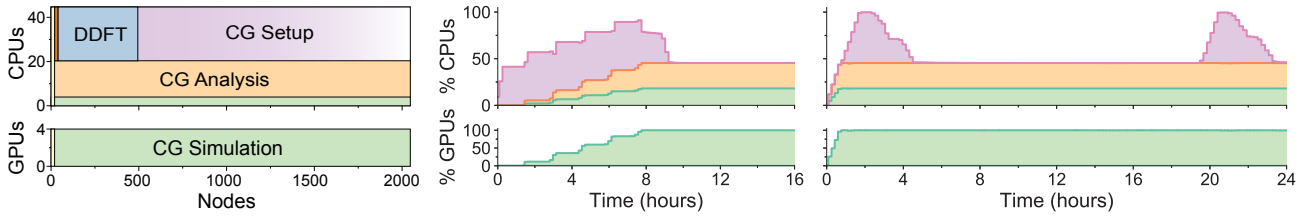
6.6 On-the-Fly Feedback

The final and one of the most critical components of our framework is an on-the-fly feedback mechanism from micro to macro scale. The (initial) macro model used in the framework was parameterized using previously executed CG simulations. However, this preliminary data was rapidly dwarfed by the output of the current campaign, both in the number of CG simulations and the variations in sampled local environments. Therefore, the fidelity of the macro model, which depends on parameters derived from initial training data, is limited. Instead, MuMMI provides a unique opportunity to continuously improve model parameters.

We use an on-the-fly feedback loop, where the in situ analysis of CG data is used to update the macro model parameters. In particular, we compute the protein-lipid parameters, RDFs, between the proteins in their different states and all the lipids. These RDFs are captured by the WM periodically, weighted based on the prevalence of each simulation (as dictated by the ML framework), and the updated RDFs are used to construct new free-energy functionals to use in the macro model. The result of this on-the-fly feedback is the progressive improvement in the accuracy of the macro model, as the parameters will now be based on ms of cumulative particle simulation data. More importantly, the CG simulations explore lipid compositions that are accessible via concentration fluctuations and can be properly reweighted, something that was not possible in the preliminary simulations used to construct the initial macro model.

7 RESULTS

MuMMI enabled us to study RAS protein dynamics on a PM by running a multiscale campaign on *Sierra* over several days. This multiscale simulation aggregated over 200 ms of MD trajectories and analyzed more than 300,000,000 frames as part of ∽120,000 MD simulations (Figure 5), with the final dataset consuming over 320 TB of disk space. Our campaign surpasses similar existing large-scale MD simulation efforts by orders of magnitude, representing a wealth of new information, potentially leading to new insights



(a) Typical resource allocation used by MuMMI (b) Resource utilization during the initial run (c) Resource utilization during a typical restart run

Figure 6: MuMMI leverages heterogenous architectures by allocating resources as needed by different tasks. (a) shows the allocation of 2040 nodes on *Sierra* for a typical run of our scientific campaign. A single node each is needed for the job scheduler (yellow) and the workflow manager (brown), with all remaining resources employed for simulations. (b) and (c) show the machine utilization for an initial run and a typical restart. Demonstrating that MuMMI is capable of achieving 100% resource utilization in less than one hour.

in understanding the role of RAS in cancer initiation. Successfully completing such a campaign at an unprecedented scale required making effective use of the available computational resources.

Each computational node of *Sierra*, currently ranked 2nd on the TOP500 supercomputing ranking [71], contains 44 3.45 GHz POWER9® processors and four NVIDIA® Tesla™ V100 GPUs. During the course of this campaign, we were granted dedicated access to 2040 nodes of *Sierra* for several continuous periods of 24 hours. Other configurations were explored ranging down to 15 nodes (without a running macro model) and up to 4000 nodes (utilizing the full machine). All results discussed in this section represent a typical 2040-node run; similar trends were observed when scaling to the full machine.

7.1 Resource Utilization

As described in Section 5.2, we used Flux to allocate portions of a node to each different type of task. Per our design, we dedicated all available GPU resources to the compute-intensive CG simulations and enforced the 20/24 split of CPU cores. Illustrated in Figure 6a, the 20-core partition was dedicated to the CG simulations and analysis, reserving the remaining cores for the macro model simulation, CG setup processes, and the WM. In a typical $n = 2040$ node setup, one node was dedicated to Flux and one 24-core partition to the WM. The single macro model simulation occupied the 24-core partition of up to 500 nodes ($d = 500$). CG simulation bundles successfully utilized both the 20-core partition and GPUs on the remaining $n-1$ nodes. We note that each pair of CPUs on a *Sierra* node share an L2 cache. In order to maximize the L2 cache available to ddcMD, each CG simulation was allocated two adjacent CPU cores, although ddcMD only used a single core. We were also asked to leave at least two cores free by the administrators for ongoing system processes, which we counted within the four extra ddcMD cores. Finally, the 24-core partitions of the $n-d-2$ remaining nodes were reserved for short-lived CG setup processes that are started only as the supply of setup CG systems starts to be depleted. We note that in order to prevent over-saturation of the system with similar patches, the ML algorithms slowed selection to prevent a stale buffer of selected patches so that newly identified patches can be considered for setup.

Figure 6 illustrates the machine utilization for two characteristic runs. For simplicity, these figures show only the $n-d-2$ nodes where the macro model was not running. Figure 6b shows the initial run where the multiscale simulation was being started. In the beginning, the workflow manager has not seen any patches and therefore takes ∽8 hours to fully load the GPUs of all 2040 nodes. We observed that setup of the first batch of patch selections finished in the expected 90 minute time frame with the corresponding ddcMD jobs starting immediately thereafter. A typical run can support a total of 8160 CG simulations but can only setup 1538 CG systems at a time. Therefore, we observed "step-like" patterns in the GPU utilization curves about every 90 minutes as batches of CG system setup completed and filled the next portion of available GPUs. Overall, our framework took just over 7.5 hours to achieve full utilization of all available GPUs when starting from scratch. Each CG (ddcMD) simulation was configured to simulate over 1 µs of MD, running a minimum of 24 hours, causing the framework to significantly slow down the selection of new patches. In particular, in the two hours following the saturation of all GPUs, the framework created the buffer of CG setups described above. The rest of the run makes steady use of 100% GPUs and about 45% CPU cores (20 out of 44 cores). Figure 6b is cropped at 16 hours, after which we observed steady-state behavior.

The restart capabilities in MuMMI make it straightforward to continue the simulation from the previous states. Figure 6c highlights the typical utilization pattern during a restart. Since restart runs already have CG simulations that were either previously running or were already set up, the delay until full utilization is reduced significantly. In particular, MuMMI takes only about an hour to reach 100% GPU (and about 45% CPU cores) utilization. This particular run also shows that when ddcMD jobs conclude (small dips in the GPU curve), the corresponding GPUs are immediately reassigned; on two instances, when enough new ddcMD jobs have been started, CG setup runs spike to refill the CG setup buffer.

We remind the reader that these results highlight the utilization of the nodes where the macro model simulation was excluded, and the macro model makes stable use of 100% CPU cores in their 24-core partition, running on 50 to 500 nodes. Furthermore, we emphasize that the variability in resource utilization as illustrated in the results is due to scientific requirements of our RAS campaign. Indeed, MuMMI is fully capable of achieving 100% utilization when required, and provides explicit control to the application for limiting the resources as needed.

7.2 Running at Scale on *Sierra*

We have successfully scaled MuMMI onto 4000 nodes of *Sierra* maintaining the resource utilization described above: 100% GPU utilization and about 45% to 100% CPU utilization based on the state of the simulation. Irrespective of the number of available nodes, our framework creates an ensemble of MD simulations using a single, scalable macro model simulation. Here, we focus on the scaling of the workflow with respect to the size of the simulation, rather than the scaling of the individual components. In particular, we highlight the load sustained by the workflow in terms of the size and temporal sampling of the data.

Macro model. The typical set up of dedicating 24-core partition of 100 nodes to the macro model results in a simulation of 12.0±0.25 µs per day for a 1 µm × 1 µm membrane simulation with 300 RAS molecules. Lipid concentrations are updated every 20 ns, and the RAS particles are integrated with a 25 ps time step. At steady state, the macro model outputs one snapshot every 150 seconds (wall time) with binary and ASCII files totaling 105 MB.

ML selection. The encoding of new patches using ML is done in real-time, and the cost of maintaining the in-memory priority queue as well as approximate-nearest-neighbor queries is negligible. It takes 14±2 seconds (with occasional instances taking up to 26 seconds) to re-evaluate the importance of about 50,000 candidate patches. To maintain computational affordability, we truncate the queue to a length of 50,000.

CG setup, simulation, and analysis. To instantiate a 30 nm × 30 nm patch from the macro model to a CG simulation, one bilayer leaflet is fixed at 1600 lipids, and the complementary leaflet is created to embody the necessary asymmetric bilayer, allowing for some variation in the total number of lipids between patches. The total lipid number in each simulation is 3070±82 and with protein(s), water, and ions totaling ∽140,000 particles. Each CG setup takes 1.5±0.1 hour (using 24 CPU cores with 4 hardware threads each). With a time step of 20 fs, ddcMD simulates 1.04 µs per day running on 1 GPU and 1 CPU core on a fully loaded node. Saved binary particle positions take ∽3.1 MB per snapshot and are generated locally every 0.5 ns (∽42 seconds wall time). For each simulation, a corresponding in situ analysis module (using 3 CPU cores) processes the data locally. Analyzed data, snapshots for offline analysis (every 2 ns), restart checkpoints, and log files are moved to GPFS every 20 ns (∽30 minutes wall time).

Throughput Variability. The summarized statistics for MuMMI are as follows:

- Macro model simulation: 12.0±0.25 µs per day using 2400 MPI tasks
- ML patch queue latency: 14±2 s per 50,000 queries
- CG setup: 1.5±0.1 hours per patch
- ddcMD performance: 1.04±0.01 µs per day

For the campaign mentioned in this paper, the goal is to run each selected micro patch for at least 1 µs. The MuMMI workflow is throughput-limited due to the length of time required to run the micro simulations, with ddcMD requiring a full day to reach the minimum run time. With forthcoming improvements in the speed of ddcMD, we expect to see a corresponding increase in overall throughput of MuMMI. During the 6 to 24 hour long runs performed for this campaign, on average 81%–98% of available GPU resources on *Sierra* were utilized. Periods of resource underutilization are due to simulation loading at the beginning of runs, initial buildup of setup patches (during an initial run), and >0.2% for turnover when simulations are stopped and new ones started. Without any resource conflicts, ddcMD is capable of producing 1.08±0.01 µs per day. During our campaign, on a fully loaded machine, over 80% of the time simulation throughput was 1.04±0.01 µs per day (5.6% less than the ideal). Of the remaining time/simulations, ∽10% experienced a slow down in output rate of about two fold and the rest had a broader distribution of throughput times. It should be noted that these runs took place while *Sierra* was in early testing and hardening phase and a number of delays due to hardware and software were observed.

7.3 RAS Campaign on Sierra

In this work, we have used MuMMI to carry out a multiscale simulation of RAS proteins on a model cell membrane at unprecedented spatial and temporal scales, enabling us to directly probe the biologically relevant processes related to cancer initiation. MuMMI efficiently utilized all of *Sierra* (176,000 CPUs and 16,000 GPUs) as well as subsections of the machine. A total of 5.6M GPU hours were used for running ∽120,000 MD simulations. These simulations aggregated over 200 ms of 30 nm × 30 nm MD trajectories representatively sampling a square µm membrane macro simulation with 300 RAS proteins over 150 µs. This volume of simulations and aggregated simulation time represents orders of magnitude increase over traditional simulation campaigns.

In total, the macro model generated ∽2M candidate patches. Of the candidate patches, ∽120,000 were selected by the ML-based importance sampler and simulated at the micro CG MD level. Each 30 nm × 30 nm membrane patch contained one or more RAS molecules and was simulated for a minimum of 1 µs averaging 1.7±0.8 µs (see Figure 5). The integration timestep of the CG MD simulations was 20 fs, requiring over 10^{13} energy evaluations of the 140,000 particle size simulations to reach the aggregated time of over 200 ms. Simulations were analyzed online every 0.5 ns and frames saved for offline analysis very 2 ns, resulting in over 100M stored frames for later analysis occupying 320 TB of disk space. Online analysis was performed for most of the simulations and over 300M frames were processed.

Detailed analysis of both the online-processed data and the saved (complete) trajectories, as well as experimental verification of key simulation are currently underway, and are beyond the scope of this paper. In particular, the majority of standard simulation analysis tools and approaches were not developed to work with such a large quantity data and range of simulated conditions, requiring improvements to existing tools. The results will be presented in forthcoming publications, along with open access to the dataset. Preliminary findings indicate RAS dynamics on the membrane are lipid dependent, showing how both RAS conformational preference and RAS-RAS aggregation is dependent on lipid composition.

8 DISCUSSION AND CONCLUSION

The workflow infrastructure of MuMMI represents a new way to effectively integrate HPC workload situated between the large, embarrassingly-parallel ensembles and the more tightly coupled multi-physics simulations. As demonstrated above, MuMMI can fully and efficiently utilize one of the world's largest supercomputers while running a complex interdependent workflow. Unlike previous efforts, MuMMI employs advanced ML methods to couple different scales to resolve scientific inquiries. This new framework uses a sophisticated workflow to overcome the difficulty of linking simulations at multiple scales using ML and creates a groundbreaking platform producing a multiscale simulation that is orders of magnitude larger than previously reported.

Running MuMMI on all of *Sierra*, especially as one of the earliest applications, required addressing a number of challenges beyond the technical contributions of this paper. To aid with reproducing or extending our approach and to suggest potential areas of future research and system development, we briefly discuss some of these challenges. As mentioned in Section 5.3, managing the I/O load, both in terms of data size as well as inode count, is crucial, and strategies such as in situ analysis can help with mitigation. Furthermore, we employed a careful job layout to allow maximal use of local on-node RAM disk between related jobs to lighten the load on the GPFS filesystem. In the future, we plan to move to a database solution, such as the DBR. Another unexpected bottleneck was the loading of a large number of shared libraries, such as Python modules. Treating all jobs as entirely independent implies an individual start-up and tear-down phase which, especially on a large parallel system, can incur significant overheads as shared libraries are simultaneously loaded by thousands of tasks. Instead, we plan to restructure the workflow to preload all necessary libraries at start-up and avoid repeated I/O by assigning new data to their corresponding tasks without a tear-down and restart. We are investigating using tools like Spindle [28] to avoid this complexity and recover the independence between jobs. However, currently Spindle is designed to load libraries within a single job and does not directly support the large number of independent jobs used by MuMMI. In general, MuMMI is under active development to accelerate the macro model, support additional protein species, and incorporate atomistic MD. Workflow-related extensions include real-time feedback, improved resource allocation, and support for different machine architectures.

The framework presented by MuMMI is generic and can be extended to other application areas such as chemistry, climate science, and materials science, where a similar multiscale implementation is beneficial. A number of other applications share the same characteristics and could potentially benefit from the MuMMI workflow. For example, one might investigate bond switches in atomistic MD simulations by selectively executing quantum mechanics at energy barriers or observe the evolution of hurricanes in a long-range climate model by initiating higher resolution weather models. MuMMI represents a flexible framework not only to easily build such applications but to ensure their scalability on large, heterogeneous HPC architectures of the future. Finally, the seamless integration of diverse components facilitated by MuMMI is an important step moving towards exascale, where the co-design of scalable frameworks that focus on portability, data movement and layout, and performance optimization are key to a sustainable hardware-software ecosystem.

ACKNOWLEDGMENTS

This work has been supported in part by the Joint Design of Advanced Computing Solutions for Cancer (JDACS4C) program established by the U.S. Department of Energy (DOE) and the National Cancer Institute (NCI) of the National Institutes of Health (NIH). For computing time, we thank Livermore Computing (LC) and Livermore Institutional Grand Challenge. For computational support, and incredible perseverance, we thank the LC Sierra team, especially John C. Gyllenhaal and Adam Bertsch. We thank the Flux team for continued and tireless support. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, Los Alamos National Laboratory under Contract DE-AC5206NA25396, Oak Ridge National Laboratory under contract DE-AC05-00OR22725, and Frederick National Laboratory for Cancer Research under Contract HHSN261200800001E. Release LLNL-CONF-771637.

REFERENCES

[1] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilárd Páll, Jeremy C. Smith, Berk Hess, and Erik Lindahl. 2015. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* 1-2 (Sept. 2015), 19–25. https://doi.org/10.1016/j.softx.2015.06.001

[2] Brian M. Adams, Lara E. Bauman, William J. Bohnhoff, Keith R. Dalbey, Mohamed S. Ebeida, John P. Eddy, Michael S. Eldred, Patricia D. Hough, Kenneth T. Hu, John D. Jakeman, J. Adam Stephens, Laura P. Swiler, Dena M. Vigil, and Timothy M. Wildey. 2009. *Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.0 User's Manual.* Sandia National Laboratory.

[3] Dong H. Ahn, Ned Bass, Albert Chu, Jim Garlick, Mark Grondona, Stephen Herbein, Joseph Koning, Tapasya Patki, Thomas R. W. Scogland, Becky Springmeyer, and Michela Taufer. 2018. Flux: Overcoming Scheduling Challenges for Exascale Workflows. In *2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. IEEE, Dallas, TX, USA, 10–19. https://doi.org/10.1109/WORKS.2018.00007

[4] Michael P. Allen and Dominic J. Tildesley. 1989. *Computer Simulation of Liquids*. Clarendon Press, New York, NY, USA.

[5] Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludascher, and Steve Mock. 2004. Kepler: an extensible system for design and execution of scientific workflows. In *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.* IEEE, 423–424. https://doi.org/10.1109/SSDM.2004.1311241

[6] Nojood A. Altwaijry, Michael Baron, David W. Wright, Peter V. Coveney, and Andrea Townsend-Nicholson. 2017. An Ensemble-Based Protocol for the Computational Prediction of Helix–Helix Interactions in G Protein-Coupled Receptors using Coarse-Grained Molecular Dynamics. *Journal of Chemical Theory and Computation* 13, 5 (2017), 2254–2270.

[7] Hans C. Andersen. 1983. Rattle: A "velocity" version of the shake algorithm for molecular dynamics calculations. *The Journal of Computational Physics* 52, 1 (1983), 24–34. https://doi.org/10.1016/0021-9991(83)90014-1

[8] Gary S. Ayton, Will G. Noid, and Gregory A. Voth. 2007. Multiscale modeling of biomolecular systems: in serial and in parallel. *Current Opinion in Structural Biology* 17, 2 (2007), 192–198. https://doi.org/10.1016/j.sbi.2007.03.004

[9] Gary S. Ayton and Gregory A. Voth. 2010. Multiscale simulation of protein mediated membrane remodeling. *Seminars in Cell & Developmental Biology* 21, 4 (June 2010), 357–362. https://doi.org/10.1016/j.semcdb.2009.11.011

[10] Herman J.C. Berendsen, J.P.M. Postma, Wilfred F. van Gunsteren, A. DiNola, and Jan R. Haak. 1984. Molecular dynamics with coupling to an external bath. *The Journal of Chemical Physics* 81, 8 (1984), 3684–3690. https://doi.org/10.1063/1.448118 arXiv:https://doi.org/10.1063/1.448118

[11] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. 2010. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, Vol. 4. Austin, TX.

[12] Tamara C. Bidone, Anirban Polley, Jaehyeok Jin, Tristan Driscoll, Daniel V. Iwamoto, David A. Calderwood, Martin A. Schwartz, and Gregory A. Voth. 2019. Coarse-Grained Simulation of Full-Length Integrin Activation. *Biophysical Journal* 116, 6 (Feb. 2019), 1000–1010.

[13] Hans-Joachim Bungartz, Florian Lindner, Bernhard Gatzhammer, Miriam Mehl, Klaudius Scheufele, Alexander Shukaev, and Benjamin Uekermann. 2016. preCICE – A fully parallel library for multi-physics surface coupling. *Computers & Fluids* 141 (2016), 250–258. https://doi.org/10.1016/j.compfluid.2016.04.003

[14] Timothy S. Carpenter, Peter J. Bond, Syma Khalid, and Mark S.P. Sansom. 2008. Self-assembly of a simple membrane protein: coarse-grained molecular dynamics simulations of the influenza M2 channel. *Biophysical Journal* 95, 8 (2008), 3790–3801.

[15] Timothy S. Carpenter, Cesar A. López, Chris Neale, Cameron Montour, Helgi I. Ingólfsson, Francesco Di Natale, Felice C. Lightstone, and Sandrasegaram Gnanakaran. 2018. Capturing Phase Behavior of Ternary Lipid Mixtures with a Refined Martini Coarse-Grained Force Field. *Journal of Chemical Theory and Computation* 14, 11 (2018), 6050–6062.

[16] Matthieu Chavent, Anna L. Duncan, Patrice Rassam, Oliver Birkholz, Jean Hélie, Tyler Reddy, Dmitry Beliaev, Ben Hambly, Jacob Piehler, Colin Kleanthous, and Mark S.P. Sansom. 2018. How nanoscale protein interactions determine the mesoscale dynamic organisation of bacterial outer membrane proteins. *Nature Communications* 9, 1 (Dec. 2018). https://doi.org/10.1038/s41467-018-05255-9

[17] François Chollet et al. 2015. Keras. https://github.com/fchollet/keras.

[18] Anthony Craig, Sophie Valcke, and Laure Coquart. 2017. Development and performance of a new version of the OASIS coupler, OASIS3-MCT_3.0. *Geoscientific Model Development* 10, 9 (2017), 3297–3308. https://doi.org/10.5194/gmd-10-3297-2017

[19] Tamara L. Dahlgren, David Domyancic, Scott Brandon, Todd Gamblin, John Gyllenhaal, Rao Nimmakayala, and Richard Klein. 2015. Poster: Scaling uncertainty quantification studies to millions of jobs. In *Proceedings of the 27th ACM/IEEE International Conference for High Performance Computing and Communications Conference (SC)*.

[20] Djurre H. de Jong, Svetlana Baoukina, Helgi I. Ingólfsson, and Siewert J. Marrink. 2016. Martini straight: Boosting performance using a shorter cutoff and GPUs. *Computer Physics Communications* 199 (Feb. 2016), 1–7. https://doi.org/10.1016/j.cpc.2015.09.014

[21] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J. Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira Da Silva, Miron Livny, and Kent Wenger. 2015. Pegasus: a Workflow Management System for Science Automation. *Future Generation Computer Systems* 46 (2015), 17–35. https://doi.org/10.1016/j.future.2014.10.008

[22] Francesco Di Natale. 2017. Maestro Workflow Conductor. https://github.com/LLNL/maestrowf.

[23] Carl Doersch. 2016. Tutorial on Variational Autoencoders. *arXiv:1606.05908 [cs, stat]* (June 2016). http://arxiv.org/abs/1606.05908

[24] Florent Duchaine, Stéphan Jauré, Damien Poitou, Eric Quémerais, Gabriel Staffelbach, Thierry Morel, and Laurent Gicquel. 2015. Analysis of high performance conjugate heat transfer with the OpenPALM coupler. *Computational Science & Discovery* 8, 1 (July 2015), 015003. https://doi.org/10.1088/1749-4699/8/1/015003

[25] Anna L. Duncan, Tyler Reddy, Heidi Koldsø, Jean Hélie, Philip W. Fowler, Matthieu Chavent, and Mark S.P. Sansom. 2017. Protein crowding and lipid complexity influence the nanoscale dynamic organization of ion channels in cell membranes. *Scientific Reports* 7, 1 (Dec. 2017). https://doi.org/10.1038/s41598-017-16865-6

[26] Giray Enkavi, Matti Javanainen, Waldemar Kulig, Tomasz Róg, and Ilpo Vattulainen. 2019. Multiscale Simulations of Biological Membranes: The Challenge To Understand Biological Phenomena in a Living Substance. *Chemical Reviews* 119, 9 (March 2019), 5607–5774. https://doi.org/10.1021/acs.chemrev.8b00538

[27] Ernest J. Friedman-Hill, Edward L. Hoffman, Marcus J. Gibson, Robert L. Clay, and Kevin H. Olson. 2015. *Incorporating Workflow for V&V/UQ in the Sandia Analysis Workbench.* Technical Report. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

[28] Wolfgang Frings, Dong H. Ahn, Matthew LeGendre, Todd Gamblin, Bronis R. de Supinski, and Felix Wolf. 2013. Massively Parallel Loading. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing (ICS '13)*. ACM, New York, NY, USA, 389–398. https://doi.org/10.1145/2464996.2465020

[29] Todd Gamblin, Matthew LeGendre, Michael R. Collette, Gregory L. Lee, Adam Moody, Bronis R. de Supinski, and Scott Futral. 2015. The Spack Package Manager: Bringing Order to HPC Software Chaos. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '15)*. ACM, New York, NY, USA, Article 40, 12 pages. https://doi.org/10.1145/2807591.2807623

[30] James N. Glosli, David F. Richards, Kyle J. Caspersen, Robert E. Rudd, John A. Gunnels, and Frederick H. Streitz. 2007. Extending Stability Beyond CPU Millennium: A Micron-scale Atomistic Simulation of Kelvin-Helmholtz Instability. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (SC '07)*. ACM, New York, NY, USA, Article 58, 11 pages. https://doi.org/10.1145/1362622.1362700

[31] Richard Gowers, Max Linke, Jonathan Barnoud, Tyler Reddy, Manuel Melo, Sean Seyler, Jan Domański, David Dotson, Sébastien Buchoux, Ian Kenney, and Oliver Beckstein. 2016. MDAnalysis: A Python Package for the Rapid Analysis of Molecular Dynamics Simulations. Austin, Texas, 98–105. https://doi.org/10.25080/Majora-629e541a-00e

[32] John M.A. Grime, James F. Dama, Barbie K. Ganser-Pornillos, Cora L. Woodward, Grant J. Jensen, Mark Yeager, and Gregory A. Voth. 2016. Coarse-grained simulation reveals key features of HIV-1 capsid self-assembly. *Nature Communications* 7 (2016), 11568.

[33] IBM. 2019. *IBM Job Step Manager (JSM).* IBM Corporation. https://www.ibm.com/support/knowledgecenter/en/SSWRJV_10.1.0/jsm/10.3/base/jsm_kickoff.html

[34] Tsuyoshi Ichimura, Kohei Fujita, Takuma Yamaguchi, Akira Naruse, Jack C. Wells, Thomas C. Schulthess, Tjerk P. Straatsma, Christopher J. Zimmer, Maxime Martinasso, Kengo Nakajima, Muneo Hori, and Lalith Maddegedara. 2018. A Fast Scalable Implicit Solver for Nonlinear Time-evolution Earthquake City Problem on Low-ordered Unstructured Finite Elements with Artificial Intelligence and Transprecision Computing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '18)*. IEEE Press, Piscataway, NJ, USA, 627–637. http://dl.acm.org/citation.cfm?id=3291656.3291722 event-place: Dallas, Texas.

[35] IEEE 1003.1 2013. *Standard for Information Technology—Portable Operating System Interface (POSIX(R)) Base Specifications* (2016 ed.). Standard. Institute of Electrical and Electronics Engineers, New Jersey, USA. Issue 7.

[36] Helgi I. Ingólfsson, Clément Arnarez, Xavier Periole, and Siewert J. Marrink. 2016. Computational 'microscopy' of cellular membranes. *Journal of Cell Science* 129, 2 (Jan. 2016), 257–268. https://doi.org/10.1242/jcs.176040

[37] Helgi I. Ingólfsson, Timothy S. Carpenter, Harsh Bhatia, Peer-Timo Bremer, Siewert J. Marrink, and Felice C. Lightstone. 2017. Computational Lipidomics of the Neuronal Plasma Membrane. *Biophysical Journal* 113, 10 (Nov. 2017), 2271–2280. https://doi.org/10.1016/j.bpj.2017.10.017

[38] Anubhav Jain, Shyue Ping Ong, Wei Chen, Bharat Medasani, Xiaohui Qu, Michael Kocher, Miriam Brafman, Guido Petretto, Gian-Marco Rignanese, Geoffroy Hautier, Daniel Gunter, and Kristin A. Persson. 2015. FireWorks: a dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience* 27, 17 (2015), 5037–5059. https://doi.org/10.1002/cpe.3505 CPE-14-0307.R2.

[39] Morris A. Jette, Andy B. Yoo, and Mark Grondona. 2002. SLURM: Simple Linux Utility for Resource Management. In *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*. Springer-Verlag, 44–60.

[40] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* (2019).

[41] Shina C.L. Kamerlin and Arieh Warshel. 2011. Multiscale modeling of biological functions. *Physical Chemistry Chemical Physics* 13, 22 (2011), 10401–10411.

[42] Kai J. Kohlhoff, Diwakar Shukla, Morgan Lawrenz, Gregory R. Bowman, David E. Konerding, Dan Belov, Russ B. Altman, and Vijay S. Pande. 2014. Cloud-based simulations on Google Exacycle reveal ligand modulation of GPCR activation pathways. *Nature Chemistry* 6, 1 (2014), 15–21.

[43] Redis Labs. 2018. Redis. https://redis.io.

[44] Paul Langevin. 1908. Sur la théorie du mouvement brownien. *Compt. Rendus* 146 (1908), 530–533.

[45] Lawrence Livermore National Laboratory. 2019. *Sierra.* Retrieved August 22, 2019 from https://hpc.llnl.gov/hardware/platforms/sierra

[46] Cesar A. López, Timothy Travers, Klaas M. Pos, Helen I. Zgurskaya, and S. Gnanakaran. 2017. Dynamics of Intact MexAB-OprM Efflux Pump: Focusing on the MexA-OprM Interface. *Scientific Reports* 7, 1 (Dec. 2017). https://doi.org/10.1038/s41598-017-16497-w

[47] Umberto M.B. Marconi and Pedro Tarazona. 1999. Dynamic density functional theory of fluids. *The Journal of chemical physics* 110, 16 (1999), 8032–8044.

[48] Siewert J. Marrink, H. Jelger Risselada, Serge Yefimov, D. Peter Tieleman, and Alex H. de Vries. 2007. The MARTINI Force Field: Coarse Grained Model for Biomolecular Simulations. *The Journal of Physical Chemistry B* 111, 27 (July 2007), 7812–7824. https://doi.org/10.1021/jp071097f

[49] Donald A. McQuarrie. 2000. *Statistical Mechanics.* University Science Books. https://books.google.com/books?id=itcpPnDnJM0C

[50] Manuel N. Melo, Clément Arnarez, Hendrik Sikkema, Neeraj Kumar, Martin Walko, Herman J. C. Berendsen, Armagan Kocer, Siewert J. Marrink, and Helgi I. Ingólfsson. 2017. High-Throughput Simulations Reveal Membrane-Mediated Effects of Alcohols on MscL Gating. *Journal of the American Chemical Society* 139, 7 (Feb. 2017), 2664–2671. https://doi.org/10.1021/jacs.6b11091

[51] Naveen Michaud-Agrawal, Elizabeth J. Denning, Thomas B. Woolf, and Oliver Beckstein. 2011. MDAnalysis: A toolkit for the analysis of molecular dynamics simulations. *The Journal of Computational Chemistry* 32, 10 (July 2011), 2319–2327. https://doi.org/10.1002/jcc.21787

[52] Akshay Mittal, Xiao Chen, Charles Tong, and Gianluca Iaccarino. 2014. A Flexible Uncertainty Quantification Framework for General Multi-Physics Systems. *arXiv e-prints* (Oct. 2014), 1410–5316.

[53] Leonard S. Ornstein and Frits Zernike. 1914. Accidental Deviations of Density and Opalescence at the Critical Point of a Single Substance. *Proceeding of Akademic Science* 17 (1914), 793–806.

[54] Alexander J. Pak, John M. A. Grime, Prabuddha Sengupta, Antony K. Chen, Aleksander E.P. Durumeric, Anand Srivastava, Mark Yeager, John A.G. Briggs, Jennifer Lippincott-Schwartz, and Gregory A. Voth. 2017. Immature HIV-1 lattice assembly dynamics are regulated by scaffolding from nucleic acid and the plasma membrane. *Proceddings of the National Academy of Sciences* 114, 47 (2017), E10056–E10065. https://doi.org/10.1073/pnas.1706600114

[55] Juan R. Perilla, Boon Chong Goh, C. Keith Cassidy, Bo Liu, Rafael C. Bernardi, Till Rudack, Hang Yu, Zhe Wu, and Klaus Schulten. 2015. Molecular dynamics simulations of large macromolecular complexes. *Current Opinion in Structural Biology* 31 (2015), 64–74. https://doi.org/10.1016/j.sbi.2015.03.007

[56] Jayson L. Peterson, Kelli D. Humbird, John E. Field, Scott T. Brandon, Steve H. Langer, Ryan C. Nora, Brian K. Spears, and Paul T. Springer. 2017. Zonal Flow Generation in Inertial Confinement Fusion Implosions. *Physics of Plasmas* 24, 3 (2017), 032702. https://doi.org/10.1063/1.4977912 arXiv:https://doi.org/10.1063/1.4977912

[57] Tyler Reddy and Mark S.P. Sansom. 2016. The Role of the Membrane in the Structure and Biophysical Robustness of the Dengue Virion Envelope. *Structure* 24, 3 (March 2016), 375–382.

[58] Benedict J. Reynwar, Gregoria Illya, Vagelis A. Harmandaris, Martin M. Müller, Kurt Kremer, and Markus Deserno. 2007. Aggregation and vesiculation of membrane proteins by curvature-mediated interactions. *Nature* 447, 7143 (2007), 461.

[59] Lars Schneidenbach, Claudia Misale, Bruce D'Amora, and Carlos Costa. 2019. IBM Data Broker. https://github.com/IBM/data-broker.

[60] Hans M. Senn and Walter Thiel. 2009. QM/MM methods for biomolecular systems. *Angewandte Chemie International Edition* 48, 7 (2009), 1198–1229.

[61] David E. Shaw, Ron O. Dror, John K. Salmon, J.P. Grossman, Kenneth M. Mackenzie, Joseph A. Bank, Cliff Young, Martin M. Deneroff, Brannon Batson, Kevin J. Bowers, Edmond Chow, Michael P. Eastwood, Douglas J. Ierardi, John L. Klepeis, Jeffrey S. Kuskin, Richard H. Larson, Kresten Lindorff-Larsen, Paul Maragakis, Mark A. Moraes, Stefano Piana, Yibing Shan, and Brian Towles. 2009. Millisecond-scale Molecular Dynamics Simulations on Anton. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09)*. New York, NY, USA, 1–11. https://doi.org/10.1145/1654059.1654126

[62] Dhirendra K. Simanshu, Dwight V. Nissley, and Frank McCormick. 2017. RAS Proteins and Their Regulators in Human Disease. *Cell* 170, 1 (June 2017), 17–33. https://doi.org/10.1016/j.cell.2017.06.009

[63] Mijo Simunovic, Anand Srivastava, and Gregory A. Voth. 2013. Linear aggregation of proteins on the membrane as a prelude to membrane remodeling. *Proceedings of the National Academy of Sciences* 110, 51 (Dec. 2013), 20396–20401. https://doi.org/10.1073/pnas.1309819110

[64] James Smith. 2017. *IBM Spectrum LSF.* IBM Corporation. https://www.ibm.com/support/knowledgecenter/en/SSWRJV_10.1.0/lsf_welcome/lsf_welcome.html

[65] Phillip J. Stansfeld and Mark S.P. Sansom. 2011. From coarse grained to atomistic: a serial multiscale approach to membrane protein simulations. *Journal of Chemical Theory and Computation* 7, 4 (2011), 1157–1166.

[66] Frederick H. Streitz, James N. Glosli, and Mehul V. Patel. 2006. Beyond finite-size scaling in solidification simulations. *Physical Review Letters* 96, 22 (2006), 225701. https://doi.org/10.1103/PhysRevLett.96.225701

[67] Frederick H. Streitz, James N. Glosli, Mehul V. Patel, Bor Chan, Robert K. Yates, Bronis R. de Supinski, James Sexton, and John A. Gunnels. 2005. 100+ TFlop Solidification Simulations on BlueGene/L. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (SC '05)*. ACM, New York, NY, USA.

[68] William C. Swope, Hans C. Andersen, Peter H. Berens, and Kent R. Wilson. 1982. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *The Journal of Chemical Physics* 76, 1 (1982), 637–649. https://doi.org/10.1063/1.442716 arXiv:https://doi.org/10.1063/1.442716

[69] The HDF Group. 1997-2019. Hierarchical Data Format, version 5. https://www.hdfgroup.org/HDF5/.

[70] Michael R. Tonks, Derek Gaston, Paul C. Millett, David Andrs, and Paul Talbot. 2012. An object-oriented finite element framework for multiphysics phase field simulations. *Computational Materials Science* 51, 1 (2012), 20–29.

[71] TOP500. 2019. June 2019 | TOP500 Supercomputer Sites. https://www.top500.org/lists/2019/06/

[72] Timothy Travers, Cesar A. López, Que N. Van, Chris Neale, Marco Tonelli, Andrew G. Stephen, and Sandrasegaram Gnanakaran. 2018. Molecular recognition of RAS/RAF complex at the membrane: Role of RAF cysteine-rich domain. *Scientific Reports* 8, 1 (May 2018), 8461. https://doi.org/10.1038/s41598-018-26832-4

[73] Vincent A. Voelz, Marcus Jäger, Shuhuai Yao, Yujie Chen, Li Zhu, Steven A. Waldauer, Gregory R. Bowman, Mark Friedrichs, Olgica Bakajin, Lisa J. Lapidus, Shimon Weiss, and Vijay S. Pande. 2012. Slow Unfolded-State Structuring in Acyl-CoA Binding Protein Folding Revealed by Simulation and Experiment. *Journal of the American Chemical Society* 134, 30 (2012), 12565–12577. https://doi.org/10.1021/ja302528z

[74] Martin Vögele, Jürgen Köfinger, and Gerhard Hummer. 2018. Hydrodynamics of Diffusion in Lipid Membrane Simulations. *Physical Review Letters* 120, 26 (June 2018), 268104. https://doi.org/10.1103/PhysRevLett.120.268104

[75] Gregory A. Voth. 2017. A Multiscale Description of Biomolecular Active Matter: The Chemistry Underlying Many Life Processes. *Accounts of Chemical Research* 50, 3 (March 2017), 594–598. https://doi.org/10.1021/acs.accounts.6b00572

[76] Tsjerk A. Wassenaar, Helgi I. Ingólfsson, Rainer A. Böckmann, D. Peter Tieleman, and Siewert J. Marrink. 2015. Computational Lipidomics with *insane* : A Versatile Tool for Generating Custom Membranes for Molecular Simulations. *Journal of Chemical Theory and Computation* 11, 5 (May 2015), 2144–2155. https://doi.org/10.1021/acs.jctc.5b00209

[77] Tsjerk A. Wassenaar, Kristyna Pluhackova, Anastassiia Moussatova, Durba Sengupta, Siewert J. Marrink, D. Peter Tieleman, and Rainer A. Böckmann. 2015. High-Throughput Simulations of Dimer and Trimer Assembly of Membrane Proteins. The DAFT Approach. *Journal of Chemical Theory and Computation* 11, 5 (May 2015), 2278–2291. https://doi.org/10.1021/ct5010092

[78] Andrew M. Waters and Channing J. Der. 2018. KRAS: The Critical Driver and Therapeutic Target for Pancreatic Cancer. *Cold Spring Harbor Perspectives in Medicine* 8, 9 (Sept. 2018), a031435. https://doi.org/10.1101/cshperspect.a031435

[79] Paul C. Whitford, Scott C. Blanchard, Jamie H. D. Cate, and Karissa Y. Sanbonmatsu. 2013. Connecting the Kinetics and Energy Landscape of tRNA Translocation on the Ribosome. *PLOS Computational Biology* 9, 3 (March 2013), 1–10. https://doi.org/10.1371/journal.pcbi.1003003

[80] Isseki Yu, Takaharu Mori, Tadashi Ando, Ryuhei Harada, Jaewoon Jung, Yuji Sugita, and Michael Feig. 2016. Biomolecular interactions modulate macromolecular structure and dynamics in atomistic model of a bacterial cytoplasm. *eLife* 5 (Nov. 2016), e19274. https://doi.org/10.7554/eLife.19274

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

We tested molecular dynamics codes ddcMD (version 2019.1) and GROMACS (version 2019.1) on LLNL's Sierra supercomputer using the IBM Spectrum MPI (rolling-release). The tests benchmarked a CUDA-enabled ddcMD GROMACS using various numbers of GPU and CPU core resources (as described in the paper). NVIDIA CUDA library with version 9.2.88 were used. The performance of each combination of GPU and CPU cores was averaged from 10 duplicated runs. For the main simulation campaign we successfully tested MOOSE and ddcMD of the same source code on several other Linux platforms: an x86 workstation, and 4 different clusters with IBM POWER8 and POWER9 cores.

ARTIFACT AVAILABILITY

Software Artifact Availability: Some author-created software artifacts are NOT maintained in a public repository or are NOT available under an OSI-approved license.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: Some author-created data artifacts are NOT maintained in a public repository or are NOT available under an OSI-approved license.

Proprietary Artifacts: There are associated proprietary artifacts that are not created by the authors. Some author-created artifacts are proprietary.

List of URLs and/or DOIs where artifacts are available:

`https://github.com/LLNL/maestrowf`
`https://github.com/XiaohuaZhangLLNL/mdanalysis`

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: Sierra, Nvidia Tesla v100, IBM POWER9 CPUs, GPFS

Compilers and versions: Python v2.7.15, Theano v1.0.2, gcc v4.9.3

Applications and versions: GROMACS v5.1.4, Moose (commit 072243866cc40abba97295729fc4ea838dda95dc), Flux (commit 70bf1098579d585db1b36e02deb18f7abdb8949f), MDAnalysis v0.16.2, ddcMD v2019.1, maestrowf v1.1.4dev1.1

Libraries and versions: IBM Spectrum MPI v10.2.0.11rtm2, NVIDIA CUDA version 9.2.88, OpenMPI 2.0.0

Paper Modifications: We modified MDAnalysis to parse the ddcMD native binary and ASCII data format. A new format named "DDCMD" is added to the trajectory reader of MDAnalysis. By doing so, the ddcMD output data has been integrated seamlessly with the MDAnalysis package.

For the Macro model simulation we used a modified version of the MOOSE finite element software. Our modifications were incorporated into the git commit 072243866cc40abba97295729fc4ea838dda95dc (Dec 14 18:34:56 2016), cloned from https://github.com/idaholab/moose.git. Our modifications consist of adding modules to the phase_field module: 1) a 'userobject' (SolutionDump) that grabs the current solution and exports it to a file for use by the particle integrator (ddcMD). This object also reads particle positions from ddcMD. 2) a 'userobject' (SolutionGrab) that implements the non-local lipid-lipid convolution functions. 3) a 'kernel' 'LipidMembrane' that implements the lipid-lipid interactions (using data from SolutionGrab). 4) a 'kernel' (HycopTermTable) that implements the lipid-particle interactions. 5) Modifications to the ConservativeNoise kernel to allow more control from the input script and timestep scaling.

ARTIFACT EVALUATION

Verification and validation studies: We need to cover all of the applications we tested. We need to describe the environment GROMACS (version 2019.1): Ran performance studies on Sierra, compiled with gcc v4.9.3, IBM Spectrum MPI v10.2.0.11rtm2, and NVIDIA CUDA version 9.2.88. A 135,973-bead Martini simulation of a single KRAS protein on an asymmetric 8-component, 3,077 lipid bilayer was used as the test case. The benchmark calculation is carried out on a Sierra node using different combination of GPU and CPU cores. There are four different categories of jobs in term of number of GPUs: 1-GPU, 2-GPU, 3-GPU, and 4-GPU per node jobs. Note that multiple GPUs are used separately for MD simulations. In other word, each MD simulation use 1 GPU. Each category of jobs ran with 1 to 8 CPU cores. Each combination of GPU and CPU cores were run for 10 trials each. A total of 4*8*10 = 320 MD simulations was performed on Sierra. The performance was calculated for each combination by averaging the accumulated time (us/day) of the MD trajectories from all 10 runs.

ddcMD (version 2019.1): Ran performance studies on Sierra, compiled with gcc v4.9.3, IBM Spectrum MPI rolling release version, and NVIDIA CUDA version 9.2.88. The same test case as GROMACS was used in the ddcMD benchmark calculation. Since ddcMD has offloaded the entire MD loop to the GPU, only one CPU core is needed for each GPU simulation. The benchmark calculation was carried out on a Sierra node using 1 GPU and 1 CPU. Using the same approach as GROMACS, we run four different categories of jobs in terms of number of GPUs: 1-GPU, 2-GPU, 3-GPU, and 4-GPU per node jobs. For 1-GPU job we run 1 MD simulation using 1 GPU and 1 CPU. For 2-GPU job we run 2 MD simulations that each MD simulation using 1 GPU and 1 CPU, and so on.

MOOSE: For the main simulation campaign MOOSE and ddcMD were compiled with gcc 4.9.3 and using OpenMPI 2.0.0, and run on Linux Intel x86 clusters (Surface and Quartz at LLNL). We have also run successful tests of the same source code on several other Linux platforms: an x86 workstation, and 4 different clusters with IBM POWER8 and POWER9 cores. Tests also included checking subsets of kernels against known solutions.

Quantified the sensitivity of results to initial conditions and/or parameters of the computational environment: We made use of Spack to maintain a consistent environment that was loaded before testing and execution.